YUNIBESITI YA BOKONE-BOPHIRIMA
NORTH-WEST UNIVERSITY
NOORDWES-UNIVERSITEIT

'N ONDERSOEK NA EN BYDRAES TOT NAVRAAGHANTERING EN -OPTIMERING DEUR DATABASISBESTUURSTELSELS.

L. MULLER Hons. B.Sc.

Verhandeling voorgelê ter gedeeltelike nakoming van die vereistes vir die graad Magister Scientiae in Rekenaarwetenskap aan die Noordwes-Universiteit

Studieleier: **Prof. T Steyn**

November 2006

Potchefstroom

DANKBETUIGINGS

'n Spesiale woord van dank word gerig aan die volgende persone wat betrokke was by die verhandeling:

Ek wil eerstens die Here, my God, bedank vir die insig en deursettingsvermoë wat Hy my gegun het om die verhandeling tot die einde toe deur te sien.

Professor Tjaart Steyn vir die hulp en bystand wat hy verleen het as projekleier.

Laastens wil ek graag my familie en vriende bedank vir hul geduld en bystand gedurende die hele proses.

OPSOMMING EN SLEUTELTERME

Titel: 'n Ondersoek na en bydraes tot navraaghantering en -optimering deur databasisbestuurstelsels.

Die probleem om databasisse effektief te ontwerp en te gebruik, raak al groter. Die inligting wat die databasis bevat raak meer kompleks en die betrokke data se grootte veroorsaak spasieprobleme. Die tegnologie moet voortdurend ontwikkel om by die toenemende aanvraag aan te pas. Ondersoek is ingestel om effektiewe riglyne te ontdek wat navrae in die algemeen ten opsigte van werkverrigting en produktiwiteit kan optimeer. Twee databasisbestuurstelsels is ondersoek om die teoretiese aspekte te vergelyk met die tegnieke wat in die praktyk gebruik word.

Microsoft SQL Server en MySQL is as die kandidaat-databasisbestuurstelsels gekies en beide was onderhewig aan 'n deeglike ondersoek. Die stelsels is ondersoek om die metode te bepaal waarmee elkeen die betrokke navraag hanteer. Die navraagoptimeerder vorm die basis vir elk van die stelsels en hanteer die ontleding en verwerking van enige navraag. Elke stelsel se metodes vir die stoor van die data is ondersoek.

Die onderskeie stelsels se hantering van tabelverbindings, gebruik van indekse en die keuse van optimale uitvoerplanne is ondersoek. Aangepaste algoritmes word vir verskeie indeksprosesse soos byvoorbeeld "B+"-bome en "hash"-indekse getoon.

Riglyne wat onafhanklik van die databasisbestuurstelsels gebruik kan word, is saamgestel wat help om relasionele databasisse te optimeer. Verder is 'n paar praktiese implementerings van navrae gedoen om die uitvoerplanne vir die betrokke navrae vir beide MySQL en SQL Server te noteer. Die plan is vir elke stelsel (saam met 'n paar ander veranderlikes soos byvoorbeeld die uitvoertyd) beskryf. 'n Model is vir beide databasisbestuurstelsels in hierdie eksperiment gevolg.

Sleutelterme: Optimering, MySQL, Microsoft SQL Server, navrae, indeks, relasionele databasis, navraagoptimeerder, uitvoerplan.

ABSTRACT AND KEY TERMS

Title: An investigation into and contributions to query handling and query optimization by database management systems.

The problems associated with the effective design and uses of databases are increasing. The information contained in a database is becomming more complex and the size of the data is causing space problems. Technology must continually develop to accommodate this growing need. An inquiry was conducted in order to find effective guidelines that could support queries in general in terms of performance and productivity. Two database management systems were researched to compare die theoretical aspects with the techniques implemented in practice.

Microsoft SQL Server and MySQL were chosen as the candidates and both were put under close scrutiny. The systems were researched to uncover the methods employed by each to manage queries. The query optimizer forms the basis for each of these systems and manages the parsing and execution of any query. The methods employed by each system for storing data were researched.

The way that each system manages table joins, uses indices and chooses optimal execution plans were researched. Adjusted algorithms were introduced for various index processes like B+ trees and hash indexes.

Guidelines were compiled that are independent of the database management systems and help to optimize relational databases. Practical implementations of queries were used to acquire and analyze the execution plan for both MySQL and SQL Server. This plan along with a few other variables such as execution time is discussed for each system. A model is used for both database management systems in this experiment.

Key terms: Optimization, MySQL, Microsoft SQL Server, queries, index, relational database, query optimizer, execution plan.

INHOUDSOPGAWE

INHOUDSOPGAWE (VERVOLG)

INHOUDSOPGAWE (VERVOLG)

INHOUDSOPGAWE (VERVOLG)

HOOFSTUK 1

PROBLEEMSTELLING, DOELWITTE EN METODE VAN ONDERSOEK

1.1 Inleiding

"The amount of information available to us is literally exploding, and the value of data as an organizational asset is widely recognized. To get the most out of their large and complex datasets, users require tools that simplify the tasks of managing the data and extracting useful information in a timely fashion." (Ramakrishnan & Gehrke, 2003:3.)

Die idee van 'n optimale databasis verwys na 'n databasis wat die beste moontlike werkverrigting op alle vlakke vir die gebruiker bied. Normaalweg is hierdie optimaliteit hoogstens 'n strewe en nie 'n realiteit nie. Ramakrishnan en Gehrke (2003:4) maak die stelling dat die groot hoeveelhede data baie maklik 'n las raak indien daar nie van kragtige en buigsame databasisbestuurstelsels gebruik gemaak word nie. Ramakrishnan en Gehrke (2003:7) koppel databasisse en databasisbestuurstelsels se groei verder aan die koms van die internet. Hulle beweer dat bladsye wat deur webblaaiers (byvoorbeeld "Internet Explorer" en "Mozilla") verwerk word, al hoe meer van databasisse gebruik maak om inligting te vertoon.

In hierdie hoofstuk word die probleemstelling, doelwitte en die metode van ondersoek bespreek wat in die verhandeling gevolg word. Paragraaf 1.2 bevat die probleemstelling en motivering en paragraaf 1.3 beskryf die doelstelling en doelwitte. Paragraaf 1.4 bespreek kortliks die metode van ondersoek en paragraaf 1.5 verskaf 'n opsomming van die verhandeling se samestelling.

1.2 Probleemstelling en motivering

In aansluiting by die aanhaling van Ramakrishnan en Gehrke hierbo skryf Chen (2006:413) die volgende: "In the modern business world, the success of an organization depends increasingly on information. We argue that the ability of an organization to obtain information in time to make crucial decisions depends heavily on the "agility" of the organization's database system. That is, database systems need to quickly respond and adapt to changes in today's dynamic environment". Die werkverrigting van 'n databasis is dus 'n krities belangrike faktor ten opsigte van finansies en tydsbesteding van enige besigheid en die vermoë om groot hoeveelhede data akkuraat te stoor en in 'n aanvaarbare tyd in bruikbare vorm terug te kry, kan aan enige besigheid 'n mededingende voordeel bied.

Die vereistes van databasisse vermeerder soos wat die data wat gestoor moet word, vermeerder. Chen (2006:413) stel dat bestaande databasisse met bestaande ontwerpe nie altyd in vandag se behoeftes kan voorsien nie. Die behoeftes raak meer spesifiek soos wat die aantal gebruikers toeneem. Die nodigheid het ontstaan vir voortdurende navorsing in optimeringstegnieke om by te bly met die veranderings wat ten opsigte van inligting en tegnologie plaasvind. Chen (2006:413) stel die volgende: "Traditionally, database design mainly focuses on modelling data in a way that is non-redundant and well structured. In order to improve performance of databases, many techniques such as query optimization, database tuning, and adaptive query processing have been proposed." Verskeie ondersteunende hulpmiddels word voortdurend ontwerp en geïmplementeer, soos byvoorbeeld Microsoft se Query Analyzer (sien paragraaf 3.2.8.2) en MySQL se "EXPLAIN"-bevel (MySQL AB, 2005:425-434).

Die uitdaging is dus om tussen die meervoudige prosesse en tegnieke, vir sowel databasisse as navrae, te onderskei en die beste moontlike kandidaat te verkry. Die hulp wat die databasisbestuurstelsel se interne navraagoptimeerder in hierdie verband kan lewer, moet in ag geneem word. Daar moet gepoog word om prosesse te volg om die databasis en die navrae op die lange duur vir 'n verskeidenheid van inligtingbehoeftes te optimeer en te vereenvoudig.

1.3 Navorsingsdoelstellings en -doelwitte

Die doel van hierdie verhandeling is om 'n databasis, databasisbestuurstelsels, navrae en die onderskeie elemente van hierdie aspekte te bestudeer, te noteer en om dan praktiese tegnieke aan die hand te doen waarvolgens die onderskeie elemente optimaal gebruik kan word.

Vir twee gekose databasisbestuurstelsels sal die proses ondersoek word waarmee by 'n optimale oplossing uitgekom word. Omdat die terrein van databasisse en die optimering daarvan so wyd is (par. 1.2), word daar hoofsaaklik gefokus op navraagoptimering deur middel van indekse om die optimering van werkverrigting te ondersoek. Die navorsing fokus dus hoofsaaklik op die beskikbare literatuur oor die twee databasisbestuurstelsels, nl. MySQL 4.1 (wat die oopbronkode-afdeling verteenwoordig) en Microsoft SQL Server 2000 (wat die kommersiële afdeling verteenwoordig)[1].

In die eksperimentele gedeelte word aan die hand van 'n databasis (uit die praktyk) eksperimentele werk gedoen ten opsigte van die definiëring van verskillende indekse en indeks-

[1] MySQL 5.0 en SQL Server 2005 het tydens die ondersoek beskikbaar geraak en die besluit is geneem om steeds met MySQL 4.1 en SQL Server 2000 te volstaan.

implementerings om werkverrigting van die twee gekose databasisbestuurstelsels te ondersoek. Die doel is dus om, gebaseer op die literatuurstudie (volgens teoretiese oorsprong en produk spesifieke dokumentasie), eksperimentele werk te doen en dan die resultate saam te vat in riglyne wat kan lei tot beter keuses van databasisbestuurstelseltegnieke en navraagtegnieke wat 'n databasisopstelling met 'n hoër produktiwiteit moontlik kan maak.

Die hoofdoel van die verhandeling is nie om verskillende databasisbestuurstelsels met mekaar te vergelyk nie, maar eerder om elke databasisbestuurstelsel se tegnieke en struktuur te ondersoek, die beste optimeringstegniek vir elke databasisbestuurstelsel uit te wys, algemene optimeringstegnieke te bespreek en praktiese voorbeelde te gebruik om die keuses en gevolge van indekse te bespreek.

1.4 Metode van ondersoek

Volgens Oates (2006) dui 'n (navorsings)paradigma op 'n patroon of model of gedeelde manier van dink. Die paradigma wat die naaste aan die navorser se navorsingsmetodiek val, is logiese positivisme. Oates (2006:283) stel dat positivisme te make het met die wetenskaplike metode. Die metode maak twee aannames, naamlik dat die wêreld geordend is en dat navorsers dit objektief kan ondersoek. Eksperimentering is een van die hoofkenmerke van positivisme (Oates, 2006:284), en die verhandeling is hoofsaaklik hierop gebaseer. Daar is spesifiek gekyk na die tegniek van herhaling (Oates, 2006:285) om die resultate se akkuraatheid te verbeter.

Die bronne wat vir die doeleindes van die verhandeling gebruik is, weerspieël tot 'n groot mate die aard van die onderwerp wat onderneem is. Dus is verskeie handboeke, internetbronne en wetenskaplike artikels gebruik wat direk aansluit by die fokus van die verhandeling ten opsigte van die gekose databasisbestuurstelsels, optimering in die algemeen en meer spesifiek die rol van indekse tydens of ter ondersteuning van optimering. Die meeste artikels en bronne wat beskikbaar is op navorsingswebbladsye soos ScienceDirect, Ebscohost en Engineering Village fokus op spesifieke areas van optimering (soos nuwe algoritmes) en nuwe tegnologieë (soos parallelle navraagverwerking en verspreide navraagoptimering). Die onderwerp van navraagoptimering is dus omvangryk en die gekose fokus 'elimineer' baie van die beskikbare bronne.

Die databasisbestuurstelsels vir die ondersoek is MySQL 4.1 (wat die oopbronkode-afdeling verteenwoordig) en Microsoft SQL Server 2000 (vir die kommersiële afdeling). Elkeen van hierdie twee stelsels (SQL Server en MySQL) se interne struktuur en optimeringsteorieë is sorgvuldig ondersoek. Met die oog op maklike hersiening en bywerking is die moderne neiging om die tegniese beskrywing van produkte via die internet beskikbaar te stel. Daarom is die

besluit geneem om grootliks op Microsoft en MySQL AB se eie literatuur te steun ten spyte van die (oënskynlik) nie-akademiese aard daarvan. Die aanname word gemaak dat hierdie twee maatskappye hulle eie produkte die beste sal ken. Die literatuur word, waar moontlik, aangevul uit akademiese/wetenskaplike bronne. Die Afrikaanse terme wat in hierdie verhandeling gebruik is, is gebaseer op die Stigting vir Bemagtiging deur Afrikaans (2001) se terme. Die verhandeling word in die volgende punte saamgevat:

- Die terme wat gebruik word, word bespreek sodat daar nie dubbelsinnigheid voorkom nie. 'n Bylaag word bygevoeg wat sekere terme se vertalings in Engels en kort beskrywings van die terme lys.
- Die spesifieke stoorenjins van die twee databasisbestuurstelsels word ondersoek. Die spesifieke data- en indekslêers word ook bestudeer.
- Daar moet onderskei word tussen die verskillende stelsels se metodes van optimering. 'n Ondersoek word ingestel vir moontlike veralgemening tussen die tegnieke van die databasisbestuurstelsels wat ondersoek word. Verskeie algoritmes word beskryf vir die soek-, bywerk- en verwyderbewerkings wat moontlik vir indekse gebruik kan word.
- Die interne argitektuur van die onderskeie databasisbestuurstelsels word ondersoek.
- Elke databasisbestuurstelsel se navraagoptimeerder word so volledig as moontlik beskryf.
- Die onderskeie databasisbestuurstelsels besit optimeringsfunksies wat gebruik kan word as 'n basis vir hierdie navorsing. Normaalweg laat databasisbestuurstelsels toe dat verskillende optimeringstegnieke gebruik word vir die navrae wat ondersoek word. Verskeie optimeringstegnieke word in die verhandeling bespreek.
- Die verskillende toepassings wat elke databasisbestuurstelsel bied, word kortliks beskryf aangesien sommige van die toepassings tydens die praktiese fase gebruik word.
- Toegang is verkry tot 'n databasis van werklike data en substansiële grootte aangesien die verwerkingspoed van moderne rekenaars nie meetbare uitvoertye vir 'n klein databasis sal lewer nie. Die opstelling van die databasis is staties aangesien bywerking nie oor 'n tydperk op die databasis toegepas word nie. Die effek van indekse word met die praktiese eksperiment getoon. Die navrae word individueel geloop en bespreek. 'n Relatief "klein" rekenaar (sien paragraaf 5.2) word vir die praktiese gedeelte gebruik. Die strategie wat in die verhandeling gevolg is, is Oates (2006:35) se eksperimentele strategie waar die fokus lê op metings vóór 'n spesifieke aksie en metings na afloop van die aksie.
- 'n Model vir optimering word geformuleer en in die eksperimentele gedeelte gebruik.
- Die navrae wat as maatstaf dien, moet ter wille van volledigheid uit algemene sowel as bronintensiewe navrae bestaan. Die navrae word met verskeie indekskombinasies getoets om die mees praktiese en optimale kombinasies te verkry.
- Resultate van elke navraag word noteer en teen mekaar opgeweeg om die indeks vir die onderskeie databasisbestuurstelsels en die navrae self uit te wys.

- Ter samevatting word die navrae almal saam met verskeie bywerk- en byvoegbewerkings volgens die sogenaamde SQL Profiler van SQL Server ondersoek om te sien watter indekse die toepassing voorstel. Die individuele navrae se indekse word met die gesamentlike voorstelling vergelyk om verskille te noteer.

Die volgende paragraaf beskryf die hoofstukindeling van die verhandeling.

1.5 Hoofstukindeling

Hierdie verhandeling bestaan uit 'n literatuurstudie van die twee databasisbestuurstelsels, naamlik SQL Server en MySQL. 'n Teoretiese beskrywing van indekse as 'n optimeringstegniek word ingesluit. Verskeie optimeringstegnieke word bespreek, daarna word indekse op verskeie navrae toegepas in 'n eksperimentele gedeelte, waarna die resultate gemeet word. Die verskillende hoofstukke word soos volg ingedeel:

- Hoofstuk 2 – Die hoofstuk bespreek kortliks die terme "databasis" en "databasisbestuurstelsel" as inleiding tot die res van die verhandeling. Indekse word as 'n belangrike vorm van optimering in hierdie hoofstuk bespreek.
- Hoofstuk 3 – Die databasisbestuurstelsels word in hierdie hoofstuk bespreek. Verskeie aspekte (onder andere die interne argitektuur) word vir elke databasisbestuurstelsel beskryf. Die navraagoptimeerder word breedvoerig vir beide SQL Server en MySQL bespreek. Verskeie toepassings en nuttige programme (soos byvoorbeeld 'n program wat indekse vir SQL Server se tabelle voorstel) word vir elke databasisbestuurstelsel beskryf.
- Hoofstuk 4 – Die hoofstuk bespreek algemene tegnieke om die databasis te optimeer.
- Hoofstuk 5 – Die hoofstuk bespreek kortliks die opstelling van die bediener vir die eksperimentele gedeelte van die verhandeling.
- Hoofstuk 6 – Die eksperimentele resultate van verskeie navrae word bestudeer. Die navrae word met en sonder indekse op elke databasisbestuurstelsel uitgevoer. Die uitwerking van die kasgeheue word vir elke navraag ondersoek.
- Hoofstuk 7 – Die resultate word in hierdie slothoofstuk saamgevat.
- Bylaag A – Die bylaag bespreek kortliks die SQL-taal van SQL Server.
- Bylaag B – Die bylaag bevat 'n lys van Afrikaanse en Engelse terme.
- Bylaag C – Die SQL-instruksies wat gebruik word om die tabelle vir die eksperimentele gedeelte te skep, word hier gelys.

1.6 Slotparagraaf

Hierdie hoofstuk het kortliks die aspekte wat in die verhandeling ondersoek word en die metode van ondersoek bespreek. Die onderskeie hoofstukke is kortliks bespreek om aan die leser 'n aanduiding te gee van die inhoud van die verhandeling. Die volgende hoofstukke bespreek die databasisbestuurstelsels, indekse, optimeringstegnieke en die eksperimentele resultate wat verkry is.

HOOFSTUK 2

DATABASISSE, DATABASISBESTUURSTELSELS EN INDEKSE

2.1 Inleiding

Databasisse het normale lêer-invoer- en -uitvoerbewerkings as die aanvaarde medium van dataverkryging en datahantering vervang. Volgens No *et al.* (2003:434) bied lêerbewerkings hoë werkverrigting, maar kan dit nie groot hoeveelhede en komplekse data hanteer nie. Databasisse voldoen weer aan hierdie vereistes, maar kan nie altyd aan grootskaalse wetenskaplike projekte se werkverrigtingsvereistes voldoen nie. Databasisse voldoen egter aan die vereistes van die meeste ander projekte wat nie so verwerkingsintensief soos wetenskaplike projekte is nie. Die hoofstuk definieer databasisse en databasisbestuurstelsels sodat 'n oorsig oor die terme verkry kan word. Indekse word kortliks as een van die belangrikste optimeringstegnieke bespreek voordat daar na elkeen van die gekose databasisbestuurstelsels in Hoofstuk 3 gekyk word.

Paragraaf 2.2 verskaf 'n paar definisies vir die term "databasis", en paragraaf 2.3 'n paar definisies vir die term "databasisbestuurstelsel" soos dit in die praktyk gebruik word. Die definisies word in die onderskeie paragrawe saamgevat. Paragraaf 2.4 beskryf die teorie rakende indekse.

2.2 'n Paar definisies van die term "Databasis"

"A database consists of some collection of persistent data that is used by the application systems of some given enterprise." (Date, 1995:9.)

"A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning." (Elmasri & Navathe, 1994:2.)

"A *database is a collection of data, typically describing the activities of one or more related* organizations." (Ramakrishnan & Gehrke, 2003:4.)

"A database is a shared, integrated computer structure that stores a collection of:

- End user data, that is, raw facts of interest to the end user.
- Metadata, or data about data, through which the data are integrated and managed."

(Rob & Coronel, 2007:6.)

"In a relational database, the data is represented in tables, called relations. In a table, the columns are the attributes of entities and all values of an entity are stated in a row, also called a tuple." (Du & Wolfe, 1997:812.)

Dit blyk dus dat die term "databasis" nie presies dieselfde betekenis vir verskillende instansies en persone besit nie. Om die verduideliking van die volgende hoofstukke te vergemaklik, word die definisies in hierdie paragraaf saamgevat.

'n Relasionele databasis kan beskryf word as:

- 'n Versameling van data.
- Die data is verwant aan mekaar; die data is dus eienskappe van een of ander objek. 'n Objek stel enige entiteit in die wêreld, soos byvoorbeeld 'n persoon of 'n motor, voor.
- Die data kan deur spesiale instruksies (navrae) of deur 'n databasisbestuurstelsel gemanipuleer word.
- Die databasis se struktuur behels tabelle, met verskeie velde of kolomme (wat getalle, teks, datums of selfs figure kan bevat). Elke tabel bevat 'n aantal rekords (of rye) wat in die onderskeie kolomme onderverdeel is. Die tabel *Werknemer* in figuur 2.1 toon drie rekords wat elk bestaan uit *nommer-, naam-, telefoon-, selfoon-* en *salaris*-kolomme.

Figuur 2.1 - Tabel *Werknemer*

	Nommer	Naam	Telefoon	Selfoon	Salaris
1	1	L Muller	011 2951564	073 231 2645	3500.00
2	2	M Pretorius	018 365 4955	082 322 6248	5000.00
3	3	G Geertsema	056 655 1321		8000.00

Relasionele databasisse word dikwels met sigblaaie vergelyk. Witkowski *et al.* (2005:84-85) dui egter duidelik die verskille aan tussen die twee produkte en stel 'n vermenging voor waar databasisse soortgelyke funksionaliteite as sigblaaie kan verkry. Die grootste verskil is die feit dat databasisse nie gemaklik berekenings op rye soos by sigblaaie kan toepas nie en dat sigblaaie weer net tweedimensionele bewerkings toelaat en nie groot hoeveelhede data kan hanteer nie. 'n Ondersoek na die vermenging val egter buite die bestek van hierdie verhandeling.

2.3 'n Paar definisies van die term "databasisbestuurstelsel"

"A database management system, or DBMS, is software designed to assist in maintaining and utilizing large collections of data." (Ramakrishnan & Gehrke 2003:4.)

"A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities or as parts of other applications." (MySQL AB, 2005:5.)

"In a sense, a database resembles a very well-organized electronic filing cabinet in which powerful software, known as a database management system, helps manage the cabinet's contents. A database management system (DBMS) is a collection of programs that manages the database structure and controls access to the data stored in the database." (Rob & Coronel, 2007:6.)[2]

"Between the physical database itself (i.e., the data as actually stored) and the users of the system is a layer of software, the database manager (DB manager) or, more usually, database management system (DBMS). All requests from users for access to the database are handled by the DBMS; ... One general function provided by the DBMS is thus the shielding of database users from hardware-level details." (Date, 1995:7.)

'n Databasisbestuurstelsel (DBBS) is dus baie kragtige sagteware wat toelaat dat die data in verskeie databasisse gemanipuleer kan word. Die stelsel groepeer data saam in logiese eenhede wat vervat is in databasisse en dan onderverdeel is in tabelle en skerm gewone gebruikers af van die tegniese detail van die gestoorde data in die databasis. Die stelsel bevat een of meer stoorenjins as kern en ondersteun verskeie analitiese en navraag funksionaliteite. Die stoorenjin en navraagverwerker interpreteer en optimeer die gebruiker se bevele en manipuleer dan die data ooreenkomstig. Verskeie moontlikhede bestaan (sien ook Date, 1995:39):

- 'n Nuwe databasis kan geskep word, die struktuur van die databasis kan verander word, of die databasis kan verwyder word.
- Die databasis se data kan gemanipuleer word deur byvoorbeeld data by te voeg, data te verander, en selfs data te verwyder.
- Netwerkverbinding word deur die stelsel hanteer sodat verskeie gebruikers gelyktydig toegang tot die data kan verkry.
- Sekuriteit kan toegepas word op die databasis en toegang tot die data kan hiermee beheer word. 'n Voorbeeld hiervan is dat sekere gebruikers slegs lees-toegang tot tabelle verkry.

[2] Die datum van hierdie handboek is wel 2007. Die boek is beskikbaar gestel aan die navorser van hierdie verhandeling.

- Indekse kan op tabelle geskep word in 'n poging om werkverrigting te verbeter.
- Toepassings bestaan waarmee statistiese inligting oor die databasis verkry kan word, en hierdie inligting kan gebruik word om die databasis te optimeer.
- Rugsteun en data-duplisering is verdere moontlikhede.

Die twee databasisbestuurstelsels (MySQL en Microsoft SQL Server) wat in die verhandeling bespreek en ondersoek word, bied ander funksionaliteite, wat in die volgende hoofstukke ondersoek word. Hoofstuk 3 vereis egter 'n basiese begrip van indekse; daarom word die teorie van indekse in die volgende paragraaf kortliks bespreek.

2.4 Indekse

"So we see that a very simple change in the execution algorithm – doing a restriction and then a join, instead of a join and then a restriction – has produced a dramatic improvement in performance. And the improvement would be more dramatic still if relation SP were indexed or hashed on P# -- the number of tuples read in Step 1 would be reduced from 10,000 to just 50, and the new procedure would then be nearly 7,000 times better than the original. Likewise an index or hash on S.S# would help with Step 2 by reducing the 100 tuple I/O's to 50, so that the procedure would now be over 10,000 times better than the original. What this means is, if the original query took three hours to run, the final version will run in just over one second. And of course numerous further improvements are possible." (Date, 1995:504.)

Volgens Feldman en Reouven (2003:15) verkry databasisbestuurstelsels data met behulp van sekere toegangsmetodes. Die toegangsmetodes gebruik indekse om data effektief en vinnig te verkry. Die databasisadministrateur moet besluit watter indekse geskep moet word om die grootste voordeel te bied. Dit wil dus voorkom of die keuse van indekse 'n merkbare impak op die werkverrigting van databasisse het; daarom word 'n paragraaf oor hierdie onderwerp ingesluit. Indekse is nie 'n nuwe konsep nie, maar die gebruik van indekse is een van die voorgestelde tegnieke om databasisse te optimeer (Whitehorn & Marlyn (2003:325), Chen (2006:413), Atzeni *et al.* (1999:332-343) en Feldman & Reouven (2003:33)).

Die gedeelte oor indekse word hoofsaaklik aan die hand van Ramakrishnan en Gehrke (2003:275-282, 339-385, 982-986) bespreek, met insette uit verskeie ander bronne. Elmasri en Navathe (1994:103) beskryf indekse as toegangstrukture wat die spoed verbeter waarmee rekords wat aan sekere soekkriteria voldoen, verkry word. Date (1995:725) sê dat 'n fundamentele voordeel van 'n indeks is dat dit navrae bespoedig, maar dat daar wel 'n nadeel ook is, nl. dat dit bywerkings vertraag. Elke keer wat 'n nuwe rekord bygevoeg of veldwaardes verander word op of in die lêer waarop die indeks gebou is, moet die indeks ook verander word.

'n Meer formele definisie stel dat 'n indeks 'n versameling van data-elemente ("entries") is, met 'n effektiewe wyse om datarekords met 'n soeksleutelwaarde *k* op te spoor (Ramakrishnan & Gehrke, 2003:276). Elkeen van hierdie data-elemente (voorgestel deur *k**) bevat voldoende inligting om een of meer rekords met die soeksleutelwaarde *k* te onttrek.

Volgens Ramakrishnan en Gehrke (2003:276) bestaan drie alternatiewe ten opsigte van data-elemente vir indekse:

- Die data-elemente *k** is in werklikheid die datarekord met soeksleutelwaarde *k* (Alternatief 1). Dus is daar geen nodigheid om die datarekords apart te stoor nie.
- Die data-element is 'n *<k, rid>* paar, waar *rid* die rekord-id van 'n datarekord met soeksleutelwaarde *k* is (Alternatief 2).
- Die data-element is 'n *<k, rid-list>* paar, waar *rid-list* 'n lys van rekord "id's" van die datarekords is met soeksleutelwaarde *k* (Alternatief 3).

Die teorie van indekse word in die volgende paragraaf bespreek.

2.4.1 Die teorie van indekse

Die gedeelte word volgens Ramakrishnan en Gehrke (2003:275-282) se interpretasie verduidelik met 'n paar beskrywings uit ander bronne. Eerstens word gekyk na 'n paar fasette wat die effektiwiteit van soektogte kan beïnvloed, naamlik gebondelde ("clustered") teenoor ontbondelde ("unclustered"), digte ("dense") teenoor yl ("sparse"), primêre of sekondêre indekse, en indekse met saamgestelde soeksleutels.

- Gebondelde teenoor ontbondelde – Die volgorde van die datarekords en die data-elemente stem ooreen in die geval van 'n gebondelde indeks. Alternatief 1 is per definisie gebondeld. Die ander twee alternatiewe kan slegs gebondel ("clustered") word indien die datarekords op die soeksleutelwaarde gesorteer is. Gebondelde indekse is redelik duur om te onderhou as gevolg van die gesorteerde struktuur gedurende bywerkings. Data-elemente mag oor bladsye geskuif word, wat tot gevolg sal hê dat al die verwysings in die databasis dan sáám moet verander. Die datalêer kan slegs volgens een soeksleutel gesorteer word; daar mag dus slegs een gebondelde indeks per tabel wees. Ontbondelde indekse mag meermale per tabel voorkom. Gebondelde indekse word normaalweg geskep met bykomende spasie sodat byvoegbewerkings maklik kan plaasvind. Elmasri en Navathe (1994:107) stel 'n strategie voor waar 'n blok geallokeer word vir elke soeksleutelwaarde om sodoende die byvoegbewerkings te hanteer. SQL Server gebruik die "FILLFACTOR"-argument om die hoeveelheid waarmee elke databladsy aanvanklik gevul word te spesifiseer (verstekwaarde

is dat al die bladsye volgemaak word). As baie wysigings plaasvind, moet die "FILLFACTOR" redelik laag gestel word sodat bladsyverdelings nie so gereeld voorkom nie. SQL Server Performance (2005a) stel voor dat 'n "FILLFACTOR" van ongeveer 50% - 70% gebruik kan word vir 'n groot aantal wysigings. As baie min wysigings plaasvind, is 'n "FILLFACTOR" van ongeveer 100% bruikbaar en enige ander aantal wysigings val tussen die twee kategorieë (80% - 90%). MySQL se InnoDB-stoorenjin-indeksbladsye is ongeveer 93,5% vol (MySQL AB, 2005:888) en MyISAM maak die bladsye heeltemal vol. Gebondelde indekse word normaalweg vir reekstoetse gebruik (soos byvoorbeeld navrae wat inligting wil kry tussen sekere datums), maar kan ook vir gelykheidstoetse gebruik word. Ontbondelde indekse word normaalweg vir gelykheidstoetse gebruik.

- Digte ("dense") teenoor yl ("sparse") – 'n Digte indeks bevat ten minste een data-element vir elke soeksleutelwaarde van 'n rekord in 'n lêer. 'n Yl indeks bevat een data-element vir elke bladsy van rekords in die datalêer. Alternatief 1 en 3 is altyd digte indekse, terwyl alternatief 2 dig of yl kan wees. Dit is nie moontlik om 'n yl indeks te bou wat nie gebondel is nie; daar mag dus net een yl indeks wees. 'n Yl indeks is tipies baie kleiner as 'n digte indeks, maar digte indekse is beter vir optimering. Date (1995:728) stel dat alhoewel yl indekse minder spasie beslaan, ontstaan die probleem dat die indeks dalk nie voldoende is vir gelykheidstoetse nie.
- Primêre en sekondêre indekse – 'n Indeks op kolomme wat die primêre sleutel insluit, word 'n primêre indeks genoem en die alternatief is 'n sekondêre indeks.
- Saamgestelde soeksleutels – Die soeksleutel vir 'n indeks bestaan uit verskeie kolomme.

Twee belangrike tipes indeksstrukture, naamlik boomindekse en "hash"-indekse word ondersoek. Paragrawe 2.4.1.1 en 2.4.1.2 bespreek hierdie twee indeksstrukture.

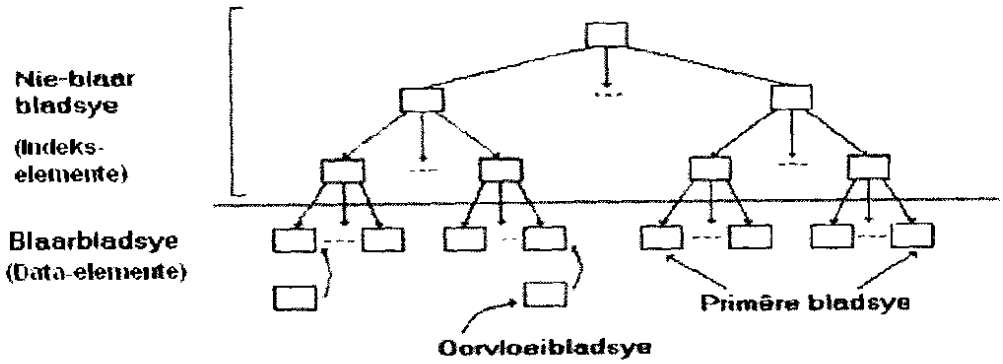
2.4.1.1 Boomindekse

Drie tipes boomindekse word hier bespreek, naamlik "Indexed Sequential Access Method (ISAM)", "B+"-bome en R-bome. Die strukture verskaf effektiewe ondersteuning vir reekssoektogte, byvoeging, verwydering en gelykheidstoetse, maar die implementering vir gelykheidstoets is nie so effektief soos "hash"-indekse (sien paragraaf 2.4.1.2) se implementering nie (McFadden & Hoffer (1988:118, 129), Atzeni *et al.* (1999:325,327) en Ramakrishnan & Gehrke (2003:292)). Die blaarbladsye (sien figuur 2.2) bevat die data-elemente (wat óf na die datarekords verwys óf self die datarekords is) en die ander bladsye bevat die indekselemente met die vorm <soeksleutelwaarde, bladsy-id> en word gebruik in die soektog na 'n data-element. Die drie tipes boomindekse word in die volgende drie paragrawe beskryf.

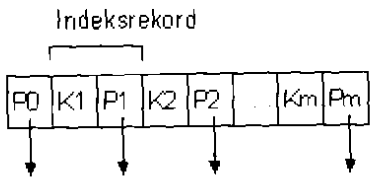
2.4.1.1.1 "Indexed Sequential Access Method (ISAM)"

Veronderstel daar bestaan 'n baie groot datalêer met rekords wat volgens 'n spesifieke soeksleutel gesorteer is en dat 'n soektog (byvoorbeeld 'n binêre soektog[3]) daarop toegepas moet word: 'n Indekslêer of 'n reeks indekslêers wat verskillende "vlakke" verteenwoordig, kan geskep word deur dit te baseer op die datalêer. Die eerste vlak of indekslêer wat in figuur 2.2 die blaarbladsye (met data-elemente) bevat, word geskep as 'n indekslêer met een rekord (data-element) vir elke bladsy in die datalêer. Die rekord se vorm is <eerste sleutel op bladsy, wyser na die bladsy waarop die sleutel is> en die indekslêer is gevolglik gesorteer volgens die sleutel. Hierdie proses kan uitgebrei word deur op die indekslêer (vorige vlak) op soortgelyke wyse 'n volgende vlak (indekslêer) te bou. In hierdie verdere vlakke verteenwoordig die (indeks)bladsye die nie-blaarbladsye (met indekselemente). Die proses van volgende vlakke word voortgesit totdat 'n indekslêer verkry word wat op een bladsy pas. So 'n struktuur staan bekend as die ISAM-indeksstruktuur en is staties omdat die indeks self nie tydens bywerk van datarekords aangepas word nie. Figuur 2.2 toon die ISAM-indeksstruktuur en figuur 2.3 toon die formaat van die indeksbladsy. Elke indeksbladsy bevat een meer wyser as die aantal sleutels aangesien die wysers na die begin en einde van die bladsye wys.

Figuur 2.2 – ISAM-indeksstruktuur



Figuur 2.3 – Formaat van 'n ISAM-indeksbladsy



[3] Paul E. Black (2005) beskryf die binêre soekmetode as 'n algoritme wat 'n gesorteerde skikking ("array") van rekords herhaaldelik in die helfte verdeel. Die eerste gebied wat ondersoek word, is ekwivalent aan die hele skikking. As die gesoekte rekordsleutel kleiner is as die sleutel in die middel van die gebied, dan word die gebied verdeel en die eerste helfte word nou die nuwe gebied. As die sleutel in die laaste helfte voorkom, word die laaste helfte die nuwe gebied om te ondersoek. Die proses word herhaal totdat die korrekte sleutel verkry word of totdat die gebied nie meer verdeel kan word nie.

Om 'n bepaalde (data)rekord met 'n spesifieke sleutelwaarde te vind, moet 'n soekmetode op die indekslêer toegepas word. Die indeks- en data-elemente met soeksleutels en wysers word gebruik om deur die verskillende vlakke van die struktuur te beweeg om die bladsye te identifiseer wat die eerste sleutelwaarde bevat wat aan die soektog voldoen. Die spesifieke bladsy word dan sekwensieel deursoek om die korrekte rekord te kry.

Oorvloeibladsye ontstaan wanneer elemente bygevoeg word en die data nie meer op een blaarbladsy pas nie. Bykomende oorvloeibladsye word geskep om die data-elemente te huisves aangesien die indeksstruktuur 'n statiese struktuur is. Indien een van hierdie oorvloeibladsye as gevolg van verwydering leeg raak, word die bladsy verwyder. Primêre blaarbladsye word egter nie verwyder wanneer dit leeg raak nie.

Die probleem ontstaan dat oorvloeibladsye dalk te veel kan raak en die soektog kan belemmer. Ramakrishnan en Gehrke (2003:344) stel voor dat die boom geskep word met 20% van elke bladsy wat oop is, alhoewel die bladsye steeds vol kan raak en slegs 'n volledige herorganiseringsproses die bladsye weer sal herstel tot op die 20%-vlak. 'n Groot voordeel van hierdie struktuur is die feit dat indeksvlakbladsye nie tydens normale gebruik (byvoeg en verwyderbewerkings) verander word nie en die bladsye daarom nie vir eksklusiewe gebruik deur gebruikers gesluit word nie. Die sluit van die bladsye kan tot potensiële bottelnekke (versperring in die vloei van data) lei.

2.4.1.1.2 "B+"-bome (of B-bome)

Die ISAM-struktuur het die probleem dat lang oorvloeikettings kan voorkom soos wat die lêer groei, wat verswakte werkverrigting tot gevolg het. Hierdie probleem het gelei tot 'n meer dinamiese struktuur wat maklik aanpas vir byvoeg en verwyderbewerkings, naamlik die "B+"-boomstruktuur. Die gebalanseerde boomstruktuur se interne nodusse rig die soektog, en die blaarnodusse bevat die data-elemente, wat enige van die alternatiewe in paragraaf 2.4 kan gebruik. Die struktuur is 'n dinamiese struktuur waar al die blaarnodusse geskakel word deur 'n dubbelgeskakelde lys ("doubly linked list"), wat die beweging tussen die bladsye tydens die soektog vergemaklik. B-bome word redelik algemeen in die praktyk gebruik. Sowel SQL Server as MySQL maak gebruik van hierdie indeksstruktuur. 'n Paar karakteristieke van die boomstruktuur word uitgelig:

- Bewerkings soos byvoeg en verwyder hou die indeks in balans.

- Ramakrishnan en Gehrke (2003:345) stel 'n voorbeeld voor waar elke nodus, behalwe die wortelnodus, altyd ten minste 50% vol sal wees mits die verwyderalgoritme later in hierdie

paragraaf (sien figuur 2.6) gevolg word. Die moontlikheid bestaan dat elemente verwyder kan word en die boom nie verstel word nie, wat dus tot leë nodusse kan lei. Die 50%-okkupasie word in die bespreking van die indekse as voorbeeld gebruik.

- Die lengte van die deurloop of pad wat die soektog volg om te beweeg vanaf die wortelnodus na die blaarnodus (wat die betrokke rekord bevat), word die hoogte van die boom genoem.

- Ramakrishnan en Gehrke (2003:346) beweer dat "B+"-bome normaalweg bo 'n gewone gesorteerde lêer verkies word indien rekords baie verander en gesorteerde toegang belangrik is.

- Die "B+"-boomstruktuur gebruik nie oorvloeibladsye nie.

- Die sluit-koste, om rekords se integriteit te behou, is duurder as die ISAM-struktuur en die boomstruktuur se bladsye kan mettertyd deurmekaar raak, wat soektogte kan belemmer.

- Die grootte van die indekselemente speel 'n belangrike rol in die berekening van die aantal elemente wat op 'n bladsy pas en dus ook die verspreiding ("fan-out") van die boom. Ramakrishnan en Gehrke (2003:358) beweer dat die hoogte eweredig is aan die formule:

  $\log_{\text{verspreiding}}$ (# aantal data-elemente of N)

  Die aantal skyf-lees- of -skryf-aksies (I/Os) om 'n datarekord in die boom te vind, is gelyk aan die hoogte. Die waarde van die verspreiding is normaalweg relatief groot aangesien die grootte van die indekselemente die aantal indekselemente op 'n bladsy bepaal, en in die praktyk kan 'n redelike hoeveelheid elemente op 'n bladsy inpas. Dit is dus belangrik om die verspreiding so groot as moontlik te kry om die hoogte te beperk. 'n Tegniek genaamd "sleutelkompressie" ("key compression") word gebruik om hiermee te help. Die tegniek werk volgens die beginsel dat die hele sleutel nie noodwendig nodig is om 'n rekord te herken nie. Indien daar byvoorbeeld tussen "David Smith" en "Devarakonda" onderskei moet word, is dit slegs nodig om "Da" en "De" te vergelyk. Die verkleinde vorm word dan gestoor. Die verkleinde vorms van die oorspronklike sleutels word vooraf bepaal deur na al die indeks- en sleutelwaardes te kyk.

- Die orde (d) van 'n boom dui die kapasiteit van die boom se nodusse aan en word baie moeilik in die praktyk bepaal. Ramakrishnan en Gehrke (2003:363) plaas eerder die fokus op spasie as 'n bepalende faktor. Die nodusse moet byvoorbeeld ten minste halfvol gehou

word. Data-elemente en indekselemente bevat normaalweg verskillende data. Die data-elemente sal normaalweg groter wees aangesien die indekselemente net 'n sleutelwaarde en wyser stoor. Spasie is dus 'n beter aanduiding van die okkupasie van 'n nodus as die aantal elemente in 'n nodus. Die grootte van die soeksleutelwaarde mag verskil indien die sleutel byvoorbeeld karakters bevat. Die grootte van die blaarnodus mag verskil vir rekords met dieselfde soeksleutelwaarde.

Gestel elke nie-blaarnodus bevat *m*-elemente, waar $d \leq m \leq 2d$. Die wortelnodus is 'n uitsondering in dié opsig dat $1 \leq m \leq 2d$. Die formaat van die nodusse is soortgelyk aan ISAM se formaat (sien figuur 2.3). Nie-blaarnodusse met *m*-indekselemente bevat *m+1*-wysers na die volgende vlak in die boom. Wyser P_i wys na die gedeelte van die boom waar alle sleutelwaardes K bestaan sodat $K_i \leq K < K_{i+1}$. Die spesiale gevalle is P_0 wat na 'n gedeelte van die boom wys waar alle sleutelwaardes kleiner is as K_1 en P_m wat na 'n gedeelte van die boom wys waar alle waardes groter of gelyk is aan K_m. Die blaarnodusse se data-elemente word aangedui as k*. Ramakrishnan en Gehrke (2003:346) beweer dat alternatiewe 2 en 3 normaalweg gebruik word, wat dan lei tot <K, I(K)>[4]-tipe data-elemente vir die blaarnodusse. Die aanname word gemaak dat die boom geen duplikate besit nie. SQL Server hanteer duplikate deur 'n unieke id ("uniqueifier") vir elke ry te genereer indien 'n unieke id nie alreeds gespesifiseer is nie (Delaney, 2001:412). Die "B+"-boomstruktuur word deur middel van pseudokode vir die soek-, byvoeg- en verwyderbewerkings in die volgende paragrawe beskryf.

- Soekbewerking – Die pseudokode vir die soektog word in figuur 2.4 getoon. Die algoritme gebruik rekursie om deur die boom te beweeg tot die nodus met die betrokke soeksleutelwaarde verkry word. Elkeen van die indekselemente word deursoek om die wyser P_i na die volgende vlak van die boom te verkry. Die blaarnodus word op die einde deursoek om die sleutelwaarde te verkry. Ramakrishnan en Gehrke (2003:347) stel twee metodes voor om die blaarnodus te deursoek, naamlik 'n lineêre en binêre soektog. 'n Voorbeeld kan in Ramakrishnan en Gehrke (2003:347-348) se boek verkry word.

- Byvoegbewerking – Figuur 2.5 toon die aangepaste algoritme vir die bewerking. Die simbool "&" dui 'n verwysing aan op die betrokke element. Die bewerking verkry met behulp van die *Soek*-funksie die blaarnodus waar die nuwe element moet kom. As die nodus spasie beskikbaar het, word die element bygevoeg. Indien die nodus vol is, moet die nodus verdeel word. Die verdeling vind plaas deur 'n nuwe kindnodus te skep, die waardes van die nodus wat verdeel word (ou nodus) tussen die ou nodus en die nuwe nodus te verdeel, en 'n wyser na die nuwe nodus in die ouernodus te plaas. 'n Kindnodus is enige nodus waarheen

[4] Stem ooreen met die alternatiewe van paragraaf 2.4 waar I(K) 'n lys of een rekord-id voorstel.

Figuur 2.4 – Aangepaste pseudokode-algoritme vir die soekbewerking van 'n "B+"-boomindeks gebaseer op Ramakrishnan en Gehrke (2003:347) se algoritme.

```
Funksie Soek (soeksleutelwaarde K) verskaf noduswyser
   // Vind die blaarnodus vir die betrokke soeksleutelwaarde
   Verskaf (Boom_soek (wortel, K));                    // soek vanaf wortel deur die boom
Einde van funksie

Funksie Boom_soek (noduswyser, soeksleutelwaarde K) verskaf noduswyser
   As *noduswyser 'n blaar is, verskaf noduswyser;   // * verkry nodus waarheen noduswyser
wys.
   Anders,
      Stel waardes K1, ..., Km gelyk aan *noduswyser se indekselemente.        // (1)
      As K < K1, verskaf Boom_soek (P0, K);
      Anders,
         As K ≥ Km, verskaf Boom_soek (Pm, K);   // m = aantal indekselemente in 'n bladsy
         Anders,
            Vind i sodat Ki ≤ K < Ki+1;              // 1 ≤ i ≤ (m – 1)
               As i gevind is, verskaf Boom_soek (Pi, K);
               Anders, verskaf geen noduswyser;    // dui aan dat K nie gevind is nie
Einde van funksie
```

'n ouernodus verwys en nodusse wat weerskante van 'n betrokke nodus is en dieselfde ouernodus besit, staan bekend as 'n verwante nodus. Die prosedure keer dan weer terug na die oorspronklike funksie. Ramakrishnan en Gehrke (2003:348-352) bied voorbeelde vir hierdie bewerking.

'n Variasie op hierdie metode versprei die elemente tussen die naaste verwante bladsy in stede van verdeling van die nodus. As die verwante nodus vol is, vind verdeling soos normaalweg plaas. Die verspreidingsproses lei tot meer lees- of skryf-aksies en Ramakrishnan en Gehrke (2003:351) stel dat dit normaalweg lonend is om nie die verspreiding toe te pas by die nie-blaarnodusse nie. 'n Beperkte vorm van die verspreiding by die blaarnodusse kan gebruik word.

- Verwyderbewerking – Die aangepaste algoritme word in figuur 2.6 vertoon. Die data-elemente word gesoek en verwyder. Die probleem ontstaan wanneer elemente in 'n nodus verminder en onder die minimum-okkupasie[5] beland. Die elemente van 'n verwante nodus moet in so 'n geval versprei na die nodus wat te min elemente bevat, of die nodus moet saamsmelt met 'n verwante nodus. Die ouernodus moet in beide gevalle verander word om die korrekte verwysings na die kindnodusse te verkry. In die geval waar die laaste indekselement in die wortelnodus en die wortelnodus self verwyder word, word die hoogte van die boom verminder met een.

[5] Minimum aantal elemente wat binne 'n nodus mag wees of minimum spasie wat vol moet wees.

Figuur 2.5 – Aangepaste pseudokode-algoritme vir die byvoegbewerking van 'n "B+"-boomindeks gebaseer op Ramakrishnan en Gehrke (2003:349) se algoritme.

```
Prosedure Byvoeg (noduswyser, element, nuwe_kind_nodus)
   // Aanvanklik is *noduswyser die wortelnodus, die orde is d en die nuwe_kind_nodus is
   // "null" totdat die kindnodus verdeel.

As *noduswyser 'n nie-blaarnodus is, sê N,
   Stel waardes K1, ..., Km gelyk aan *noduswyser se indeks- of data-elemente-waardes;
   Vind i sodat Ki ≤ element se sleutelwaarde < Ki+1;          // Kies gedeelte van boom
   Byvoeg (Pi, element, nuwe_kind_nodus);                      // Gebruik rekursie
   As nuwe_kind_nodus "null" is, keer terug na roepende funksie; // Kind het nie verdeel nie
   Anders;
      As N spasie bevat,
         voeg *nuwe_kind_nodus in, stel nuwe_kind_nodus gelyk aan "null" en keer terug;
      Anders,
         Verdeel N:     // 2d + 1 sleutelwaardes (insluitende die *nuwe_kind_nodus) en
                        // 2d + 2 noduswysers
            Voeg *nuwe_kind_nodus in die gesorteerde 2d sleutelwaardes in
            Eerste d sleutelwaardes en d+1 noduswysers bly in N,
            Laaste d-sleutels en d+1-wysers beweeg na nuwe nodus, N2,
            Die middelste sleutelwaarde (Kd+1) word behou;
         nuwe_kind_nodus = &(<middelste sleutelwaarde, wyser na N2>);
         As N die wortelnodus is,
            Skep nuwe nodus met <wyser na N, *nuwe_kind_nodus>;
            Laat die boom se wortelnodus wyser na die nuwe nodus wys;
         Keer terug na roepende funksie;
Anders (stel blaarnodus is L),
   As L spasie bevat,
      Voeg element by, stel nuwe_kind_nodus gelyk aan "null" en keer terug;
   Anders,
      Verdeel L:
         Eerste d elemente bly en die res beweeg na die nuwe nodus L2;
         As element ≤ laaste element in L,
            Voeg element in L in op gesorteerde plek;
         Ander,
            Voeg element in L2 in op gesorteerde plek;
         nuwe_kind_nodus = &(<kleinste sleutelwaarde op L2, wyser na L2>);
         Stel verwante nodusse aan weerskante van oorspronklike L se wysers na L en L2;
         Keer terug na roepende funksie;
Einde van prosedure
```

Samesmelting (of vereniging) vind plaas deur die verdeelsleutel vanaf die *ouerwyser-nodus af te trek en in verwante linkerkantste nodus te plaas. Die elemente van die regterkantste nodus word na die linkerkantste nodus geskuif. Die leë nodus word verwyder en die wysers word gekorrigeer.

Verspreiding vind plaas deur elemente vanaf die nodus met die meeste elemente oor te dra na die nodus wat onder die minimum-okkupasie-getal is. In die geval van die nie-blaarnodusse sal die element wat oorgedra word die verdeelsleutel in die ouernodus vervang en die verdeelsleutel skuif oor na die kindnodus met die minste elemente. Die blaarnodusse versprei die elemente en vervang dan die verdeelsleutel deur die laagste waarde van die regterkantste nodus.

Figuur 2.6 – Aangepaste pseudokode-algoritme vir die verwyderbewerking van 'n "B+"-boomindeks gebaseer op Ramakrishnan en Gehrke (2003:353) se algoritme.

```
Prosedure Verwyder (ouerwyser, noduswyser, element, ou_kind_element)
    // Verwyder element van 'n gedeelte van die boom met wortel *noduswyser en orde d.
    // ou_kind_element is "null" tensy 'n kindnodus verwyder word.

As *noduswyser 'n nie-blaarnodus is, sê N,
    Stel waardes K1, ..., Km gelyk aan *noduswyser se indeks- of data-elemente-waardes;
    Vind i sodat Ki ≤ element se sleutelwaarde < Ki+1;          // Kies gedeelte van boom
    Verwyder (noduswyser, Pi, element, ou_kind_element);        // Rekursiewe verwyder
    As ou_kind_element "null" is, keer terug na roepende funksie;
    Anders,
        Verwyder *ou_kind_element van N,                        // (1)
        As N elemente oor het wat meer as die minimum-okkupasie-getal is,
            Stel ou_kind_element gelyk aan "null" en keer terug na roepende funksie;
        Anders,
            Verkry verwante nodus S van N:          // Maak gebruik van ouerwyser
            As S nie bestaan nie,
                As N leeg is,
                    Verwyder N, stel ou_kind_element gelyk aan "null" en keer terug na roepende funksie;
                Anders,
                    Stel ou_kind_element gelyk aan "null" en keer terug na roepende funksie;
            As S elemente oor het wat meer as die minimum-okkupasie-getal is,
                Versprei die elemente eweredig tussen N en S m.b.v. die ouerwyser;
                Elke keer wat 'n element oorgaan vanaf S na N, word die verdeelsleutel na N
                geskuif en die element wat oorgegaan het raak die nuwe verdeelsleutel
                in die *ouerwyser-nodus;
                Stel ou_kind_element gelyk aan "null" en keer terug na roepende funksie;
            Anders, verenig N en S                  // Noem regterkantste nodus M
                ou_kind_element = &(huidige element in *ouerwyser-nodus vir M);
                Trek verdeelsleutel vanaf *ouerwyser na die linkerkantste nodus van N of S;
                As *ouerwyser leeg is,
                    Verwyder *ouerwyser en verstel verwysings;
                Skuif alle elemente vanaf M na linkerkantste nodus;
                Verwyder leë nodus M en keer terug na roepende funksie;
Anders, (stel blaarnodus is L)
    As L elemente oor het wat meer as die minimum-okkupasie-getal is,
        Verwyder element, stel ou_kind_element gelyk aan "null" en keer terug na roepende funksie;
    Anders,
        Verkry verwante nodus S van L;
        As S elemente oor het meer as die minimum-okkupasie-getal is,
            Versprei die elemente eweredig tussen L en S;
            Vind element in *ouerwyser vir regterkantste nodus;        // Noem die nodus M
            Vervang verdeelsleutel in die *ouerwyser-nodus deur die nuwe laagste sleutelwaarde in M;
            Stel ou_kind_element gelyk aan "null" en keer terug na roepende funksie;
        Anders, verenig L en S                                         // Noem regterkantste nodus M
            ou_kind_element = &(huidige element in *ouerwyser-nodus vir M);
            Skuif alle elemente van M na linkerkantste nodus;
            Verwyder leë nodus M, verander verwante nodusse se wyser, keer terug na roepende funksie;
Einde van prosedure
```

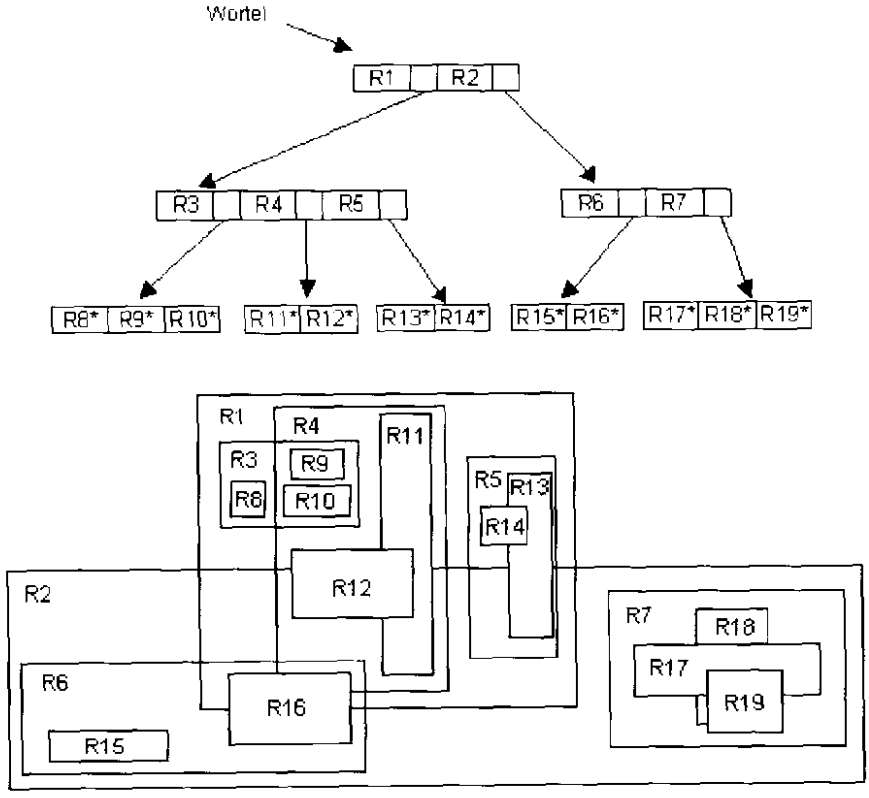
2.4.1.1.3 R-bome

Die teorie van R-bome word aan die hand van Ramakrishnan en Gehrke (2003:982-986) verduidelik. R-boomindekse is gebaseer op "B+"-boomindekse, hanteer ruimtelike ("spatial") data en is hoogte-gebalanseerde boomstrukture. Ramakrishnan en Gehrke (2003:969)

definieer ruimtelike data as 'n breedvoerige benaming van alle multidimensionele punte, lyne, vierkante, polinome, kubusse en ander geometriese objekte, en elke ruimtelike data-objek verteenwoordig 'n spesifieke area wat 'n posisie en grens bevat. R-bome word nie breedvoerig bespreek nie aangesien net MySQL die boomstruktuur ondersteun.

Die soeksleutel vir R-bome is 'n versameling intervalle, met een interval per dimensie. Die soeksleutelwaarde stel 'n boks voor wat begrens word deur intervalle, en elke kant van die boks is parallel aan 'n as. Die soeksleutelwaarde word 'n begrensde boks ("bounding box") genoem. 'n Data-element bestaan uit *<n-dimensionele boks, rid>*-pare, waar *rid* die objek identifiseer en die boks die kleinste boks voorstel wat die objek kan bevat. 'n Spesiale geval kom voor waar die objek 'n punt is en nie 'n area nie. Die boks om die punt stel ook 'n punt voor. Data-elemente word in die blaarnodusse gestoor en nie-blaarnodusse bevat indekselemente van die vorm *<n-dimensionele boks, wyser na 'n kindnodus>*. Die boks by die nie-blaarnodus N is die kleinste boks wat al die ander bokse van die kindernodusse ook bevat, met ander woorde die boks bevat die area wat al die data-objekte stoor in die subboom met die wortel by N. Figuur 2.7 toon 'n voorbeeld met twee verskillende voorstellings van 'n R-boom aan.

Figuur 2.7 – Twee voorstellings van R-bome gebaseer op
Ramakrishnan en Gehrke (2003:982)



Die eerste voorstelling wys die boomstruktuur en die tweede voorstelling wys die data-objekte en die begrensde bokse. Die 19 areas word voorgestel deur areas R1 – R19. Die areas R8* – R19* dui data-elemente aan, waar R8* byvoorbeeld bestaan uit 'n begrensde boks vir die area R8. Area R1 stel die begrensde boks voor vir die linkerkantste subboom, wat die data-objekte R8 – R14 insluit. Die begrensde boks vir twee kindernodusse van 'n gegewe nodus kan oorvleuel, byvoorbeeld die kindernodusse R1 en R2 van die wortelnodus wat oorvleuel. Elke data-objek word in een blaarnodus gestoor, al val die objek se begrensende boks in verskeie areas wat ooreenstem met twee of meer hoërvlaknodusse. Die data-objek R9 is deur beide R3 en R4 bevat en sou in beide die eerste en die tweede blaarnodusse geplaas kon word.

Die verskillende bewerkings, naamlik soek, byvoeg en verwyder, word nie hier behandel nie aangesien die tipe indeks nie in die eksperimentele gedeelte gebruik word nie. Ramakrishnan en Gehrke (2003:984-986) bied 'n bespreking oor hierdie bewerkings.

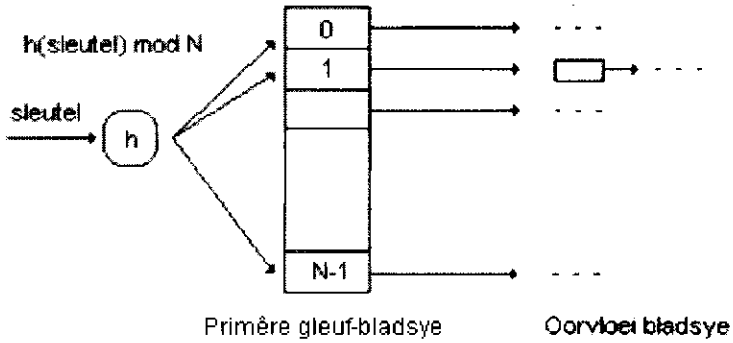
2.4.1.2 "Hash"-indekse

'n "Hash"-funksie word gebruik om waardes in 'n soekveld om te skakel na 'n gleufnommer ("bucket number") om die bladsyverwysing te verkry waar die data-element voorkom. Die tipe indeks ondersteun nie reekssoektogte nie en word vir gelykheidsoektogte gebruik. Boomindekse ondersteun wel reekssoektogte en is ook aanvaarbaar ten opsigte van gelykheid-soektogte. Die "hash"-indeks word gebruik waar gelykheid 'n groot rol speel, soos byvoorbeeld by verbinding van tabelle in 'n navraag. Twee tipes "hash"-indekse word hier beskryf, naamlik statiese "hashing" en dinamiese "hashing".

2.4.1.2.1 Statiese "hashing"

Figuur 2.8 toon die basiese elemente van die metode. 'n Versameling gleuwe bevat die bladsye waar die data-elemente voorkom. Elke gleuf bevat 'n aanvanklik primêre bladsy en mag later oorvloeibladsye bykry. 'n Data-element word verkry deur 'n "hash"-funksie *h* te gebruik om die gleuf te identifiseer en dan daarna die gleuf te deursoek.

Figuur 2.8 – Statiese "Hashing"



Die soektog se spoed kan verbeter word deur die data-elemente te sorteer volgens die soeksleutelwaarde in die gleuf. Die byvoeg van 'n nuwe data-element geskied deur die "hash"-funksie toe te pas en die element dan in die betrokke gleuf te plaas. Indien die gleuf vol is, word 'n oorvloeibladsy geskep en die data-element word hierin gestoor. Die oorvloeibladsy word dan bygevoeg tot die oorvloeiketting van die gleuf. 'n Element kan net so maklik verwyder word. Die gleuf word volgens die "hash"-funksie gesoek en daarna deursoek vir die data-element, wat dan verwyder word. Indien die laaste element van 'n oorvloeibladsy verwyder word, word die oorvloeibladsy verwyder en in 'n skoonbladsyelys geplaas. Die probleem met hierdie metode is dat die hoeveelheid gleuwe staties is en dit baie oorvloeibladsye tot gevolg kan hê. 'n Tegniek ("re-hashing") bestaan om die lêer se oorvloeibladsye te verwyder en die bladsye weer volgens 'n voorafbepaalde persentasie te vul. Die tegniek neem egter tyd en die indeks is onbruikbaar tydens "re-hashing". Ramakrishnan en Gehrke (2003:372) stel een moontlike formule vir die "hash"-funksie voor: $h(waarde) = (a * waarde + b)$ met a en b arbitrêr gekies met die oog daarop om die "hash"-funksie te optimeer. Alhoewel SQL Server nie "hash"-indekse spesifiek gebruik nie, word baie van die "hash"-beginsels gebruik tydens sortering en verbindings ("joins") van tabelle.

2.4.1.2.2 Dinamiese "hashing"

Dié tipe indekse is gebaseer op die statiese "hashing"-tipe. Die dinamiese tipe gebruik een of ander vorm van 'n wyser wat na elke gleuf wys. Die aantal gebruikte gleuwe vermeerder soos wat elemente bygevoeg word en die gleuwe vol raak. Sekere voorwaardes veroorsaak dat die aantal wysers, en dus ook die aantal beskikbare gleuwe, vermeerder. Oorvloeibladsye word wel in dinamiese "hashing" gebruik, maar die aantal word beheer deur die vermeerdering van die gleuwe en die daaropvolgende verspreiding van elemente.

Ramakrishnan en Gehrke (2003:373-379) toon 'n voorbeeld van uitgebreide ("extendible") "hashing" waar 'n gids van wysers gebruik word om na die onderskeie gleuwe te wys. "Hash"-funksies word gebruik om te bepaal watter wyser in die gids gebruik word. 'n Verdubbeling van

die aantal beskikbare gleuwe vind plaas as die gleuf alreeds verdeel het sedert die vorige verdubbelingsproses en die gleuf nou weer moet verdeel. Nog 'n voorbeeld van dinamiese "hashing" is lineêre "hashing". Lineêre "hashing" gebruik nie die gids van wysers nie, maar eerder 'n aantal verwante "hash"-funksies met elke funksie wat twee keer soveel gleuwe kan hanteer as die vorige funksie. Verdeling van gleuwe vind plaas in rondtes en verdubbeling vind plaas as al die gleuwe een keer verdeel het sedert die vorige verdubbeling.

Die soek-, byvoeg- en verwyderbewerkings word in die volgende paragrawe bespreek.

- Soekbewerking – 'n Element word verkry deur die "hash"-funksie op die element toe te pas en dan die spesifieke gleuf te deursoek.

  Ramakrishnan en Gehrke (2003:373) bepreek aan die hand van 'n voorbeeld 'n "hash"-funksie wat die laaste twee bisse van die binêre waarde van die soekwaarde neem om die ooreenkomstige wyser in die gids te kry. In die praktyk sal 'n groter aantal bisse gebruik word om meer moontlikhede en gleuwe te verskaf. Die gleuf waarna die wyser in die gids of "hash"-funksie wys, bevat die data-element.

- Byvoegbewerking – Die gleuf[6] word met die "hash"-funksie verkry. As die gleuf 'n oop spasie besit, word die data-element in die gleuf bygevoeg. Indien daar nie 'n oop spasie bestaan vir die gleuf nie, word oorvloeibladsye gebruik of die aantal gleuwe word vermeerder om die bykomende elemente te kan huisves. Ramakrishnan en Gehrke (2003:373-379) bespreek 'n voorbeeld van die byvoegbewerking vir uitgebreide ("extendible") "hashing", en Ramakrishnan en Gehrke (2003:379-384) bespreek 'n voorbeeld van die byvoegbewerking vir lineêre "hashing".

- Verwyderbewerking – Die bewerking is net die teenoorgestelde van die byvoegbewerking. As data-elemente in die gleuf oor is na verwydering, is die bewerking klaar. As oorvloeibladsye leeg raak, word die bladsye verwyder. As die gleuf leeg is, word die gleuf verwyder en die onderskeie wysers wys na slegs een gleuf[7]. Dit is moontlik dat die aantal beskikbare gleuwe gehalveer kan word indien die aantal gebruikte gleuwe weer ooreenstem met die aantal gleuwe voor enige verdeling plaasgevind het, alhoewel Ramakrishnan en Gehrke (2003:378) stel dat hierdie stap in die praktyk uitgelaat mag word.

[6] 'n Wyser na die gleuf kan ook met 'n "hash"-funksie (bepaal deur die tipe dinamies "hashing"-algoritme) verkry word.

[7] Een van die gleuwe wat tevore betrokke was tydens die verdeling van 'n gleuf.

2.5 Slotparagraaf

In hierdie hoofstuk is die basiese beginsels van databasisse en databasisbestuurstelsels bespreek. Verskeie tipes indekse is bespreek aangesien indekse een van die belangrikste optimeringstegnieke vorm en in hoofstuk 6 in die eksperimentele gedeelte gebruik word. Boom- en "hash"-indekse is ondersoek en bespreek. Vir beide boom- en "hash"-indekse is gekyk na statiese sowel as dinamiese strukture. Die volgende hoofstuk bespreek die databasisbestuurstelsels SQL Server en MySQL se interne werking. Elke databasisbestuurstelsel word ondersoek ten opsigte van die betrokke stelsel se ontstaan, argitektuur, stoorenjins, lêerstrukture, indekse en navraagoptimeerder, asook verskeie toepassings (programme) van elk.

HOOFSTUK 3

DIE DATABASISBESTUURSTELSELS: SQL SERVER EN MYSQL

3.1 Inleiding

Die databasisbestuurstelsels wat in die ondersoek gebruik word, is MySQL 4.1 (MySQL Server 5.0 het eers aan die einde van hierdie navorsing vir produksie beskikbaar geraak) en Microsoft SQL Server 2000. Die beskrywing konsentreer op elemente wat belangrik is vir die proses van optimering, sowel as algemene inligting oor die onderskeie databasisbestuurstelsels. Die databasis- en lêerstruktuur vir elke databasisbestuurstelsel word ondersoek en die navraagoptimeerder van SQL Server en MySQL word beskryf. Die hantering van indekse deur SQL Server en MySQL word ook in hierdie hoofstuk bespreek.

Paragraaf 3.2 beskryf Microsoft SQL Server 2000. Die onderwerpe wat aangeroer word, is die ontstaan van SQL Server, 'n oorsig oor die elemente binne SQL Server, die interne argitektuur van SQL Server, geheuehantering, die databasis- en lêerstruktuur van SQL Server, sommige datatipes in SQL Server, die struktuur van 'n data-ry, SQL Server se hantering van indekse, die navraagverwerker en verskeie toepassings in die SQL Server-pakket. Paragraaf 3.3 beskryf MySQL 4.1. Die onderwerpe waarna gekyk word, is die ontstaan van MySQL, 'n oorsig oor die elemente in MySQL, die wyses waarop MySQL bywerkings hanteer, die interne argitektuur van MySQL (sluit stoorenjins in), indekse, die wyses waarop MySQL optimering toepas en die toepassings wat MySQL moontlik maak. Paragraaf 3.4 is die slotparagraaf.

3.2 Microsoft SQL Server 2000

Die beskrywing van hierdie relasionele databasisbestuurstelsel volg aan die hand van die boek geskryf deur Kalen Delaney (2001) met ondersteuning van 'n paar ander bronne wat genoem sal word soos wat die bron in gebruik kom.

3.2.1 Die ontstaan van SQL Server

Die gedeelte is gebaseer op die werk van Delaney (2001:4-30). Microsoft het in 1986 begin met die ontwikkeling van databasisbestuurstelsels. Produkte soortgelyk aan databasisbestuurstelsels (soos byvoorbeeld Lotus 1-2-3, MicroRim se Rbase, Ansa Software se Paradox en Ashton-Tate se dBase-produkte) het reeds bestaan.

Microsoft het Sybase, Inc. begin ondersoek. Sybase wou sy eerste kommersiële weergawe van DataServer in Mei 1987 vir Sun-werkstasies met Unix vrygestel het. Die twee besighede het die voordeel van 'n onderlinge ooreenkoms ingesien om 'n databasisbestuurstelsel te ontwikkel: Microsoft verkry eksklusiewe regte tot die DataServer-produk wat op 'n OS/2-bedryfstelsel sou loop, en ook vir alle Microsoft-bedryfstelsels terwyl Sybase 'n winsaandeel sowel as krediet vir hulle produk verkry. Met hierdie ooreenkoms het Sybase toegetree tot die mark van persoonlike rekenaars wat met OS/2 werk. Die ooreenkoms is op 27 Maart 1987 tussen Microsoft se president, Jon Shirley, en Sybase se president en mede-eienaar, Mark Hoffman, gesluit.

Ashton-Tate se produkte het egter alreeds die grootste deel van die mark besit. Die beste manier waarop Microsoft hulle produk vinnig kon bemark en terselfdertyd die produk se reputasie kon verhoog, was om vir Ashton-Tate so ver te kry om die produk te ondersteun. In 1988 is die eerste betaweergawe van Ashton-Tate / Microsoft SQL Server vrygestel.

Ashton-Tate / Microsoft SQL Server weergawe 1.0 is in Mei 1989 met goeie resensies vrygestel. Die verkope het egter nie aan verwagtinge voldoen nie. Een van die hoofredes is dat die verkope van OS/2-bedryfstelsels nie so hoog was nie. Daar was ook nie baie programmatuur beskikbaar om SQL Server-toepassings mee te skep nie. Ashton-Tate se beloofde dBase IV Server Edition, wat data in die SQL Server sou stoor, was nie klaar nie.

Teen 1990 het die ooreenkoms met Ashton-Tate om SQL Server aanloklik te maak vir die dBase-gebruikers nog nie werklik vrugte afgewerp nie. Die produk dBase IV vir gewone rekenaars is in 1989 uitgerol maar het verskeie gebreke gehad. Die beloofde dBase IV vir bedieners was nog nie beskikbaar nie.

Ashton-Tate het begin sukkel om kop bo water te hou en het weer eksklusiewe aandag aan die dBase-produkte gegee. Microsoft wou OS/2 LAN (Lokalearea-netwerk) Manager vrystel wat op SQL Server gesteun het om te help met die ontwikkeling van kliënt/bedienertoepassings wat op Microsoft LAN Manager en Microsoft OS/2 sou loop. Die ooreenkoms tussen Microsoft en Ashton-Tate is beëindig en die produk is Microsoft SQL Server genoem.

Microsoft SQL Server weergawe 1.1 is in 1990 vrygestel as 'n opgradering van die Ashton-Tate / Microsoft SQL Server weergawe 1.0-produk. Baie van die foute van weergawe 1.0 is reggestel, maar die grooste voordeel van die nuwe weergawe was die ondersteuning deur die Microsoft Windows 3.0-bedryfstelsel wat in Mei 1990 vrygestel is.

Nuwe kliëntsagteware, programmeringsbiblioteke ("libraries") en administratiewe toepassings is by die nuwe weergawe gevoeg. Die kern SQL Server-enjin was steeds in Sybase se besit (saam met die kode). Microsoft was besig om 'n onderhoudgroep te ontwikkel vir SQL Server. Die groep het egter nie na wense gevorder nie as gevolg van 'n gebrek aan interne kennis van die SQL-enjin en -bronkode.

Microsoft het In 1991 leesregte tot die bronkode van Sybase gekry. Veranderings moet steeds na Sybase gestuur word. In die middel van 1991 is skryfregte aan Microsoft toegeken, mits die veranderings deur Sybase goedgekeur is. In dieselfde jaar is Microsoft SQL Server 1.11 vrygestel, wat as 'n onderhoudweergawe gesien is. OS/2 het steeds nie verkoop soos verwag nie, maar Windows 3.0 het baie goed verkoop. In Mei 1991 het Microsoft besluit om hulle ooreenkoms met IBM vir die produsering van OS/2 te beëindig en te konsentreer op 'n nuwe bedryfstelsel met die kodenaam NT ("new technology"). Die stelsel se naam sou aanvanklik OS/2 weergawe 3.0 wees, maar nadat die ooreenkoms tussen Microsoft en IBM verbreek is, is dit verander na Microsoft Windows NT. Die produk sou ongeveer twee jaar neem om te ontwikkel en Microsoft SQL Server moes intussen steeds op OS/2 bly bestaan.

Microsoft SQL Server 4.2 was gemik op OS/2 weergawe 2.0, wat die eerste 32 bis-weergawe van OS/2 was. Die betaweergawes van OS/2 2.0 was egter stadiger as die vorige weergawes (net 16 bis) ten spyte van die meer effektiewe geheue-adressering van die 32 bis-weergawe. Die vrystelling van OS/2 2.0 sou moontlik eers aan die einde van 1992 geskied. Microsoft het SQL Server vir die 16-bis OS/2 1.3-bedryfstelsel aangepas. Die OS/2 1.3-bedryfstelsel het egter net op IBM se tipe rekenaars gewerk. Die probleem is oorkom deurdat Microsoft 'n OS/2 1.3-weergawe (kodenaam: Tiger) vrygestel het wat Microsoft SQL Server en Microsoft LAN Manager ingesluit het. Microsoft SQL Server 4.2 se betaweergawe is gedurende 1991 vrygestel en die finale weergawe is in Maart 1992 vrygestel wat 'n gebruikerskoppelvlak, om administrasie te vergemaklik, ingesluit het.

Die meeste kliënte van SQL Server was nog op OS/2 en 'n 32 bis-weergawe van Microsoft SQL Server vir OS/2 2.0 is verwag. Die SQL Server ontwikkelaars moes so gou as moontlik 'n SQL Server produseer vir Windows NT. Microsoft het egter nie die aanvanklike 32 bis-ontwikkeling heeltemal laat vaar nie. Die probleem om die nuwe SQL Server te ontwikkel vir beide OS/2 2.0 en Windows NT het ontstaan. Die besluit is geneem om die ontwikkeling vir OS/2 2.0 te laat vaar en op Windows NT te konsentreer wat op daardie stadium geen kliëntebasis gehad het nie. Slegs foutregstelling en onderhoud is vir die bestaande OS/2 SQL Server gedoen. Baie gebruikers wat op OS/2 wou bly, was ongelukkig met hierdie besluit.

Sybase het intussen 'n nuwe produk vir Unix genaamd System 10 ontwikkel. Sybase het vereis dat Microsoft SQL Server verenigbaar met System 10 moet wees, maar Microsoft het prioriteit gegee aan Windows NT. Microsoft het SQL Server 4.2 vir OS/2 oorgeskakel na Windows NT. Sybase het Windows NT as een van die kernbedryfstelsels vir System 10 ingebring. Die OS/2-produk is oorgedra aan Sybase sodat die gebruikers wat op OS/2 wou bly, dit wel kon doen.

Die nuwe SQL Server was verenigbaar met die vorige OS/2-weergawe en met die nuwe Sybase-produkte. Die nuwe SQL Server se naam was Microsoft SQL Server for Windows NT (intern bekend as SQL NT). In Julie 1992 het Microsoft 'n voorbetaweergawe van Windows NT vrygestel saam met programmeringsbiblioteke wat benodig word om toepassings van OS/2 en 16 bis Windows na Windows NT oor te dra. Microsoft het terselfdertyd 'n onderhoudweergawe vir die OS/2 SQL Server vrygestel wat verskeie foutregstellings behartig het.

In Julie 1993 is Windows NT 3.1 vrygestel en binne 30 dae is die eerste weergawe van Microsoft SQL Server for Windows NT ook vrygestel. Die meeste OS/2-gebruikers was teen Desember 1993 oorgeskakel na die nuwe SQL Server en die meeste oorblywendes het aangetoon dat hulle graag wou oorskakel, ten spyte van die ontwikkeling van Sybase se System 10 vir OS/2. Die opgradering van OS/2 se SQL Server na Windows NT het sonder probleme plaasgevind.

Die sukses van SQL Server het veroorsaak dat Sybase se inkomste begin afneem het. Alhoewel Sybase steeds 'n winsaandeel verkry het, was die aandeel nie voldoende om die swak verkope van hulle eie produk te delg nie. Die verhouding tussen Sybase en Microsoft het begin verander en hul prioriteite het begin verskil. Op 12 April 1994 is die ooreenkoms tussen Microsoft en Sybase verbreek. Microsoft het alleenreg op die Microsoft SQL Server behou, waar Sybase die System 10-produk na Windows NT kon oorbring. Beide produkte het steeds vorige weergawes ondersteun.

Die volgende weergawe van SQL Server (bynaam SQL95) het aspekte soos replikasie[8] en "scrollable cursors"[9] ingesluit. Die onwikkelingspan het aan 'n nuwe stel administratiewe toepassings gewerk (kodenaam: "Starfighter") – wat vandag se SQL Server Enterprise Manager geword het. Die voorgestelde datum vir die vrystelling van die nuwe SQL Server was in 1995. Die eerste betaweergawe is egter teen die einde van Oktober 1994 vrygestel. Verskeie betabywerkings het die vrystelling gevolg, totdat die finale produk, genaamd Microsoft SQL Server 6.0, op 14 Junie 1995 gelewer is.

[8] Microsoft Corporation (2005a) verduidelik replikasie as die kopiëring en verspreiding van data- en databasis-objekte van een databasis na 'n ander, asook die sinkronisasie tussen die databasisse ter wille van konsekwentheid.

[9] Microsoft Corporation (2005b) definieer "cursors" as die meganisme waardeur daar met een ry op 'n slag vanaf 'n stel rekords gewerk kan word.

Die ontwikkelaars kon as gevolg van mededingers soos Oracle, Informix, IBM en Sybase nie ophou werk nie. Die 6.5-weergawe het elemente ingesluit wat in weergawe 6.0 uitgelaat was as gevolg van die skaal van die projek, soos byvoorbeeld internet en datapakhuiskapasiteit. 'n Beta 6.5-weergawe is op 15 Desember 1995 vrygestel en die finale weergawe is in April 1996 uitgebring.

Gedurende hierdie tydperk (1995) was 'n ander groep Microsoft-ontwikkelaars besig om 'n nuwe navraagverwerker te ontwikkel wat later die Microsoft Data Engine (MSDE) sou word. Terselfdertyd is OLE DB ontwikkel, wat toegelaat het dat sekere elemente van SQL Server se kernproduk opgebreek kon word en as onafhanklike komponente ontwikkel kon word. Die komponente kon met OLE DB as medium kommunikeer. Die nuwe navraagverwerkerkomponent kon teen die einde van 1995 in die SQL Server-kode geïmplementeer word en die ontwikkelling van die nuwe SQL Server (kodenaam: Sphinx) het begin met die twee spanne wat saamwerk. Klem is daarop gelê om die SQL Server só te ontwikkel dat dit kon uitbrei soos wat gebruikers en data bygevoeg word. Die nuwe komponentstruktuur het toegelaat dat nuwe algoritmes enige tyd ingevoeg kon word. Die oorhoofse mikpunt was sluitingmeganismes vir rye en 'n nuwe navraagverwerker wat ad hoc-navrae goed kon hanteer. Die SQL Server 7.0 beta 1-weergawe het in 1997 verskyn en die beta 2-weergawe is in Desember 1997 vrygestel. Beta 3 is in Junie 1998 vrygestel, en die finale weergawe is op 16 November 1998 aangekondig en op 2 Desember 1998 vrygestel, en was toeganklik vir die publiek in Januarie 1999.

Die volgende weergawe (kodenaam: Shiloh) was oorspronklik net 'n dienspakket ("Service pack") vir die 7.0-weergawe, maar na twee dienspakkette uitgegee is het Shiloh 'n volwaardig weergawe met weergawenommer 7.5 geword. Die strewes van die nuwe weergawe was die verbetering van werkverrigting, saam met nuwe bykomende elemente. In September 1999 is die beta 1-weergawe vrygestel, en 'n kort tydjie daarna het Microsoft aangekondig dat die produk SQL Server 2000 sou wees. Die interne weergawe is egter 8.00.xxx – afhangende van die tipe installasie. Beta 2 is in April 2000 afgegee en op 9 Augustus 2000 is die finale produk vrygestel.

3.2.2 'n Oorsig oor die elemente binne SQL Server

Hierdie gedeelte word volgens die werk van Delaney (2001:31-66) beskryf. Microsoft SQL Server 2000 ondersteun verwerkings van grootvolumetransaksies, datapakhuise[10] en besluitsteunstelsels.

[10] Kimball *et al.* (1998:19) definieer 'n datapakhuis as 'n databron waarop navrae gerig kan word wat nie op die relasionele databasismodel geskoei is nie aangesien dit die model bemoeilik en die werkverrigting belemmer.

SQL Server is ontwikkel om op Windows NT Workstation 4, Windows 2000 Professional, Windows 98 en Windows Millennium Edition (Me) te funksioneer. SQL Server 2000 maak gebruik van Transact-SQL-navraagsintaksis wat op die ANSI SQL-92-standaard gebaseer is. 'n Kort beskrywing van die sintaksis kan in Bylaag A gevind word. Die volgende paragrawe beskryf kortliks die outomatiese optimeerder, programmeringsfunksionaliteit, gestoorde prosedures, databasisintegriteit, simmetriesebediener-argitektuur, data-duplisering, verskeie toepassings en kliëntontwikkelingskoppelvlakke. Sommige van die onderwerpe word later in die verhandeling in meer detail bespreek.

3.2.2.1 Outomatiese optimeerder van SQL Server

'n Outomatiese optimeerder word gebruik om die beste wyse te bepaal om die data vir die navraag te manipuleer, hetsy dit seleksie, bywerking of byvoeg is. Die optimeerder stel die gebruiker vry van die taak om te kies watter indekse gebruik moet word vir die navraag. Statistiese inligting oor die volume en verspreidheid van die data word uitgewerk, wat dan weer verder gebruik word om die beste moontlike uitvoerplan te kies. Dit is egter steeds moontlik om die optimeerder te forseer in die keuses vir die uitvoerplan deur die leidrade-sintaksis te gebruik (sien paragraaf 3.3 in Bylaag A).

3.2.2.2 Programmeringsfunksionaliteit van SQL Server

Transact-SQL verskaf programmeringsfunksionaliteit, soos byvoorbeeld veranderlikes, voorwaardelike sintaksis (if-then-else) en herhaling-sintaksis (while). Die moontlikheid ontstaan dus dat bewerkings wat andersins 'n toepassing sou vereis het, nou binne SQL Server gebruik kan word. Dit is dus nie nodig om oor die netwerk te kommunikeer nie, want die bewerkings vind lokaal op die bediener self plaas. SQL Server het 'n ingeboude vertaler (T-SQL Debugger) wat gebruik word om sintaksisfoute binne die Transact-SQL-kode op te spoor.

3.2.2.3 Gestoorde prosedures

Gestoorde prosedures en funksies is nog 'n funksionaliteit van SQL Server. Hiermee kan SQL-uitdrukkings vooraf ontleed, toegangsregte verifieer en 'n optimale uitvoerplan uitwerk en saam met die vorm van die uitdrukking stoor. Dit maak die uitvoer van die gestoorde prosedures vinniger vir 'n groot hoeveelheid uitdrukkings aangesien die uitdrukkings net een keer ontleed word. 'n Prosedure en funksie verskil in dié opsig dat prosedures 'n sekere klomp take uitvoer en geen waarde teruggee nie, terwyl funksies 'n waarde teruggee vir gebruik. Die ander voordeel van gestoorde prosedures en funksies is dat dit 'n sentrale plek bied waar kode een keer kan verander, en dit dan vir elke program wat die prosedure of funksie roep, sal reflekteer.

Uitgebreide gestoorde prosedures, wat elemente buite SQL Server kan bereik, is ook beskikbaar, soos byvoorbeeld prosedures wat HTML-lêers ("Hypertext Markup Language"-lêers) kan skep. Microsoft self gebruik van hierdie metodes. Dit is byvoorbeeld moontlik om e-pos te stuur met een van hierdie gestoorde prosedures: "xp_sendmail" of "xp_readmail".

3.2.2.4 Databasisintegriteit

'n Belangrike deel van enige databasis is die data se integriteit. Ten opsigte van optimering is dit belangrik om daarop te let dat hoe meer integriteitsvoorwaardes gestel word, hoe meer toetse moet die SQL-enjin behartig voordat daar byvoorbeeld 'n rekord bygevoeg kan word. SQL Server bied verskeie databasisintegriteitsmoontlikhede:

- Deklarasiedata-integriteit – Die integriteitsmoontlikheid stel integriteitsbeperkings op wanneer die databasistabel geskep word. Die eerste tipe beperking hier ter sprake is sogenaamde "entiteit-integriteit" waar elke ry van 'n tabel uniek geïdentifiseer word deur 'n attribuut of kolom (mag meer as een wees) wat as die primêre sleutel bekend staan. Entiteit-integriteit word verseker deur hierdie primêre sleutelbeperking. Verskeie ander kolomme (behalwe die primêre sleutel) kan ook die ry uniek identifiseer, en staan bekend as alternatiewe of kandidaatsleutels. SQL Server bied die funksie om 'n kolom (identiteitskolom) te skep wat outomaties nommers genereer wat uniek vir elke ry is.

  Die tweede beperking onder deklarasie-integriteit is verwysingsintegriteit. Vreemdesleutelbeperkings ("foreign keys") word gebruik om verwysings tussen tabelle se integriteit te verseker. Die vreemde sleutel is 'n kolom of kolomme wat na die primêre of alternatiewe sleutel van 'n ander tabel verwys en so die twee tabelle koppel.

  Die laaste tipe beperking is gebied- of domeinintegriteit. Voorbeelde hiervan is persentasies wat net tussen 0 en 100 mag wees en kolomme wat nie "NULL"-waardes[11] mag aanneem nie indien die beperking van geen "NULL"-waardes gestel is.

- Datatipes – Elke kolom mag net sekere tipes data bevat, byvoorbeeld karakters, getalle en datums. Die gebruiker kan ook sy/haar eie tipes definieer.

- "CHECK"-beperkings en reëls – Slegs waardes wat aan die beperkings en reëls voldoen, mag gedurende 'n byvoeg- of bywerktransaksie aanvaar word vir die kolom waarvoor hierdie beperkings en reëls geld.

[11] "NULL"-waardes dui onbekende waardes vir 'n kolom aan.

Die beperkings bou op die datatipe-beperking om verdere beperkings op die data toe te pas. Die beperkings vereis dat waardes binne bepaalde grense val, of 'n sekere patroon volg, of gelyk aan elemente in 'n spesifieke lys is.

- Verstekwaardes – Indien daar tydens byvoeging geen waarde vir die kolom gegee word nie, word die verstekwaarde, indien enige, in die kolom bygevoeg.

- "Triggers" – "Triggers" is gestoorde prosedures wat outomaties geroep word. 'n Voorbeeld hiervan is 'n persoon met 'n agterstallige rekening, wat nie verwyder mag word nie. Die prosedure verhoed dat die persoon se rekords verwyder word. Du & Wolfe (1997:819) en Atzeni *et al.* (1999:447) definieer databasisse wat gebruik maak van hierdie prosesse as "aktiewe databasisse".

3.2.2.5 Simmetriesebediener-argitektuur

SQL Server verskaf verbeterde werkverrigting deur 'n tegniek wat as simmetriesebediener-argitektuur bekend staan. Volgens Delaney (2001:44) gebruik dié tegniek slegs één geheue-area vir die databasisbestuurstelsel wat die werk verminder aangesien gedeelde geheue nie bestuur moet word nie.

Die tradisionele metodes het behels dat argitektuur 'n aantal prosesse (een vir elke gebruiker) gebruik het, of net een proses wat 'n bedryfstelsel se "threading"-tegniek naboots. Vir die metode waar elke gebruiker 'n proses gebruik, moet gedeelde geheue gebruik word vir die prosesse om met mekaar te kan kommunikeer. Die metode is oneffektief aangesien baie verwerking vir die beheer van die verdeling benodig word. Die "threading"-tegniek skuif tussen die verskillende opdragte volgens 'n "round robin"-algoritme. Tanenbaum en Woodhull (1997:84) stel dat elke proses 'n vaste tydkwantum in die "round robin"-algoritme verkry. 'n Lys word opgebou soos elke proses vir uitvoer beskikbaar raak. Elke proses kry vir die tydsduur van die kwantum geleentheid om uit te voer volgens die proses se posisie in die lys. Sodra die proses uitgevoer is, word die proses weer aan die einde van die lys geplaas. Hierdie prosedure word herhaal totdat al die prosesse klaar uitgevoer het. Die "threading"-metode se nadeel is daarin geleë dat indien een van die "threads" sou breek, al die ander "threads" ook in gevaar is.

SQL Server 2000 gebruik 'n algoritme wat tussen die twee tradisionele metodes is. Windows NT / 2000 ondersteun liggewig "threads" wat "fibers" genoem word. SQL Server maak gebruik van een of meer bedryfstelsel "threads" as die toepassing se skeduleerders ("User Mode Schedulers (UMS)"). SQL Server gebruik een "UMS thread" per sentrale verwerkingseenheid (SVE, "Central Processing Unit (CPU)"). Elke SQL Server "fiber" is gekoppel aan een "UMS

thread" en die "fiber" bly by hierdie "UMS thread" vir die "fiber" se leeftyd. Windows NT / 2000 kan elke "UMS thread" skeduleer op enige van die beskikbare verwerkers. Elke "UMS thread" bepaal watter SQL Server prosesse wat met die "UMS thread" geassosieer is, mag uitvoer. Hierdie tegniek van SQL Server bied verbeterde werkverrigting vir 'n groter aantal gebruikers. Hiermee kan SQL Server honderde of selfs duisende gebruikers gelyktydig hanteer.

3.2.2.6 Data-duplisering

SQL Server besit die funksionaliteit om data te dupliseer na 'n ander bediener vir rugsteun en groepering ("clustering") wat toelaat dat dienste ononderbroke voortgaan al is een van die bedieners buite werking (sogenaamde "failover"). Dit moet in gedagte gehou word dat die duplisering van hierdie data 'n uitwerking op die produktiwiteit van die databasis kan hê. Die drie tipes dupliseringsmetodes van SQL Server is soos volg:

- Transaksie-replikasie – Die een bediener (uitgewer) is die eienaar van die data en die item (tabel of gedeelte van die tabel) word na ander bedieners wat ingeteken is om die veranderings te ontvang, gestuur sodra 'n verandering plaasvind.
- Saamvoegreplikasie ("merge replication") – Enige intekenaarbediener kan veranderings maak, wat dan na die res van die bedieners versprei word. Verskeie reëls kan opgestel word om die bywerkingskonflik te beheer.
- Momentopname-replikasie ("snapshot replication") – Die uitgewerbediener stuur die data periodiek na die ander intekenaarbedieners.

3.2.2.7 Toepassings in SQL Server

Die pakket van SQL Server 2000 sluit ook toepassings in wat gebruik word om die databasis en verwante take te bestuur. Hier volg 'n paar van die belangrikste toepassings (sien paragraaf 3.2.8 vir 'n vollediger beskrywing van sekere van hierdie toepassings):

- "SQL Server Enterprise Manager" – Die toepassing stel die gebruiker in staat om byvoorbeeld take soos die toekenning van sekuriteit aan gebruikers met die hulp van grafiese koppelvlakke te behartig.
- "SQL Server Agent" – Die agent is 'n diens wat op Windows loop en saam met die bedryfstelsel gelaai word. Die agent verskaf 'n skeduleringsenjin wat take volgens 'n skedule aktiveer en loop. Die gebruiker kan dus sekere take, soos die skoonmaak van 'n tabel, skeduleer en die agent voer dan die take uit.
- "SQL Profiler" – Die toepassing verskaf 'n manier om die aktiwiteit van 'n databasis te monitor om byvoorbeeld te bepaal watter prosedures die meeste op die databasis loop en

sodoende 'n optimeringstrategie vir die databasis te ontwikkel. 'n Belangrike aspek is dat die SQL Profiler direk kan integreer met die Windows NT/2000-bedryfstelsel se eie produktiwiteit-moniteringstoepassings om sodoende inligting oor byvoorbeeld die geheue se gebruik, die aantal gebruikers, die transaksies per sekonde, en die persentasie van die sentrale verwerker se kapasiteit wat gebruik word, te verkry. Ander aspekte van hierdie toepassing word in paragraaf 3.2.8.1 hanteer.

- "SQL Server Service Manager" – Hierdie toepassing verskaf 'n maklike manier om SQL Server, SQL Server Agent en 'n paar ander dienste te stop of te laat uitvoer.
- "SQL Query Analyzer" – Die toepassing verskaf 'n grafiese koppelvlak wat die implementering van navrae vergemaklik. Die toepassing verskaf funksies soos 'n grafiese uitvoerplan van die stappe wat die optimeerder vir die betrokke navraag gekies het. Sien paragraaf 3.2.8.2 vir 'n vollediger beskrywing.

3.2.2.8 Kliëntontwikkelingskoppelvlakke in SQL Server

Die belangrikste koppelvlakke is ODBC ("Open Database Connectivity"), RDO ("Remote Data Objects"), OLE DB ("Object Linking and Embedding Database"), ADO ("ActiveX Data Objects"), DB-Library en ESQL/C ("Embedded SQL for C"). Aangesien hierdie koppelvlakke ook 'n impak op die produktiwiteit van 'n databasis kan hê, volg 'n kort beskrywing van elk:

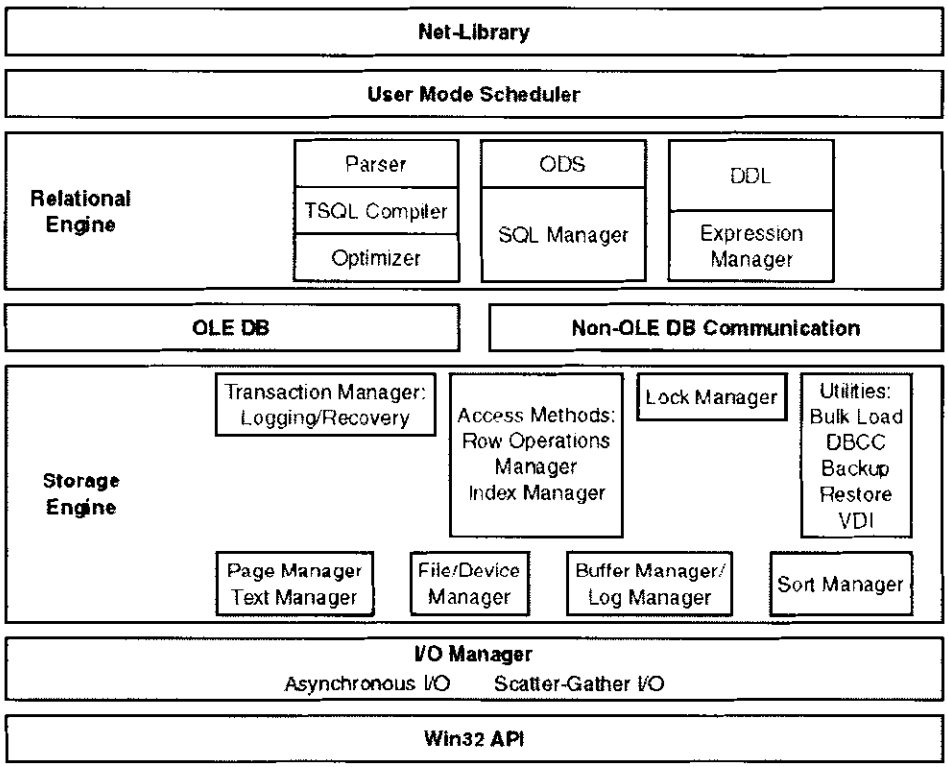
- ODBC – Die koppelvlak verskaf 'n toepassingsprogrammeringskoppelvlak (API) tot die databasis. Die koppelvlak is die formele standaard volgens ANSI ("American National Standards Institute") en ISO ("International Organization for Standardization"). Microsoft Corporation (2005c) definieer ODBC as 'n C-koppelvlak (programmeertaalkoppelvlak) wat dit moontlik maak vir toepassings om toegang tot data te verkry deur 'n verskeidenheid van databasisbestuurstelsels. Die toepassing is onafhanklik van die databasisbestuurstelsel.
- RDO – 'n Objekkoppelvlak wat baie na aan ODBC en geredelik beskikbaar vir Visual Basic programme is. RDO ondersteun die bou van visuele komponente wat direk tot SQL Server se data verbind is.
- OLE DB – Die koppelvlak verskaf 'n COM-koppelvlak ("Component Object Model"-koppelvlak) tot tabelformaat databronne wat sigblaaie (byvoorbeeld Excel) en tekslêers saam met databasistabelle insluit. OLE DB is 'n objekweergawe van ODBC, maar kan meer toegang tot data van die onderskeie databronne verkry.
- ADO – Die koppelvlak is 'n hoëvlakobjekkoppelvlak wat op OLE DB voortbou en vir die programmering van internettoepassings voorgestel word.
- DB-Library – Hierdie SQL Server API is die oorspronklike SQL Server-programmeringskoppelvlak.

- ESQL/C – SQL Server verskaf 'n ESQL/C-vertaler wat toelaat dat ontwikkelaars die SQL-navraagsintaksis direk in hulle C-bronkode kan plaas.

3.2.3 Die interne argitektuur van die SQL Server-stelsel

Die volgende gedeelte beskryf die interne werking van die SQL Server-stelsel en is gebaseer op Delaney (2001:69-111). Die argitektuur sal aan die hand van figuur 3.1 verduidelik word.

Figuur 3.1 – Die hoofkomponente van SQL Server-argitektuur (Delaney, 2001:71)



3.2.3.1 Die "Net-Library" van SQL Server

Die abstraksievlak maak dit moontlik vir SQL Server om met verskeie netwerkprotokolle te kommunikeer. Die komponent word deur SQL Server op beide bediener- en kliëntrekenaars gebruik en verskeie kliëntrekenaars met verskillende netwerke kan gelyktydig ondersteun word. Twee primêre biblioteke is beskikbaar, naamlik "Super Socket" en gedeelde geheue. Verskeie sekondêre biblioteke, soos byvoorbeeld TCP/IP en IPX/SPX, is ook beskikbaar. Kommunikasie tussen rekenaars word deur die "Super Socket Net-Library" hanteer, terwyl lokale kommunikasie tussen 'n toepassing en die SQL Server deur die gedeelde geheue "Net-Library" plaasvind (as dit geaktiveer is, wat wel die verstekopstelling is).

Die "Super Socket Net-Library" het twee komponente: 'n kommunikasie pad (kommunikasie tussen die toepassing en die SQL Server) en 'n enkripsievlak (gebruik "Secure Socket Layer (SSL) API (Application Programming Interface)"). Hierdie enkripsie kan moontlik die netwerk stadiger maak as gevolg van die ekstra werk van enkripsie en dekripsie wat tussen die kliënt en bediener plaasvind.

Teoreties is "TCP/IP Sockets Net-Library" die vinnigste "Net-Library", maar die spoed van die netwerk het 'n klein impak op die algehele werkverrigting. Enkripsie se koste is ook weglaatbaar klein en die meeste toepassings sal nie die verskil agterkom nie.

3.2.3.2 Die "User Mode Scheduler" van SQL Server

Die "User Mode Scheduler (UMS)" is 'n bedryfstelsel "thread" wat deur SQL Server gebruik word. Die skedulering van "fibers" word deur die "User Mode Scheduler" hanteer. Slegs een "User Mode Scheduler" word vir elke SVE van die rekenaar gebruik. Elke "User Mode Scheduler" bepaal watter SQL Server prosesse wat met die "UMS thread" geassosieer is, mag uitvoer.

3.2.3.3 Die "Open Data Services (ODS)" van SQL Server

Die gedeelte is 'n kliëntbestuurder vir SQL Server wat tussen die bediener "Net-Libraries" en verskeie bedienertoepassings (wat SQL Server insluit) as koppelvlak dien. ODS luister vir nuwe verbindings, verwyder foutiewe verbindings, reageer op kansellasie van bevele ("attentions"), koördineer "thread"-dienste na SQL Server en stuur resultaatstelle, boodskappe en statusse terug aan die kliënt wat boodskappe aan SQL Server stuur.

SQL Server-kliënte en die bediener self gebruik 'n vorm van kommunikasie wat bekend staan as "Tabular Data Stream" (TDS) wat inligting oor kolomname, datatipes, gebeurtenisse (byvoorbeeld kansellasies) en statuswaardes deurgee. Die koppelvlakke OLE-DB, ODBC en "DB-Library" word gebruik om in TDS te kommunikeer en nie die kliënt of bediener nie.

SQL Server besit twee invoerbuffers en een uitvoerbuffer. Die rede vir die twee invoerbuffers is dat 'n bevel ("attention") moontlik kan deurkom terwyl SQL Server data lees. Die een uitvoerbuffer kan moontlik 'n bottelnek veroorsaak as die kliënt nie data vanaf die netwerk kan lees nie. Die netwerkkasgeheue ("cache") hoop op en SQL Server hou op om skryfinstruksies te stuur. Die grootte van die uitvoerbuffer kan die spoed beïnvloed waarteen die kliënt die eerste resultaatstel ontvang. Gestel twee navrae word gelyk gestuur en dat die eerste navraag 'n klein hoeveelheid data besit. In so 'n geval sal die resultate eers teruggestuur word wanneer

die tweede navraag klaar is of sodra daar voldoende data is om die uitvoerbuffer vol te maak. Nog 'n geval wat die invloed van die uitvoerbuffer se spoed illustreer, is wanneer een van die navrae stadig uitvoer en altwee navrae dan moet wag vir die spesifieke navraag om te voltooi. Die oplossing is om die een navraag as 'n aparte bondel te sien. Soms is dit egter meer voordelig om navrae in 'n bondel te plaas sodat die prosessering tussen kliënt en bediener verminder word.

ODS bied aan die bediener se kant dieselfde funksionaliteit as wat ODBC, OLE DB en "DB-Library" aan die kliënt se kant bied. Die bedienertoepassing reageer op gebeurtenisse wat veroorsaak word deur versoeke van die bedieners en kliënte. Die bediener-ODS reageer op bindgebeurtenisse (SQL Server doen sekuriteitstoetse vir reg van toegang), taalgebeurtenisse (SQL Server gee bevelstringe aan die bevelontleder ("command parser")) en afgeleë gestoordeprosedure-gebeurtenisse (die kliënt en bediener is geskei deur 'n groot fisiese afstand).

ODS beheer "threads" en "fibers" vir SQL Server op 'n laer vlak as die "User Mode Scheduler". Dit sluit die skep, terminering en die beskikbaarstelling van die "thread" aan die "User Mode Scheduler (UMS)" in.

3.2.3.4 Die relasionele enjin van SQL Server

Al die komponente wat benodig word om navrae te vertaal, te optimeer en uit te voer word in hierdie gedeelte vervat. Die verskeie aspekte is soos volg:

- Bevelontleder ("Command parser") – Die komponent hanteer die taalgebeurtenisse van ODS. Sintaksistoetsing en vertaling van die Transact-SQL-bevele na 'n interne formaat (navraagboom) word deur hierdie aspek behartig.
- Optimeerder of navraagverwerker ("Optimizer") – Die navraagboom word hier voorberei vir uitvoering. Die bevele-bondel word vertaal, die navrae word geoptimeer en die sekuriteit word getoets. Hierdie proses verskaf 'n uitvoerplan as resultaat. Die navraagverwerker word breedvoeriger in paragraaf 3.2.7 beskryf.
- SQL Bestuurder ("SQL Manager") – Die komponent hanteer gestoorde prosedures en hul uitvoerplanne. Die komponent bepaal of die prosedure weer as gevolg van veranderings in die onderliggende objekskema vertaal moet word. Kasberging ("Caching") van die planne vir hergebruik word hier behartig. Nog 'n belangrike funksie is die outo-parametrisering van navrae (sien paragraaf 3.2.7.5). Dit behels dat sekere navrae hanteer word as geparametriseerde gestoorde prosedures (gewoonlik 'n navraag wat 'n eenvoudige

gelykheidstoets met 'n konstante behels). Die volgende navraag waarvan net die konstante verskil, maak dan van die geparametriseerde gestoorde prosedure gebruik.

- Uitdrukkingbestuurder ("Expression manager") – Bewerkings, vergelykings en data-oordragte word hier hanteer.
- Navraaguitvoerder ("Query executor") – Die uitvoerplan met al sy bevele word hier uitgevoer deur elke stap van die bondel deur te werk.

3.2.3.5 Die kommunikasievlak tussen die relasionele enjin en die stoorenjin

Die hoofkommunikasiemiddel is OLE DB. Die relasionele enjin gebruik die OLE DB API om die stoorenjin te vra om die resultaatstelle van die navraag beskikbaar te stel. Soos wat die relasionele enjin deur die stappe van die uitvoerplan werk, word OLE DB voortdurende gebruik indien rye vanaf die oop resultaatstelle verkry moet word. Die stoorenjin dra dan die data vanaf die databuffers oor na die relasionele enjin. Die relasionele enjin neem die data vanaf die stoorenjin en voeg dit saam in die finale resultaatstel wat aan die gebruiker vertoon word.

3.2.3.6 Die stoorenjin van SQL Server

Die enjin bevat die komponente wat benodig word om toegang tot die gestoorde data te verkry en die data te wysig. Die komponente word in die volgende paragrawe beskryf.

3.2.3.6.1 Transaksiebestuurder ("Transaction Manager")

Die bestuurder koördineer stawing[12]. Transaksies is volgens Balling en Zawodny (2004:22) SQL-navrae wat as 'n eenheid hanteer word), herstel ("recovery") en bufferhantering. Die bestuurder bepaal die grense van die uitdrukkings wat bymekaar gegroepeer moet word om 'n bewerking te vorm. Transaksies oor verskeie databasisse in dieselfde SQL Server, sowel as geneste transaksies, word deur die transaksiebestuurder beheer.

Die bestuurder merk spesiale stoorpunte in elke transaksie waarna die databasis weer herstel kan word met dieselfde waardes soos geneem voor die transaksiegedeelte begin uitvoer het. Die transaksiebestuurder werk saam met die sluitbestuurder ("Lock manager") om die sluit van transaksies te beheer volgens die isolasievlak wat daarvoor gestel is. Die isolasievlakke is opgedeel in vier tipes, naamlik onbevestigdelees- ("uncommitted read"), bevestigdelees- ("committed read"), herhaalbarelees- ("repeatable read") en die "serializable"-vlak.

[12] Stawing of "logging" stoor inligting in 'n log oor transaksies.

Die isolasievlakke word kortliks bespreek.

- Onbevestigde lees ("Uncommitted read") – Die transaksie kan enige data op die databladsy lees, ongeag of die data bevestig is of nie. Die probleem ontstaan deurdat 'n ander gebruiker data verander terwyl die transaksie data lees. Die veranderde waarde word dus gelees. Die gebruiker se transaksie word teruggerol en die verkeerde waarde is gelees.
- Bevestigde lees ("Committed read") – Hierdie verstekisolasievlak verseker dat die transaksie wag vir die bevestiging voordat data gelees word.
- Herhaalbare lees ("Repeatable read") – Die vlak werk bo-op die bevestigdelees-vlak deur te verseker dat die data nie verander vir herhaalde navrae wat kort opmekaar volg nie.
- "Serializable" – As 'n navraag weer geprosesseer moet word, mag rye nie tussen die twee navrae bygevoeg word nie. Die vlak bou op die herhaalbarelees-vlak.

3.2.3.6.2 Toegangmetodesbestuurder ("Access Methods Manager")

Die bestuurder doen 'n soektog vir databladsye en indeksbladsye om die data te verkry, en berei dan die OLE DB-resultaatstel voor wat dan aan die relasionele enjin teruggestuur word. Van die dienste wat die bestuurder besit, is om 'n tabel oop te maak, die kwalifiserende data te onttrek en die data by te werk. Die bestuurder onttrek eintlik nie self die bladsye nie, maar rig 'n versoek aan die bufferbestuurder wat die bladsye vanaf die kasgeheue lees of dit vanaf die skyf na die kasgeheue skryf en dit dan lees. Wanneer die soektog begin, word 'n vooruitkykmetode ("look-ahead method") gebruik om die kwalifiserende rye of indekse op 'n bladsy te verkry. Hierdie proses waar rye verkry word wat aan sekere kriterium voldoen, staan bekend as 'n kwalifiserende onttrekking ("qualified retrieval"). Die bestuurder word ook vir bywerk- en verwyderbewerkings gebruik.

3.2.3.6.3 Ry-bewerkingbestuurder ("Row Operations Manager")

Hierdie bestuurder is eintlik saam met die indeksbestuurder deel van die toegangmetodesbestuurder aangesien die bestuurders die werklike metode van toegang behartig. Die bestuurder onttrek, verander en doen bewerkings op individuele rye data. SQL Server ondersteun drie bywerkmetodes:

- Inplekmetode ("In-place method") – Die nuwe data word geskryf na dieselfde posisie op die databladsy. Die metode word gebruik vir 'n "heap" of gebondelde indeks (sien paragraaf 2.4.1 vir 'n beskrywing van indekse) wanneer die gebondelde sleutels ("clustered keys") nie verander het nie.

- Verdeelmetode ("split method") – Die metode word gebruik om nie-unieke indekse by te werk wanneer die sleutel verander. Die bywerking is verdeel in verwyder- en daarna byvoegbewerkings. Die bewerkings is onafhanklik van mekaar.
- Verdeel-met-ineensakking-metode ("split with collapse method") – Die metode word gebruik vir die bywerk van 'n unieke indeks wanneer die indekssleutel verander. Die verwyder- en byvoegbewerkings word saamgevoeg in 'n enkele bywerkbewerking.

3.2.3.6.4 Indeksbestuurder ("Index Manager")

Die bestuurder ondersteun soektogte op B-bome wat vir SQL Server se indekse gebruik word. Die verduideliking van B-bome is in paragraaf 2.4.1.1 gegee. Indekse is een van die kern tegnieke van optimering.

3.2.3.6.5 Slotbestuurder ("Lock Manager")

Die bestuurder hanteer die sluit van data vir konsekwentheid in 'n multigebruikeromgewing.

3.2.3.6.6 Bladsybestuurder en teksbestuurder ("Page Manager and Text Manager")

Die twee bestuurders werk saam om 'n versameling van bladsye as 'n databasis te hanteer. Elke databasis bestaan uit 'n versameling van 8 kilogreep-skyfbladsye wat oor een of meer fisiese lêers versprei is. SQL Server bevat agt tipes skyfbladsye, naamlik databladsye, teks/beeld-bladsye, indeksbladsye, "Page Free Space (PFS)"-bladsye, "Global Allocation Map (GAM en SGAM ("Shared" GAM))"-bladsye, "Index Allocation Map (IAM)"-bladsye, "Bulk Changed Map"-bladsye en "Differential Changed Map"-bladsye. Die detail van hierdie skyfbladsye volg in paragraaf 3.2.5.

3.2.3.6.7 Lêer- en Rugsteuntoestelbestuurder ("File / Device Manager")

Die bestuurder beheer die skryf- en lees-aksies van lêers en die rugsteuntoestelle.

3.2.3.6.8 Bufferbestuurder ("Buffer Manager")

Die bestuurder hanteer skyf-lees- en -skryf-aksies om data- en indeksbladsy na die bufferarea ("buffer pool") te bring sodat gebruikers die data mag deel.

3.2.3.6.9 Logbestuurder ("Log Manager")

Alle veranderings word eers na die transaksielog geskryf voordat die werklike verandering na die skyf geskryf word. Dit is nodig vir databasisherstel tot 'n stabiele vlak. Alle skryf na die transaksielog volg mekaar en wag vir erkenning van die voltooiing van die aksie. Die logbestuurder formateer die transaksielogrekords in die geheue voordat hulle na die skyf geskryf word. Die logrekords word alleenlik in 'n geheue-area wat bekend staan as logkasgeheue-areas ("log caches") gestoor. Daar bestaan normaalweg twee of meer van hierdie logkasgeheue-areas waarvan een die huidige logkasgeheue-area is en die ander word gebruik sodra die huidige een vol is. Die logbestuurder het twee lyste ("queues") van logkasgeheue-areas, naamlik 'n "flushQueue" (bevat die logkasgeheue-areas wat na die hardeskyf uitgeskryf moet word) en 'n "freeQueue" (bevat 'n lys van leë logkasgeheue-areas). Die logskrywer is 'n "thread" wat deur die "flushQueue" beweeg en die rekords een vir een na die skyf skryf. Die bestuurder vorm deel van figuur 3.1.

3.2.3.6.10 Sorteerbestuurder ("Sort Manager")

Die bestuurder hanteer sorteerbewerkings.

3.2.4 Geheuehantering

Die volgende gedeelte word met behulp van Delaney (2001:91-101) verduidelik. SQL Server bevat 'n beleid van geheue-hantering om oorskakeling tussen verskillende Windows-bedryfstelsels en -hardeware te vergemaklik. Wanneer die optimale geheuegrootte vir SQL Server bereken word, probeer die geheue-bestuurder om ten minste 4 megagrepe ("megabytes") geheue oop te hou vir die bedryfstelsel. Die geheue-bestuurder maak 'n skatting van die gemiddelde leeftyd wat 'n bladsy in die kasgeheue is indien die bladsy nie gebruik word nie. Die skatting is gebaseer op die spoed waarteen die "lazywriter" (sien paragraaf 3.2.4.3 vir 'n beskrywing van die "lazywriter") kyk vir die verwysde bladsye en die totale hoeveelheid geheue wat beskikbaar is, gebaseer. Indien die gemiddelde leeftyd klein is, probeer die geheue-bestuurder die gereserveerde geheue by 4 megagrepe hou om sodoende die geheue aan SQL Server te vermeerder. Indien die leeftyd vergroot, word die gereserveerde spasie vir die bedryfstelsel verhoog tot 10 megagrepe en die beskikbare geheue vir SQL Server word verminder. Dit is moontlik om die gereserveerde geheue op 'n vaste waarde te stel. Die onderskeie elemente wat benodig word vir geheue-hantering word in die volgende paragrawe bespreek.

3.2.4.1 Die bufferbestuurder en geheue-areas ("Memory pools")

Die hoofgeheuekomponent van SQL Server is die bufferarea. Die bestuurder beheer die skyf-lees- en -skryf-aksies om data- en indeksbladsye in die bufferarea in te bring sodat die data tussen die gebruikers gedeel kan word. Wanneer komponente geheue benodig, word 'n buffer vanaf die geheue-areas aangevra. 'n Buffer is 'n bladsy in die geheue met dieselfde grootte as 'n data- of indeksbladsy. Die bedryfstelsel bevat ook 'n geheue-area wat slegs gebruik word wanneer SQL Server geheue in groter blokke as die 8 kilogreep-bladsye van die bufferarea versoek.

Die prosedurekasgeheue is ook 'n geheue-area waar navraagbome en planne van gestoorde prosedures, "triggers", gebruikersfunksies en "ad hoc"-navrae gestoor word. Nog geheue-areas word gebruik deur geheue-intensiewe navrae wat sortering of "hashing" gebruik en spesiale geheue-objekte wat minder as 8 kilogreep-bladsye benodig.

3.2.4.2 Toegang tot geheuebladsye

"Hashing" maak die toegang baie vinnig na die bladsye in die bufferarea. "Hashing" is 'n tegniek wat 'n sleutel uniform met 'n "hash"-funksie oor 'n stel "hash"-gleuwe ("buckets") versprei. 'n "Hash"-gleuf is 'n struktuur in geheue wat 'n skikking van wysers na bufferbladsye bevat. As al die wysers na bufferbladsye nie op 'n enkele "hash"-bladsy pas nie, word 'n geskakelde lys gebruik om die "hash"-bladsye met mekaar te verbind.

Die "hash"-funksie gebruik die databasis-id, lêernommer en bladsynommer om 'n sleutel van die "hash"-gleuf te verkry. Die "hash"-gleuf dien as 'n indeks tot die spesifieke bladsye wat benodig word. Indien die databladsy wat gesoek word, nie in die kasgeheue is nie, word dit vanaf die skyf in die kasgeheue ingelees.

3.2.4.3 Toegang tot vry-bladsye ("Lazywriter")

Data- of indeksbladsye kan slegs gebruik word, indien die bladsy in die geheue is. Dus moet 'n buffer beskikbaar wees waarin die bladsy ingelees kan word. Die beskikbaarheid van só 'n buffer is baie belangrik vir optimering.

Die bufferarea word deur die "lazywriter"-proses beheer en 'n klokalgoritme word gebruik om deur die bufferarea te lees. Die "lazywriter thread" gebruik 'n wyser om sekwensieel soos die wysers van 'n horlosie deur die bufferarea te lees. Wanneer die "lazywriter" by 'n buffer kom, word bereken of die buffer al sedert die vorige deurgaanproses gebruik is. Die aantal

verwysings word in die buffer se opskrif ("header") gestoor. As die aantal nie nul is nie, bly die buffer in die geheue en die verwysingsaantal word verminder. As die aantal wel nul is, word die buffer beskikbaar gestel vir hergebruik deur die buffer (indien nodig) na die skyf te skryf, vanaf die "hash"-lys te verwyder en in 'n spesiale lys van vry buffers geplaas. Die verwysingsaantal word elke keer vermeerder wanneer die buffer-inhoud deur 'n proses gebruik word. Data- en indeksbladsye word met een vermeerder, maar objekte soos gestoordeprosedure-planne wat duur is om te skep, kry 'n hoër verwysingsaantal wat op vervangingskoste dui.

Die "lazywriter" begin die deurgaanproses wanneer die aantal bladsye in die vrye lys onder 'n minimum grootte is. Die grootte word as 'n persentasie van die oorhoofse bufferareagrootte bepaal, maar is tussen 128 kilogrepe en 4 megagrepe. Die beskikbaarheid van vrye lyste word bepaal deur die aanvraag op die stelsel en die aantal stakings wat plaasvind. 'n Staking is wanneer 'n proses 'n vry-bladsy benodig en daar niks beskikbaar is nie. Die proses gaan in 'n slaapfase in en wag vir die "lazywriter" om bladsye vry te maak. SQL Server sal die minimum grootte van die vry-lys vermeerder indien baie stakings voorkom (drie of vier per sekonde) en andersom.

Dit is moontlik om tabelle te merk sodat hulle bladsye nooit in die vry-lys kom nie, maar altyd in die geheue bly (sogenaamde "pinning of a table").

3.2.4.4 Kontrolepunte ("Checkpoints")

Kontrolepunte verminder die werk vir SQL Server wanneer die databasis herstel moet word tydens die aansit van die stelsel. Die kontrolepunte skryf "dirty"-bladsye (gewysigde) van 'n databasis na die skyf sodat die veranderings nie verlore raak nie. SQL Server skryf met elke kontrolepunt 'n kontrolepuntrekord na die transaksie "log"-lêer. Dit verskaf aan die herstelproses 'n lys van moontlike gewysigde bladsye. Kontrolepunte word geaktiveer deur sekere gevalle. Die gevalle word hier genoem.

- 'n Eksplisiete bevel deur die databasiseienaar om 'n kontrolepunt te skep.
- As die log begin vol raak (meer as 70% van kapasiteit) en die databasis is in eenvoudige herstelmodus of die opsie "trunc. Log on chkpt." is gestel. Die kontrolepunt word geaktiveer om die transaksielog uit te vee ("truncate") en spasie vry te maak. As geen spasie vrygemaak kan word nie (as gevolg van lang transaksies), vind geen kontrolepunt plaas nie.
- As die hersteltyd moontlik langer as die herstelintervalparameter (verstekwaarde van 0) gaan wees, word 'n kontrolepunt geaktiveer.

In 'n stelsel met groot geheue kan kontrolepunte dalk baie skryfbewerkings uitvoer. SQL Server beperk die skryfbewerking tot 100 gelyktydige bewerkings. Die kontrolepuntalgoritme hou tred met watter bladsye alreeds geskryf is deur gebruik te maak van 'n generasienommer vir elke buffer in die kasgeheue. Kontrolepunte bespoedig herstel, maar het wel 'n impak op uitvoer-werkverrigting.

3.2.4.5 Hantering van bladsye deur die bufferbestuurder

Die bufferbestuurder hanteer die geheue-eweknie van elke fisiese skyfbladsy en verskaf aan ander modules veilige toegang tot die bladsye. Wanneer 'n benodigde bladsy nie in die bufferarea is nie, word dit met 'n fisiese lees- of skryf-aksie vanaf die skyf verkry. Fisiese lees- of skryf-aksies is duurder ten opsigte van die hoeveelheid werk wat moet plaasvind. 'n Bladsy word herken deur die databasisnommer-, lêernommer- en bladsynommer-kombinasie. Die bestuurder verskaf 'n wyser na die geheuebuffer met die betrokke bladsy aan die roepende module.

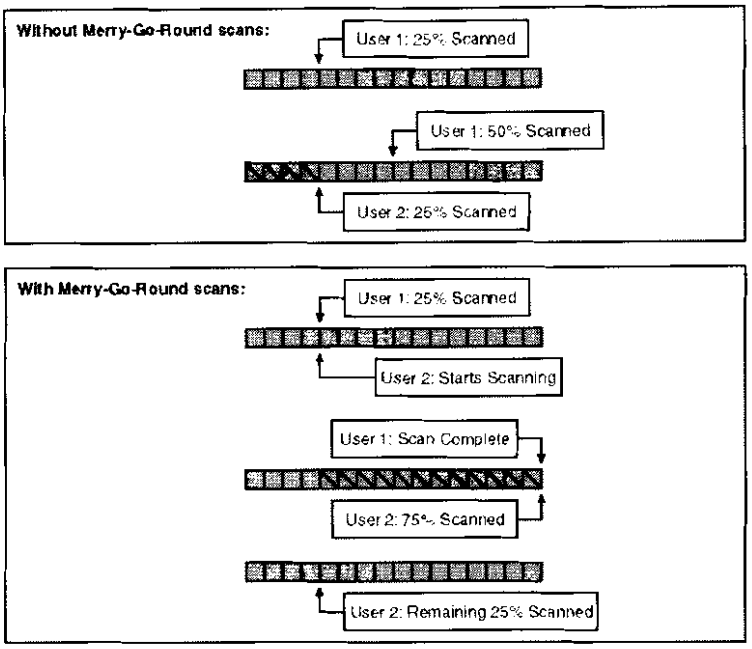
3.2.4.6 Groot geheue-kwessies

Die werkverrigting van SQL Server is deur Microsoft getoets met 'n masjien wat 2 gigagrepe geheue besit het. Groot geheue beteken dat daar minder skyf-lees- of -skryf-aksies plaasvind, wat voordelig is vir optimering. Die hoeveelheid geheue is normaalweg net 'n gedeelte van die databasis se totale grootte. Die geval waar 2 gigagrepe geheue en slegs 1 gigagreep databasisgrootte beskikbaar is, toon duidelik dat die groter geheue onnodig is. Groter geheue lewer dus soms net 'n effense verhoging in die algehele produktiwiteit van die databasis. Die standaard 32 bis-adressering ondersteun slegs 4 gigagrepe fisiese geheue. SQL Server bevat twee geheue-optimeringstegnieke, naamlik vooruitlees ("read-ahead") en rondomtalielees ("Merry-Go-Round Scans"):

- Vooruitlees ("Read-ahead") – Die nodigdheid van data- en indeksbladsye word vooraf geantisipeer en die betrokke bladsye word dan in die bufferarea geplaas. Die tegniek geskied outomaties met geen veranderings wat benodig word nie. Daar bestaan twee tipes vir hierdie tegniek, naamlik een vir die lees van tabelle op "heaps" en een vir indeksreekse. Die lees-van-die-tabelle-tipe gebruik die tabel se allokasiestruktuur om die tabel in skyf-orde te lees. SQL Server lees genoeg data vooruit om 1% van die bufferarea te vul. Vir die indeksreeks word die eerste vlak van die indeksstruktuur gebruik om te bepaal watter bladsye eerste gelees word. Gestel die navraag het 'n 'WHERE state = 'WA'"-gedeelte. Die tipe soek die indeks vir *key* = 'WA' en bepaal dan volgens die eerste vlak se nodi na hoeveel bladsye gekyk moet word om die lees-aksie te bevredig.

- Rondomtalielees ("Merry-Go-Round Scans") – Figuur 3.2 toon 'n voorbeeld van hierdie tegniek.

Figuur 3.2 – Rondomtalielees-geheue-optimeringstegniek (Delaney, 2001:100)



Die tegniek fokus op ongeordende leeswerk. Indien hierdie tegniek nie gebruik word nie, kan 'n proses 'n tabel begin lees en wanneer 'n ander proses dieselfde data wil lees, mag die bladsye alreeds uit die bufferarea wees en moet die bladsye weer vanaf die skyf gelees word. Die tegniek laat die tweede proses toe om by die punt waar die eerste proses tans is, te begin lees. Beide prosesse lees en gebruik dieselfde data. Wanneer die eerste proses afgehandel word, kan die tweede proses die eerste gedeelte weer lees.

3.2.5 Databasis- en lêerstruktuur

Die gedeelte word aan die hand van Delaney (2001:171-315) beskryf. 'n Belangrike onderskeid moet tussen die term databasis en die SQL Server-stelsel gemaak word. Die databasis is slegs een van die onderafdelings van die stelsel. Teoreties kan daar 32 767 databasisse geskep word, maar prakties sal die getal onrealisties wees. Die databasis word opgebou uit twee of meer bedryfstelsellêers wat tydens die skep van die databasis of enige verandering van die databasis geskep word.

'n Paar stelseldatabasisse word outomaties geskep met elke nuwe SQL Server-installasie:

- Meesterdatabasis – Die stelseltabelle in die kritiese databasis bevat inligting oor die bedienerinstallasie en al die ander databasisse wat geskep word. Elke databasis bevat

stelseltabelle wat inligting oor die databasis se objekte hou. Die meesterdatabasis se stelseltabelle bevat inligting oor die skyfspasie, die gebruik van die databasis, aantekeninligting, stelselwye parameters en inligting oor die bestaan van ander SQL-bedieners.

- Modeldatabasis – As 'n nuwe databasis geskep word, word die modeldatabasis as basis vir die nuwe databasis gebruik.
- Tydelike databasis ("tempdb") – Die databasis is 'n werkarea waar tydelike tabelle deur gebruikers geskep mag word en werkstabelle wat resultate bevat, tydelik gestoor kan word. Die databasis word elke keer van nuuts af geskep met die aansit van SQL Server. Daar word tred gehou met die transaksies in die databasis sodat die transaksies wel teruggerol kan word. Daar is egter onvoldoende inligting vir 'n volledige herstel van die transaksie.
- "pubs"-databasis – Die databasis word in voorbeelde in die SQL Server-dokumentasie gebruik en is nie noodsaaklik nie.
- "Northwind"-databasis – Die voorbeelddatabasis is oorspronklik geskep vir Microsoft Access. Dokumentasie wat te make het met die API ("Application programming interfaces") maak dikwels van die databasis gebruik. Die databasis is nie noodsaaklik nie.
- "msdb"-databasis – Die databasis word gebruik deur die SQL Server-agentdiens wat geskeduleerde take soos rugsteun en replikasie hanteer.

Elke databasis het drie tipes lêers vir berging beskikbaar:

- Primêre datalêers – Die lêer hou tred met die res van die datalêers en stoor die data. Die lêer se uitbreiding is normaalweg "MDF".
- Sekondêre datalêers – Die databasis kan nul of meer sekondêre datalêers met uitbreiding "NDF" besit.
- Loglêers – Die lêers bevat die inligting wat benodig word vir herstel en terugrol van transaksies met uitbreiding "LDF".

Die lêers het vyf attribute, naamlik 'n logiese lêernaam, 'n fisiese lêernaam, 'n oorspronklike grootte, 'n maksimum grootte en 'n groeifaktor. Hierdie inligting word in die *sysfiles*-tabel vir elke databasis gestoor.

Die databasislêers kan in lêergroepe saamgegroepeer word. In sommige gevalle kan die werkverrigting verbeter word deur die posisionering van die data en indekse in spesifieke lêergroepe te beheer. Die groep wat die primêre datalêer bevat, is die primêre lêergroep. Daar bestaan slegs een primêre lêergroep en indien nie anders gespesifiseer nie, word al die datalêers in die groep geplaas. Die lêergroepe word gebruik wanneer verskeie tabelle na verskillende skywe geskryf moet word.

Die databasis word opgedeel in logiese bladsye (8 kilogrepe elk) en die bladsye word oor al die lêers vanaf 0 tot by *x* (wat afhanklik is van die grootte van al die lêers) genommer. Spasie word geallokeer deur eenhede wat spasie-aanpassings ("extents") genoem word. 'n Enkele spasie-aanpassing bestaan uit 8 logiese bladsye (of 64 kilogrepe spasie). Die twee tipes spasie-aanpassings sluit 'n uniforme spasie-aanpassing in, wat uniform uit een objek se bladsye bestaan, en 'n gemengdespasie-aanpassing, wat uit soveel as agt verskillende objekte se bladsye kan bestaan. Inligting oor hierdie spasie-aanpassings word in spesiale bladsye gestoor:

- "Global Allocation Map (GAM)"-bladsye – Die bladsye toon watter spasie-aanpassing in gebruik is. Gewoonlik is daar een GAM-bladsy met grootte van 4 gigagrepe in 'n lêer.
- "Shared Global Allocation Map (SGAM)"-bladsye – Dui aan watter spasie-aanpassings tans gebruik word as gemengdespasie-aanpassings en ten minste een oop bladsy besit.
- "Index Allocation Map (IAM)"-bladsye – Karteer die spasie-aanpassings in 'n databasislêer wat deur 'n "heap" (tabel sonder 'n gebondelde indeks) of indeks gebruik word.
- "Page Free Space (PFS)"-bladsye – Dui aan of individuele bladsye geallokeer is en die hoeveelheid spasie wat op elke bladsy skoon is.
- "Differential Changed Map (DCM)"-bladsye – Die bladsye hou tred met watter spasie-aanpassings verander het sedert die vorige volle databasisrugsteun.
- "Bulk Changed Map (BCM)"-bladsye – Die bladsye word gebruik wanneer 'n spasie-aanpassing in die lêer gebruik word in 'n minimale of 'n "bulk-logged"-bewerking (spesiale rugsteunmodel).

3.2.5.1 Tabelle

Die fundamentele datastrukture in die databasis is tabelle. Tabelle is 'n versameling van data met betrekking tot 'n spesifieke entiteit (persoon, plek of objek) wat 'n eindige aantal beskrywende attribute bevat. Die attribute word voorgestel in kolomme en elke element in die tabel word voorgestel deur rye. Gewoonlik het elke ry 'n unieke sleutel wat die ry uniek identifiseer, genoem die primêre sleutel, maar dit is nie noodsaaklik nie. Die meeste tabelle is verwant aan ander tabelle in die databasis en die verwantskap kan geforseer word deur vreemde sleutels ("foreign keys") of die toepassing wat die databasis gebruik, moet die verwantskap toepas. Die basiese datatipes is numeries, karakter, datum en tyd, groot objekte ("Large Object" (LOB)) of 'n paar ander spesiale tipes, soos byvoorbeeld die "sql_variant"-tipe wat enige data kan bevat behalwe "LOB", ry-weergawe ("timestamp") en enige tipe wat nie vir 'n kolom gedefinieer kan word nie (byvoorbeeld "cursor"). Die gebruiker mag self ook tipes byvoeg tot die databasis.

Elke kolom mag 'n waarde "NULL" aanneem indien die kolom so gedefinieer is. SQL Server bevat 'n spesiale "bitmap" ('n bis ("bit") mag 'n 0 of 1 bevat) in elke data-ry wat aandui watter kolomme "NULL"-waardes bevat. Dit beteken dat SQL Server vir elke ry eers die "bitmap" moet verwerk wanneer toegang tot die ry versoek word. Die verstekwaarde vir die deklarasie van 'n kolom is dat "NULL"-waardes toegelaat word.

'n Kolom mag as 'n identiteitskolom geklassifiseer word deur die identiteitsattribuut van die kolom te stel. Die kolom bevat dan unieke waardes wat op mekaar volg en by 'n voorafvasgestelde waarde begin. Die waardes word outomaties by elke nuwe rekord bygevoeg. Die probleem met hierdie kolom is dat die volgorde deur verwyderbewerkings onderbreek kan word. Die verskillende datatipes en die struktuur van 'n data-ry wat in 'n databladsy gestoor word, word in die volgende twee paragrawe bespreek.

3.2.5.2 Datatipes

SQL Server bied verskeie datatipes waarvan sommiges tot 'n vaste lengte beperk is en sommiges 'n dinamiese lengte kan aanneem met 'n sekere maksimum wat gespesifiseer moet word. Die datatipes wat in die eksperimentele gedeelte van die verhandeling gebruik word, word volgens Delaney (2001:226-235) beskryf. Die ander datatipes se beskrywing is in SQL Server se dokumentasie en die gedeelte van Delaney (2001:226-235) bekombaar. Die datatipes word in tabel 3.1 beskryf, wat die datatipe, die grootte van die datatipe in grepe en 'n kort beskrywing van die datatipe aandui.

Tabel 3.1 – Sekere datatipes van SQL Server

Datatipe	Grootte (Grepe)	Beskrywing
varchar(*n*)	*L*	Karakterstring met dinamiese lengte *L* en maksimum *n*.
char(*n*)	*N*	Karakterstring met 'n vaste lengte van *n* karakters (of *n* grepe).
datetime	8	'n Datum en tyd word in hierdie datatipe gestoor.
bigint	8	Groot heelgetal van (-2 tot die mag 63) tot by (2 tot die mag 63) - 1
int	4	Heelgetal van (-2 tot die mag 31) tot by (2 tot die mag 31) – 1
float	8	Desimale getal van -1.79 * (10 tot die mag 308) tot by 1.79 * (10 tot die mag 308)

3.2.5.3 Die struktuur van 'n data-ry

Tabelle se data word gestoor in databladsye met grootte van 8 kilogrepe, wat uit drie dele bestaan, naamlik die bladsy-opskrif ("header"), die data-rye en die ry se oorsprongskikking ("row offset array").

Die opskrifgedeelte ("header"), met 'n grootte van 96 grepe, bevat inligting oor byvoorbeeld die lêer- en bladsynommer van die bladsy in die databasis en die id van die eienaar-objek van die bladsy. Die data-rye bevat die data, en die ry se oorsprongskikking dui die oorsprong en die logiese orde van al die data-rye op elke bladsy aan. Die maksimum grootte van 'n enkele data-ry is 8060 grepe. Elke data-ry het 'n 2 greep-inskrywing in die oorsprongskikking. Die aantal data-rye bepaal die grootte van die oorsprongskikking, en die skikking kan 'n invloed hê op die aantal data-rye wat op die databladsy kan pas. Die struktuur van elke data-ry word aan die hand van tabel 3.2 verduidelik (Delaney, 2001:253).

Tabel 3.2 – Die inligting wat in elke data-ry gestoor word (SQL Server)

Inligting	Grootte (grepe)	Beskrywing
Statusbisse A	1	Statusbisse A bevat 'n "bitmap" met die volgende betekenisse: • Bis 0 – Weergawe-inligting, maar in SQL Server 2000 is dit altyd 0. • Bis 1 tot 3 – Die gedeelte word geneem as 'n 3 bis-waarde wat inligting bevat soos byvoorbeeld of die ry 'n primêre rekord besit. • Bis 4 – Dui aan dat die "NULL bitmap" bestaan. • Bis 5 – As die bis gestel is, dui dit daarop dat daar 'n veranderlikelengtekolom in die ry bestaan. • Bis 6 en 7 – Nie van toepassing in SQL Server 2000 nie.
Statusbisse B	1	Die greep word nie in SQL Server 2000 gebruik nie, maar die plek word wel gehou.
Vastelengtegrootte	2	Die grootte van die vastelengtekolomme.
Vastelengtedata	Grootte van vastelengtekolomme	Die vastelengtekolomme se data word hier gestoor.
Aantal kolomme	2	Die totale aantal kolomme in die tabel.
"NULL bitmap"	Rond (aantal kolomme / 8) op.	Elke bis verteenwoordig 'n kolom in die tabel en dui aan of die kolom se waarde "NULL" is (met 1) of nie (met 0). Die "bitmap" is altyd teenwoordig selfs al is daar geen "NULL"-kolomme nie.
Aantal veranderlikelengtekolomme	2	Die aantal veranderlikelengtekolomme wat die tabel besit.
Veranderlike kolom eindpuntskikking	2 * aantal veranderlikelengtekolomme	Die skikking dui elke veranderlikelengtekolom se eindpunt aan.
Veranderlikelengtedata	Grootte van veranderlikelengtekolomme	Die veranderlikelengtedata word hier gestoor.

Die "LOB"-data ("Large binary object"-data) word weens die data se grootte nie altyd in die data-ry gestoor nie, alhoewel dit 'n verbetering in werkverrigting veroorsaak indien die data wel daar gestoor kan word. 'n Wyser na die data word gewoonlik in die data-ry gestoor. Die data word dan in 8 kilogreep-bladsye gestoor volgens 'n B-boomstruktuur.

Indekse se ry-strukture is baie dieselfde as 'n normale data-ry (Delaney, 2001:415-427). Die verskille is dat daar geen statusbisse B en vastelengtegrootte grepe bestaan nie. Die

"pminlen"-waarde wat in die bladsy-opskrif is, word gebruik om die eindpunt van die vastelengtekolomme aan te toon. As geen van die kolomme "NULL"-waardes kan aanneem nie, word die aantal kolomme en die "NULL bitmap"-elemente uit die data-ry uitgelaat. Die data-inhoud van die indeks-ry is afhanklik van die vlak van die indeks. Alle rye behalwe die rye van die blaarvlak bevat 'n 6 greep- (2 grepe vir lêernommer en 4 grepe vir bladsynommer) afwaartsebladsywyser ("down-page"-wyser) saam met die sleutelwaardes en boekmerke (verwys na waar die data gevind kan word ("rid")). Die afwaartsebladsywyser is die laaste kolom in die vastelengtedata-gedeelte van die ry en wys na die volgende bladsy in die boomstruktuur. 'n Gebondelde indeks se blaarbladsye het die struktuur van 'n data-ry.

3.2.6 Algemene inligting oor die indekse van SQL Server

Delaney (2001:403-448) se gedeelte oor indekse word vervolgens beskryf om na die praktiese implementering van indekse in SQL Server te kyk.

3.2.6.1 Inleiding

SQL Server maak van B-boomindekse (sien paragraaf 2.4.1.1.2) gebruik. Die gebondelde indeks se blaarnodusse bevat die werklike datarekords (alternatief 1), terwyl die ontbondelde indekse 'n wyser na die datarekords bevat (alternatief 2 of 3). Die gebondelde indeks stoor die data-elemente in 'n tabel wat volgens die soeksleutelwaarde gesorteer is. Die databladsye word in 'n dubbelgeskakelde lys (elke elemente in die lys bevat wysers na beide die volgende en voorafgaande elemente) gehou, genaamd die bladsyketting. Die orde van bladsye in die bladsyketting en die orde van die rye in die databladsye is afhanklik van die orde van die indekssleutels. Die navraagoptimeerder verkies 'n gebondelde indeks omdat die data-elemente in die blaarnodusse beskikbaar is. Reekssoektogte is vinnig omdat die navraagoptimeerder vasstel dat slegs 'n sekere aantal databladsye deursoek moet word. Gebondelde indekse word nie fisies in volgorde gestoor nie, maar die bladsyketting is wel in volgorde. Nuwe elemente kan dus maklik bygevoeg word deur slegs die skakels te verander. Elke gebondelde indeks is uniek en SQL Server voeg 'n 4 greep-"uniqueifier"-veld toe tot elke ry wat moontlik 'n duplikaat mag wees.

Gestel daar bestaan 'n gebondelde indeks en ook 'n ontbondelde indeks. Die wysers van die ontbondelde indeks is dan die gebondelde indekssleutel vir die data-ry. Indien daar slegs ontbondelde indekse bestaan, is die tabel 'n "heap" en die wyser is 'n RID (ry-id met formaatlêer, bladsy en posisie ("slot")). Elke tabel mag in die omgewing van 249 ontbondelde indekse bevat.

3.2.6.2 Skep van indekse (SQL Server)

Die sintaksis en verduideliking vir die skep van 'n indeks word bespreek (Delaney, 2001:408):

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX *index_name*
ON *table_name* (*column_name* [ASC | DESC][,...n])
[WITH
[FILLFACTOR = *fillfactor*]
[[,] [PAD_INDEX]
[[,] IGNORE_DUP_KEY]
[[,] DROP_EXISTING]
[[,] STATISTICS_NORECOMPUTE]
[[,] SORT_IN_TEMPDB]
]

Die tabelnaam, kolomme en of elke kolom stygend ("ASC") (wat die verstekwaarde is) of dalend ("DESC") gesorteer moet word, word gespesifiseer.

- "UNIQUE" – Die sleutelwoord verseker dat duplikate nie voorkom nie.
- Gebondel ("CLUSTERED") of ontbondel ("NONCLUSTERED") – Die tipe van die indeks word hierdeur gespesifiseer. Ontbondelde indekse is die verstekwaarde.
- "index_name" – Die indeks moet 'n naam kry.
- "table_name" – Die naam van die tabel moet gespesifiseer word.
- "column_name" – Elke kolom waarop die indeks van toepassing is, moet saam met die wyse van sortering, naamlik stygend of dalend, gelys word.
- "FILLFACTOR = *fillfactor*" – Die attribuut spesifiseer die hoeveelheid spasie wat vir elke blaarnodus van die indeks gereserveer moet word. Aangesien gebondelde indekse se data-elemente in die blaarnodusse gestoor word, word hierdie attribuut gebruik om te bepaal hoeveel spasie vir die tabel oopgelos kan word. Die spasie word egter nie op hierdie vlak gehou nie, maar die attribuut dui net op 'n aanvanklike allokering. Die "DBCC DBREINDEX"-bevel sal die indeks oorbou om die oorspronklike "FILLFACTOR" te akkomodeer. Die verstekwaarde is nul, wat beteken dat die blaarbladsye so vol as moontlik gemaak word.
- "PAD_INDEX" – Die nie-blaarbladsye mag ook spasie reserveer. Die PAD_INDEX-attribuut word vir hierdie doel gebruik en gebruik dieselfde waarde as die "FILLFACTOR".
- "IGNORE_DUP_KEY" – Die attribuut veroorsaak dat duplikate nie die hele transaksie terugrol nie. Die duplikaat word geïgnoreer en die ander rye sal nog steeds bygevoeg en verander word.

- "DROP_EXISTING" – Die attribuut veroorsaak dat die verwydering en herbou van indekse in een transaksie plaasvind. Die attribuut word gebruik tydens die herbou van gebondelde indekse om te voorkom dat ontbondelde indekse twee keer herbou moet word (een keer vir die verwyder en een keer vir die herbou).
- "STATISTICS_NORECOMPUTE" – Die statistiese inligting op kolomme word gebruik deur die navraagoptimeerder om die beste verwerkingstegniek te ontwikkel vir 'n navraag. Hierdie attribuut veroorsaak dat die statistiese inligting nie outomaties verwerk word nie. Die statistiese inligting word net eenmalig met die skep van die indeks verwerk. Die statistiese inligting word in paragraaf 3.2.7 ondersoek wat handel oor die navraagoptimeerder (verwerker).
- "SORT_IN_TEMPDB" – Die attribuut bepaal waar die sortering van die sleutelwaarde plaasvind. SQL Server gebruik standaard die spasie van die lêergroep waar die indeks geskep word. Die attribuut sal die werkverrigting verhoog, veral as die "tempdb"-databasis op 'n ander hardeskyf is as die hardeskyf met die gewone databasis. Twee verskillende lees/skryf-koppe van die hardeskywe word gebruik om die tabelbladsye te lees en die sorteerbuffers te hanteer. 'n Ander metode vereis dat verskillende lêergroepe vir die tabel en indekse geskep moet word. As hierdie twee lêergroepe op verskillende hardeskywe is, sal dit die lees/skryf-kop se werk verminder.

Die gebruik van primêre sleutel en unieke beperkings lei tot 'n unieke indeks wat min of meer dieselfde is as wanneer 'n spesifieke indeks geskep word met die "CREATE INDEX"-bevel. Die grootste verskil tussen die twee metodes is dat die beperkings se indekse slegs met 'n "ALTER TABLE"-bevel verwyder kan word, terwyl die gewone indekse met "DROP INDEX"-bevele verwyder kan word. Vreemdesleutelbeperkings ("foreign key constraints") wat op die primêre sleutel of unieke beperking toegepas is, moet eers verwyder word voor die indeks vir die beperkings kan verwyder.

3.2.6.3 Die struktuur van indeksbladsye

Die struktuur van indeksbladsye is soortgelyk aan databladsye met 'n vaste grootte van 8 kilogrepe. Die indeksbladsye bestaan uit 'n 96 greep-opskrif ("header") met 'n oorsprong-skikking ("offset array") aan die einde wat twee grepe vir elke ry beslaan om die oorsprong vir elke ry te stoor. Die opskrifinligting stem baie ooreen met databladsye se inligting met die verskil van die tipe wat 1 vir data en 2 vir indekse is. Die opskrif van indeksbladsye bevat nie-nul-waardes vir die vlak- en indeks-id-kolomme waar databladsye nul is. Die drie tipes indeksbladsye is: die blaarnodusse vir ontbondelde indekse, nie-blaarnodusse vir gebondelde indekse en die nodus-vlak vir ontbondelde indekse.

3.2.6.4 Fragmentasie van indekse

SQL Server onderhou indekse outomaties, maar die probleem ontstaan dat indekse fragmenteer. Twee tipes fragmentasie kom voor, naamlik interne en eksterne fragmentasie:

- Interne fragmentasie – Die indeks neem te veel spasie op. Dit beïnvloed die leesbewerkings, aangesien meer bladsye gelees word as wat nodig is. Hierdie geval is egter nie krities nie, aangesien verdeling van bladsye 'n duur bewerking is en meer spasie op 'n bladsy voordelig kan wees.
- Eksterne fragmentasie – Hierdie fragmentasie geskied wanneer die logiese orde van die bladsye nie ooreenstem met die fisiese orde nie of wanneer die spasie-aanpassings van 'n tabel nie opeenvolgend is nie. Die fragmentasie is nie 'n probleem indien slegs enkele rye verkry moet word nie, aangesien SQL Server die rye maklik kan vind. 'n Gesorteerde soektog van 'n gedeelte of die hele tabel of indeks kan egter deur fragmentasie bemoeilik word, aangesien die logiese bladsyketting gevolg moet word.

SQL Server bevat 'n bevel "DBCC SHOWCONTIG" wat die fragmentasie van die indeks kan ondersoek deur die bladsyketting van elke blaarnodusse deur te gaan. Die moontlike resultaatwaardes van die "DBCC SHOWCONTIG"-bevel word in die boek deur Delaney (2001:432-434) beskryf.

Die fragmentasie van indekse kan verwyder word deur die indeks te herbou of om 'n sogenaamde "defragmenteer"-aksie ("defragment"-aksie) op die indeks toe te pas. Die herbou van indekse is nadelig aangesien die tabel nie beskikbaar is tydens die proses nie. Ontbondelde indekse vereis 'n gedeelde slot op die tabel tydens die herbouproses, wat beteken dat slegs seleksienavrae kan plaasvind, maar hierdie navrae geskied sonder indeksvoordele. Gebondelde indekse besit 'n eksklusiewe slot op die tabel, wat geen toegang toelaat tot die tabel nie. Die defragmenteer proses kan die fragmentasie verwyder sonder om die indeks te herbou. Die "DBCC INDEXDEFRAG"-bevel hersorteer die blaarvlakbladsye in logiese en fisiese orde deur slegs na die bladsye te kyk wat alreeds aan die blaarvlak geallokeer is. Die bladsye word in-plek gesorteer met 'n algoritme wat soortgelyk aan borrelsorteer is. Die proses verminder die fragmentasie na tussen 0 en 2%. Die proses streef om die oorspronklike "FILLFACTOR"-waarde te bereik.

3.2.6.5 Spesiale indekse van SQL Server

Twee spesiale indekse kan geskep word, naamlik indekse op berekende kolomme en indekse op skyntabelle. Berekende kolomme en skyntabelle is logiese strukture met geen fisiese stoor

van data nie. As indekse op hierdie logiese strukture geskep word, word die data wel fisies in die indeksstruktuur gestoor. Verskeie voorvereistes is nodig vir die skep van hierdie spesiale indekse. Sekere stelselopsies moet gestel word en die indekse is slegs van toepassing op sekere funksies en tabelle wat aan sekere voorwaardes voldoen. Een van die voorvereistes is byvoorbeeld die waarde "ARITHABORT" wat 'n navraag termineer indien 'n oorvloei- of nuldelingfout ("devide-by-zero error") voorkom (Microsoft Corporation, 2005d). Die onderskeid word gemaak tussen deterministiese funksies en nie-deterministiese funksies. Deterministiese funksies verskaf elke keer dieselfde resultaat met dieselfde invoerwaardes, soos byvoorbeeld 2 * Kolom A, met Kolom A 'n heelgetalwaarde of sommering van 'n kolom. Nie-deterministiese funksies sluit byvoorbeeld 'n funksie in wat die naam van die dag van die maand of weeksdae verskaf. Aangesien die funksie afhanklik is van die taal vir die betrokke navraag, kan die resultate verskil vir dieselfde invoer.

SQL Server verskaf 'n funksie "COLUMNPROPERTY" wat die gebruiker in kennis stel of die spesifieke berekende kolom deterministies is of nie, voor die indeks geskep is. Verdere voorwaardes vir skyntabelle om 'n indeks te verkry, word benodig, soos byvoorbeeld dat die skyntabel se selekteerdefinisie nie die argument "TOP" mag bevat nie (sien paragraaf 3.1 in bylaag A). Die funksie "OBJECTPROPERTY" verifieer dat die skyntabel wel 'n indeks mag verkry.

3.2.6.6 Hantering van databywerking in indekse (SQL Server)

Hierdie gedeelte word beskryf volgens die boek van Delaney (2001:476-497). In SQL Server het alle tabelle normaalweg 'n gebondelde indeks, alhoewel uitsonderlike gevalle bestaan waar 'n tabel nie so 'n indeks bevat nie. Die byvoeg-, verwyder- en wysigbewerkings en 'n paar ander aspekte word in die volgende paragrawe bespreek.

3.2.6.6.1 Byvoegbewerking

SQL Server moet elke keer bepaal waar die nuwe ry (ten opsigte van die teorie (sien paragraaf 2.4) is dit die data-element) geplaas moet word. In die geval waar die tabel 'n "heap" (geen gebondelde indeks) is, word die ry in enige bladsy met spasie geplaas. IAM- en PFS-bladsye (sien paragraaf 3.2.5) word gebruik om te bepaal watter "extents" in 'n lêer alreeds aan tabelle behoort en watter bladsye in die "extents" spasie oor het.

Elke ry het 'n spesifieke plek met die gebruik van 'n gebondelde indeks. Indien daar nie voldoende spasie bestaan vir die nuwe ry op die bladsy waar die ry hoort nie, moet 'n bladsy geallokeer word en in die lys van bladsye geskakel word, en verdeling van bladsye moet

plaasvind. Indien die "extent" nie spasie oor het vir 'n nuwe bladsy nie, moet 'n nuwe "extent" (agt bladsye of 64 kilogrepe) aan die tabel geallokeer word. Tydens bladsyverdeling bly die helfte van die rye in die oorspronklike bladsy oor en die helfte beweeg oor na die nuwe bladsy. Dit is moontlik dat die verdeling nog steeds te min spasie bied vir die nuwe ry as gevolg van veranderlikelengtekolomme. Na die verdeling word een of twee rye na die ouerbladsy gepromoveer.

Die indeksboom word altyd van bo by die wortel af na onder deursoek. Elke ouernodus (nie-blaarnodus) is gesluit vir verandering totdat die kindnodus beskikbaar is vir sluiting, waarna die ouernodus se slot verwyder word. As die ouerbladsy deursoek word om 'n ry by te voeg by die indeks, bepaal SQL Server eers (voordat die slot verwyder word) of die bladsy nog twee rye sal kan huisves. Indien nie, word die bladsy verdeel om te probeer verseker dat die ouerbladsy altyd spasie het vir 'n ry of rye wat afkomstig is van die kindbladsye se verdeelproses. SQL Server bevat drie tipes verdelings vir indekse, naamlik wortelbladsyverdeling, nie-blaarbladsyverdeling (nie wortel nie) en databladsyverdeling.

- Wortelbladsyverdeling – Twee nuwe bladsye word tot die indeks geallokeer. Al die rye van die wortel word tussen die twee bladsye verdeel en die nuwe indeks-ry word in een van hierdie nuwe bladsy geplaas. Die oorspronklike wortelbladsy is steeds die wortel met slegs twee rye wat na die twee nuwe bladsye verwys. 'n Nuwe vlak is in die indeksboom geskep, en volgens Delaney (2001:475) gebeur dit nie gereeld nie. Die teorie, soos beskryf in paragraaf 2.4.1.1.2 se figuur 2.5, verskil in dié opsig dat een van die nuwe bladsye wat geskep is, die wortel raak. Verdeling vind plaas tussen die ou wortelbladsy en 'n nuwe bladsy, en 'n nuwe wortelbladsy word dan geskep.
- Nie-blaarbladsyverdeling – Verkry die middelpunt van die indekssleutels op die bladsy, allokeer 'n nuwe bladsy en kopieer die onderste helfte van die ou indeksbladsy na die nuwe bladsy.
- Databladsyverdeling – Die verdeling geskied net by gebondelde indekse. Die middelpunt van die indekssleutels op die databladsy word verkry. 'n Nuwe bladsy word geallokeer en die helfte van die ou bladsy word oorgedra na die nuwe bladsy. Hierdie proses benodig die indeksbestuurder om die bladsy te identifiseer waar die nuwe ry moet gaan en om groot rye wat nie op die ou of nuwe bladsy pas nie, te hanteer. Normaalweg plaas SQL Server die nuwe ry in die nuwe bladsy.

Die bladsyverdelings is nie te duur nie, alhoewel die frekwensie van die verdelings tot 'n minimum beperk moet word tydens piektye (baie gebruikers). Die "FILLFACTOR"-opsie word normaalweg gebruik om die frekwensie te reguleer. Herskepping van die indeks gedurende stil tye is normaalweg 'n goeie idee. Indien geen stil tye bestaan nie, kan die "DBCC

INDEXDEFRAG"-bevel gebruik word. Die SQL Server-agent kan gebruik word om hierdie prosesse te skeduleer.

3.2.6.6.2 Verwyderbewerking

SQL Server verminder nie outomaties spasie wanneer 'n ry uit 'n "heap"-tabel verwyder word nie. Die vermindering geskied eers wanneer 'n bladsy bykomende spasie benodig vir die byvoeg van 'n nuwe ry.

'n Verwydering in 'n B-boomindeks se blaarnodusse geskied deur die betrokke ry te merk as 'n spookrekord ("ghost record"). Die ry bly op die bladsy met een bis wat verander in die ry se opskrif ("header"), wat aandui dat dit 'n spookrekord is. Die aantal spookrekords word in die bladsy-opskrif gestoor. Die spookrekords word gebruik vir konkurrente optimeringstegnieke tydens sluit van sleutelreekse. As spookrekords nie gebruik word nie, sal die hele reeks rondom 'n afgeleë sleutel gesluit moet word. Gestel daar bestaan 'n unieke indeks op 'n kolom wat heelgetalle met die waardes 1, 30 en 100 bevat: As 30 verwyder word, sal SQL Server die hele reeks tussen 1 en 100 moet sluit om byvoeging te voorkom.

Spookrekords veroorsaak dat 30 steeds gebruik kan word as die eindpunt van 'n sleutelreeks-slot[13], wat toelaat dat ander waardes buiten 30 bygevoeg kan word. 'n Spesiale opruiming "thread" deursoek B-boomindekse vir spookrekords en verwyder hulle op verskillende stadiums ("asynchronous") vanaf die blaarnodusse van die indeks. Die "thread" hanteer die outomatiese verkleining van die databasis (indien die opsie gestel is). Die wysers van nie-blaarnodusse moet altyd tydens verwydering onderhou word, aangesien sommige van die wysers na nodusse wys wat nie meer bestaan nie. Rye in die indeksbladsye raak nie spookrekords nie en word hanteer soos "heap"-tabelle ten opsigte van die spasievermindering.

As die laaste ry uit 'n databladsy verwyder is, sal die bladsy verwyder word (tensy dit die laaste bladsy in die tabel is). Die ry wat na die databladsy verwys het, word uit die indeksbladsy verwyder. Indeksbladsye word verwyder as daar slegs een ry in die bladsy oor is. Die ry word verskuif na verwante bladsye, en die ou indeksbladsy word verwyder. Die metode breek weg van die teorie (sien paragraaf 2.4.1.2.2) in dié opsig dat die teorie slegs die indeksbladsy verwyder as die bladsye heeltemal leeg is.

[13] Die slot sluit 'n reeks van sleutels tussen twee punte.

3.2.6.6.3 Wysigbewerking

SQL Server het verskeie maniere om rye te wysig. Die metode word gekies deur te kyk na die aantal rye wat geaffekteer word, hoe toegang na die rye sal geskied en of die indekssleutelwaarde gaan verander. Wysigings kan in-plek (verstekmetode) of met 'n verwyder- en daaropvolgende byvoegbewerking geskied. Die wysigbewerking kan óf deur die navraagverwerker óf die stoorenjin hanteer word.

Die situasie kan ontstaan dat 'n ry na 'n nuwe posisie geskuif moet word. Dit gebeur byvoorbeeld wanneer 'n ry se veranderlikelengtekolom vergroot en die ry nie meer op die bladsy pas nie. Elke keer wat veranderings aan die gebondelde indeks se sleutel (of, in hierdie konteks, ry-opspoorder) aangebring word, sal al die ontbondelde indekse ook verander moet word. Die keuse van die gebondelde indekssleutel is dus belangrik. In die geval van 'n "heap"-tabel is die ry-opspoorder die fisiese posisie van die ry, en met die verskuiwing van 'n ry na 'n nuwe bladsy, sal 'n aanstuurwyser ("forwarding pointer") gelaat word in die oorspronklike posisie, en die ander ontbondelde indekse benodig geen verandering nie, aangesien die indekse na die oorspronklike posisie wys en van daar verwys word deur die aanstuurwyser ("forwarding pointer") na die nuwe posisie.

Die aanstuurwysers vergemaklik bywerkings in "heap"-tabelle. Indien die ry weer moet skuif, word die aanstuurwyser net gewysig na die nuwe posisie. As die aangestuurde ry voldoende verklein het sodat die ry weer in die oorspronklike posisie kan kom, word die aanstuurwyser verwyder.

SQL Server 2000 bied twee strategieë om indekse te onderhou, naamlik tabelvlakbywerkings en indeksvlakbywerkings. Die navraagoptimeerder kies tussen hierdie twee strategieë volgens die verwagte uitvoerkoste vir elke strategie. Die tabelvlakbywerking hanteer elke ry se indekse soos wat die ry gewysig word. Dit lei tot 'n groot aantal willekeurige indekssoektogte, aangesien toegang tot elke indeks vir elke bywerking en vir elke ry moet geskied. As die bywerkstroom nie gesorteer is nie, sal SQL Server baie soektogte op elke indeks moet toepas. Die indeksvlakbywerkings lei tot 'n sortering van die rye volgens elke indeks. Die aantal sorteerbewerkings stem ooreen met die aantal indekse vir die ry. Vir elke indeks word die bywerkings aangebring en geen indeksbladsy word ooit meer as een keer gebruik nie. As die bywerking klein is en die tabel en indekse se grootte is aanpasbaar, sal die navraagoptimeerder normaalweg die tabelvlakbywerking kies. Groot bywerkings sal eerder van indeksvlak gebruik maak.

As voorbeeld, neem 'n "heap"-tabel waar nuwe rye aan die einde bygevoeg word. Gestel twee ontbondelde indekse kom op kolom *a* en *b* voor. Die invoerstroom is gesorteer op kolom *a* en saamgevoeg in die indeks op *a*. As die indekselement 20 grepe is, sal 'n 8 kilogreep-bladsy wat 70% vol is ongeveer 280 elemente bevat (8 kilogrepe * 70% / 20 grepe). Agt bladsye (een "extent") bevat 2240 elemente as die aanname geld dat die blaarbladsye eweredig in die "extents" versprei is. As die tabel met 1% groei, beteken dit dat gemiddeld 2.8 nuwe elemente op die bladsy bygevoeg is of dat 22.4 nuwe elemente per "extent" bygekom het. Tabelvlakbyvoeging sal elke bladsy gemiddeld 2.8 keer lees, wysig en skryf (dus 5.6 lees- of skryf-aksies per bladsy of 44.8 lees- of skryf-aksies per "extent") tensy die bufferarea groot is. Indeksvlakbyvoeging sal elke bladsy een keer lees, wysig en skryf (ook "extent" met eweredige verspreiding). Die beste geval sal dus 2 lees- of skryf-aksies per "extent" tot gevolg hê. Die koste sluit egter die koste van die sortering van elke indeks in.

Alle wysigbewerkings benodig 'n vorm van eksklusiewe sluiting van die data (alleenreg op die data). Verskeie slotte kom voor, naamlik 'n ry-slot (hou die spesifieke ry vas en is die verstekslot), bladsyslotte (hou bladsy vas) en tabelslotte (hou tabel vas). Eksklusiewe slotte word altyd behou vir die hele transaksie sodat terugrolbewerkings kan plaasvind. Gedurende 'n volledige tabeldeursoek moet elke ry deursoek word om die ry te kry wat moet verander. Die rye is dan geblok vir alle ander prosesse wat die rye wil verander.

3.2.7 SQL Server se navraagverwerker ("Optimizer")

Hierdie gedeelte word met die hulp van Delaney (2001:815-865) beskryf. SQL Server se navraagverwerker is nie 'n enkele komponent nie, maar eerder verskeie verwante komponente. Die belangrikste deel van die verwerker is die navraagoptimeerder. Die optimeerder verskaf 'n uitvoerplan (bevat stappe om die navraag uit te voer, besluit watter indekse om te gebruik en bied strategieë vir verbindingsprosesse ("joins"), groeperingsprosesse ("GROUP BY") en sorteringsprosesse) vir die navraag. Die navraagverwerking word meestal deur die SQL-bestuurder (sien figuur 3.1) beheer.

Vertaling ("compilation") en uitvoering is twee verskillende fases en die tyd tussen hierdie fases kan 'n paar millisekondes wees of selfs 'n paar dae. Die vertalingsfase sluit optimering in. Navrae wat gebruik maak van tyd- of gebruikerafhanklike data kan verskil vir die vertaling- en uitvoerfase en moet eers goed oorweeg word. Gedurende die vertalingsfase word Transact-SQL-kode deur 'n spesiale enjin verwerk wat hierdie prosedure-strukture (byvoorbeeld "if" en "while") kan hanteer. Die normale SQL-uitdrukkings word deur die navraagoptimeerder verwerk. Die verskillende aspekte van die optimeringproses word in die volgende paragrawe bespreek.

Volgens Hellerstein (1998:131) is dit moeilik om die effektiwiteit van 'n navraagoptimeerder te bepaal. Verskeie klein, willekeurige en standaard werkverrigtingstoetse kan ondersoek word om sodoende die effektiwiteit te meet. Navrae wat die optimeerder dwing om een optimeringstegniek bo 'n ander te kies, kan ook gebruik word (Hellerstein, 1998:132).

3.2.7.1 Vertaling

Elke uitdrukking van die navraag word ontleed ("parsed") en 'n volgorde-reeksboom ("sequence tree") word geskep. Die boom word genormaliseer, wat verbinding tot gevolg het. Die normalisering verifieer dat tabelle en kolomme bestaan en laai die metadata (data oor data) vir die tabelle en kolomme. Inligting oor implisiete omskakelings word tot die volgorde-reeksboom bygevoeg (byvoorbeeld as die navraag 10.0, wat 'n reële getal is, wil byvoeg by 'n heelgetal, word die 10.0 implisiet omgeskakel na die heelgetal 10). Normalisering vervang die verwysing na 'n skyntabel met die definisie van die skyntabel. Normalisering doen 'n paar sintaksisgebaseerde optimerings, soos byvoorbeeld vir INSERT-, UPDATE- of DELETE-bevele (normale SQL-bevele) word inligting van die volgorde-reeksboom oor die navraag verkry en 'n spesiale struktuur, die navraaggrafiek, word vir die optimeerder geskep. Die navraaggrafiek word geoptimeer en 'n uitvoerplan word verkry.

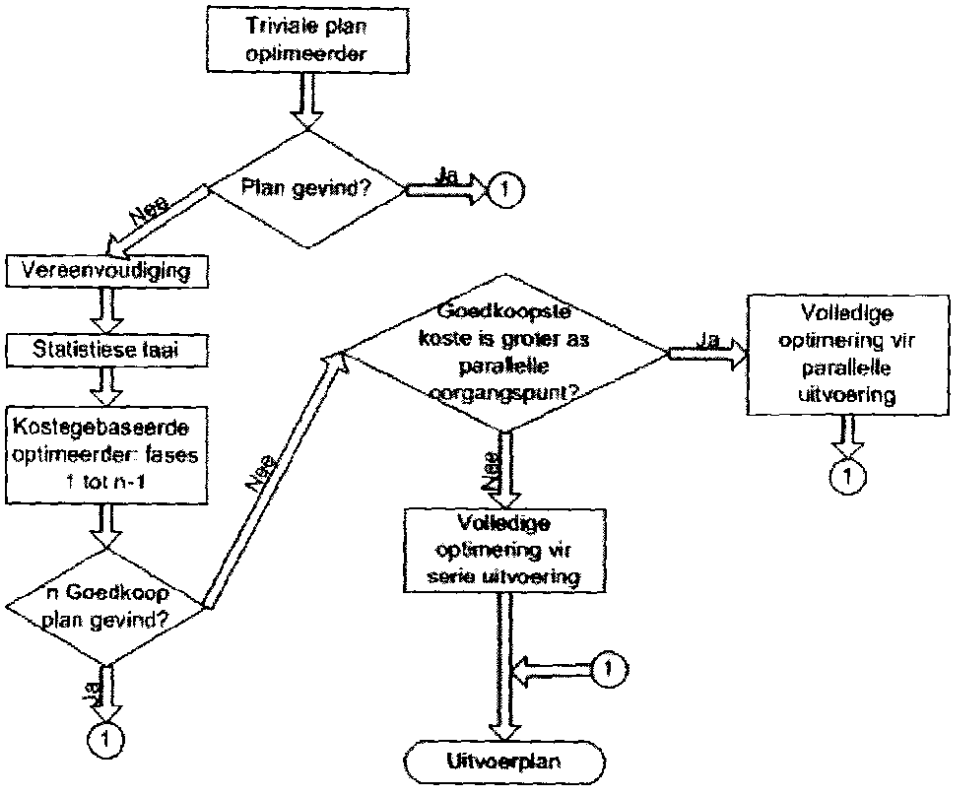
3.2.7.2 Optimering

Die navraagoptimeerder is kostegebaseer, met ander woorde die goedkoopste uitvoerplan word vir elke SQL-uitdrukking verkry. Die koste word bereken ten opsigte van die aantal hulpbronne wat gebruik word. Die optimeerder beoordeel die uitvoerplanne en kies die een met die laagste koste. Sommige SQL-bevele kan 'n groot hoeveelheid uitvoerplanne voorlê. Die optimeerder analiseer nie elke moontlikheid nie, maar probeer die uitvoerplan verkry wat die naaste aan die teoretiese minimum kom. Die koste word bereken ten opsigte van watter plan die vinnigste terugvoer met 'n aanvaarbare koste vir hulpbronne aan die gebruiker sal verskaf. 'n Voorbeeld hiervan is die parallelle prosessering van 'n navraag wat meer hulpbronne gebruik (byvoorbeeld meer as een verwerker (SVE)), maar die vinnigste terugvoer aan die gebruik sal verskaf. Die optimeringsproses word in figuur 3.3 getoon en daarna verduidelik.

Die kostegebaseerde optimering is 'n hulpbronintensiewe proses en moet vermy word as dit kan. As die optimeerder bewus raak dat daar net een moontlike uitvoerplan vir die navraag bestaan, word die plan gebruik. Die metode word in die triviale plan optimeerder gebruik en kan baie effektief wees. As die triviale plan optimeerder nie 'n eenvoudige plan kry nie, probeer SQL Server om die navraag te vereenvoudig. Die volgende stap behels die laai van metadata, wat

die statistiese inligting van elke indeks bevat. Die optimeerder gaan dan deur verskillende fases van kostegebaseerde optimering.

Figuur 3.3 – Die stappe van die optimeersproses tydens navraagverwerking.



Die kostegebaseerde optimeerder gebruik 'n stel transformasiereëls wat verskeie permutasies van indekse en verbindingstrategieë deurgaan. As gevolg van die groot hoeveelheid moontlike permutasies word fases gebruik vir die optimering. Elke fase besit 'n stel reëls. Na elke fase word die koste van die plan wat verkry is, bereken en uitgevoer indien dit goedkoop (met ander woorde aanvaarbaar ten opsigte van hulpbrongebruik) is. As die plan nie voldoende was nie, word die volgende fase ingegaan. As die optimeerder deur al die maklikste eerste fases sonder 'n resultaat beweeg het, word die beste plan tot dusver geneem en vergelyk met 'n spesifieke oorgangspunt ("threshold"). As die koste bokant die oorgangspunt is, word 'n fase betreë wat die volledige optimeringsfase genoem word. Die fase maak die aanname dat die plan in parallel (meer as een sentrale verwerkingseenheid (SVE, "CPU")) verwerk moet word. As die betrokke rekenaar net 'n enkele SVE besit of die rekenaar se prosesseringstyd beperk is, sal parallelle verwerking nie gebruik word nie, anders probeer die optimeerder 'n goeie parallelle plan verkry. As die koste van die beste plan onder die parallelle oorgangspunt is, word 'n brutekragmetode gevolg om 'n serieplan te verkry deur elke moontlike kombinasie van indekse en prosesseringstrategieë te vergelyk.

3.2.7.3 Interne werking van die navraagoptimeerder

Die optimeerder bereken 'n koste vir elke navraag deur die aantal logiese leesprosesse en geheuehulpbronne in ag te neem wat vereis word en beskikbaar is. Die optimering geskied in drie stappe, naamlik navraaganalise, indeksseleksie en verbindingseleksie ("join"), alhoewel indeksseleksie en verbindingseleksie baie van dieselfde prosesse benut.

Elkeen van die stappe en die hantering van "GROUP BY"-, "DISTINCT"- en "UNION"-bewerkings word kortliks in die volgende paragrawe bespreek.

3.2.7.3.1 Navraaganalise

Elke gedeelte van die navraag word ondersoek om te bepaal of die betrokke gedeelte geoptimeer kan word. Die gedeelte word geklassifiseer as 'n soekargument of as deel van 'n verbindingskriteria, indien dit wel geoptimeer kan word. Die soekargument gedeelte kan indekse gebruik om data te verkry. Die soekargument beperk die soektog aangesien dit gelykheidstoetse, reekswaardes of saamvoeging van twee of meer items wat verbind is deur "AND" spesifiseer. Die vorm van die soekargument is 'n kolom, gevolg deur 'n spesifieke operator, wat gevolg is deur 'n konstante of veranderlike. Die volgorde kan ook omgekeerd wees, met die konstante of veranderlike aan die begin. Die operatore sluit =, >, <, ≤, ≥, "BETWEEN" en soms "LIKE" in (sien bylaag A). Die "wildcard"-karakters (paragraaf 3.4 in bylaag A) bepaal of die "LIKE"-operator 'n soekargument tot gevolg het. As die "wildcard"-karakter aan die begin van die uitdrukking is, soos byvoorbeeld "LIKE '%JOHN' ", sal die indeks nie toegepas kan word nie en is die uitdrukking nie 'n soekargument nie. 'n Paar voorbeelde van soekargumente volg:

- naam = 'Piet'
- salaris > 40000
- 60000 < salaris
- naam = 'Piet' AND salaris > 100000

Die laaste voorbeeld bied 'n situasie waar, indien 'n indeks op (*naam, salaris*) bestaan, slegs een soekargument voorgestel word en een indekssoektog gebruik word. As die uitdrukking nie 'n soekargument is nie, sal SQL Server al die rye moet deursoek om die gesoekte rye te vind en 'n indeks kan nie gebruik word nie. 'n Goeie voorbeeld is "naam = 'Piet' OR salaris > 100000", aangesien altwee kriteria moontlik waar kan wees. Elke uitdrukking van die vorige voorbeeld mag egter apart wel 'n soekargument wees.

SQL Server probeer om skalaarvereenvoudiging te doen, om sodoende dalk 'n soekargument te verkry. Die uitdrukking "WHERE prys * 12 = koste * 1" sal, indien *koste* 'n indeks besit, vereenvoudig word na "WHERE prys = koste / 12", aangesien die indeks minder rekords bevat (vermenigvuldiging met 1 is vanselfsprekend). Die vereenvoudigde uitdrukking word net gedurende optimering gebruik en wanneer bepaal word of die ry in die resultaat sal voorkom, word die oorspronklike uitdrukking gebruik.

In die geval van 'n ontbondelde indeks (met elke sleutelwaarde in die blaarnodusse wat wysers is na die fisiese plek van die ry op die hardeskyf) kan te veel rye moontlik die soekargumente bevredig en die optimeerder mag besluit dat 'n gewone soektog deur die hele tabel goedkoper sou wees. Die optimeerder gebruik nie altyd die indeks tydens die uitvoering van die navraag nie, maar die indeks kan nog steeds handig gebruik word tydens optimering om die aantal rye, wat deel vorm van die resultaat, te bereken. Die aantal rye word gebruik in besluite om die beste plan te verkry.

3.2.7.3.2 Indeksseleksie

Die tweede stap van optimering is indeksseleksie. Die navraagoptimeerder bepaal of 'n indeks vir die soekargument-uitdrukking bestaan, bepaal die hoeveelheid rye wat verkry gaan word, en voorspel die koste om die rye te vind. Die indeks word oorweeg indien die indeks se eerste kolom in die soekargument voorkom en die soekargument 'n onderste, of boonste, of beide grense bepaal om die navraag te beperk. Die indeks word ook oorweeg indien al die kolomme in die navraag in die indeks voorkom, selfs al is nie een van die kolomme die eerste kolom van die indeks nie. Hierdie indeks wat al die verwysde kolomme bevat, word 'n oordekkingsindeks ("covering index") genoem en is een van die vinnigste maniere om die data te bereik, aangesien al die data uit die indeks verkry kan word. Ontbondelde indekse is baie keer oordekkingsindekse, aangesien die indeks die gebondelde indeks (indien dit bestaan) se sleutelwaarde as deel van 'n verwysing bevat.

Indeks statistiese inligting speel 'n belangrike rol in die keuse van indekse. Die optimeerder neem die indekse wat tot dusver oorweeg word en evalueer die indeks volgens die aantal rye wat verkry word. Die statistiese inligting word verkry uit 'n tabel "sysindexes" en die histogramwaardes in die rye van die tabel word elke keer bygewerk met die uitvoer van die "UPDATE STATISTICS"-bevel. SQL Server verseker dat die statistiese inligting op datum is deur die inligting tydens die optimeringfase by te werk. Die histogramwaardes bestaan uit 'n ewekansige steekproef van die waardes vir die indekssleutel volgens die huidige bestaande data. Die aantal bladsye in die tabel of indeks is belangrike inligting vir die optimeerder.

Inligting oor die uniekheid (of digtheid) van die data verskaf 'n maatstaf van die selektiwiteit van die indeks. 'n Hoër selektiwiteit beteken dat meer rye uitgeskakel kan word. Unieke indekse is die meeste selektief en die digtheid se waarde is *1 / aantal rye in die tabel*. Digtheidwaardes is tussen 0 en 1, en hoë selektiewe indekse het 'n digtheid van 0.10 of laer. As voorbeeld het 'n unieke indeks op 'n spesifieke tabel, 8345 rye en is die digtheid 0.00012 (1 / 8345). Gestel 'n nie-unieke ontbondelde indeks het 'n digtheid van 0.2165 op dieselfde tabel. Elke indekssleutel wys na ongeveer 1807 rye (0.2165 * 8345). Dit is waarskynlik nie selektief genoeg om die indeks bo 'n tabelsoektog te kies nie. Dit beteken dat ongeveer 1807 data-toegange benodig word aangesien geen gebondelde indeks op die tabel voorkom nie.

Die enigste geval waar 'n databladsy nie weer gelees word nie, is wanneer twee opeenvolgende indekselemente toevallig na dieselfde databladsye verwys. Indien 40 rye op 'n bladsy pas, sal ongeveer 209 bladsye bestaan (8345 / 40). Die kans dat twee opeenvolgende indekselemente na dieselfde bladsy kan wys, is ongeveer 0,5% (1 / 209). Die aantal logiese toegange vir die databladsye, sonder indeks-lees- of -skryf-aksies, is ongeveer 1807. Die hele tabel kan egter deursoek word met net 209 logiese lees-aksies (die aantal bladsye in die tabel). In hierdie geval sal die optimeerder ander indekse soek of die tabeldeursoek as die beste plan kies.

Die statistiese-inligting-histogram verkry waardes vir die eerste ("lead") kolom van die indeks. Die optimering hou daarmee rekening dat 'n indeks bruikbaar is as, en slegs as, die indeks se eerste kolom in die "WHERE"-gedeelte van die navraag is. Digtheidinligting word egter vir verskeie kombinasies van kolomme in 'n saamgestelde indeks[14] gestoor. SQL Server definieer *digtheid* as *1 / kardinaliteit van die indekssleutels* waar die kardinaliteit die aantal unieke waardes in die data reflekteer. Die optimeerder gebruik die digtheidwaarde om die bruikbaarheid van die indeks vir die verbinding ("joins") van tabelle te bepaal. Nog data wat gestoor word is die tyd van die laaste statistiese ontleding, die aantal rye gebruik om die histogram en digtheidinligting te produseer, die gemiddelde sleutellengte en die digtheid vir ander kombinasies van kolomme.

Die statistiese inligting word met die bevel "DBCC SHOW_STATISTICS" vir elke indeks vertoon. 'n Voorbeeld van hierdie bevel se resultaat word in figuur 3.4 vertoon.

Die statistiese inligting in die figuur 3.4 is op 31 Augustus 2000 laas gewysig. Die tabel het 830 rye en 89 verskillende proefnemings ("samples") is gebruik. Die gemiddelde sleutellengte vir al die kolomme word in die eerste ry resultate van figuur 3.4 vertoon.

[14] Indeks oor meer as een kolom.

Die volgende gedeelte van figuur 3.4 wys die digtheid en gemiddelde sleutellengte van elke kombinasie van die verskillende kolomme in die indeks.

Figuur 3.4 – Resultaat van die bevel
"DBCC SHOW_STATISTICS('Orders', 'cust_date_indx')" op 'n voorbeelddatabasis.

```
Statistics for INDEX 'cust_date_indx'.
Updated     Rows   Rows Sampled  Steps  Density  Average key length
----------- ------ ------------- ------ -------- ------------------
Aug 31 2000 830    830           89     0.0      22.0

All density     Average Length  Columns
--------------- --------------- ------------------------------
1.1235955E-2    10.0            CustomerID
1.2150669E-3    18.0            CustomerID, OrderDate
1.2048193E-3    22.0            CustomerID, OrderDate, OrderID

RANGE_HI_KEY RANGE_ROWS  EQ_ROWS DISTINCT_RANGE_ROWS  AVG_RANGE_ROWS
------------ ----------- ------- -------------------- --------------
ALFKI        0.0         6.0     0                    0.0
...
```

Die aantal unieke waardes vir *CustomerID* is ongeveer 89 (verkry deur 1 / (0.011235955 digtheid)). Die unieke waardes vir die kolomme *CustomerID* en *OrderDate* is ongeveer 823 (1 / (0.0012150669 digtheid)) en vir die kolomme *CustomerID*, *OrderDate* en *OrderID* is ongeveer 830 (1 / (0.0012048193 digtheid)) wat beteken dat die sleutel met die drie kolomme uniek is vir al die rye in die tabel.

Die laaste gedeelte in figuur 3.4 (voorgestel deur "...") vertoon 'n maksimum van 200 proefnemingwaardes. Die reeks sleutelwaardes tussen elke proefnemingwaarde word 'n stap ("step") genoem. Die proefnemingwaarde is die einde van die reeks. Die resultaat van die bevel word breedvoerig deur Delaney (2001:828-833) beskryf.

SQL Server stoor ook statistiese data oor kolomme met geen indekse. Die digtheid en waarskynlikheid dat 'n sekere waarde voorkom, help die optimeerder om die optimum vertalingstrategie te bepaal. Navraagoptimering is gebaseer op waarskynlikheid, en die optimeerder kan moontlik verkeerd wees.

Die berekening van die indeks se koste is belangrik ten opsigte van die keuse van die optimeringsplan. Die moontlikheid ontstaan dat 'n bruikbare indeks dalk te duur ten opsigte van hulpbronne is en van die hand gewys word. Een van die belangrikste kostebepalers is die aantal logiese lees- of skryf-aksies, wat 'n aanduiding is van die aantal bladsye waartoe toegang verkry word. Die logiese lees- of skryf-aksies word getel al is die bladsye reeds in die kasgeheue of op die skyf. Die verhouding van lees- of skryf-aksies wat plaasvind in die

kasgeheue teenoor die logiese lees- of skryf-aksies word die kasgeheue-tref-verhouding ("cache-hit ratio") genoem.

Die optimeerder bereken die moontlikheid dat die waarde in die indeks gevind gaan word deur die digtheid en stapwaardes in die statistiese inligting te gebruik. Die optimeerder gebruik hierdie waardes om te bepaal hoeveel rye aan 'n gegewe soekargument voldoen en hoeveel logiese lees-aksies die kwalifiserende rye sal onttrek. Die optimeerder poog om die toegangsmetode te kry wat die minste logiese lees-aksies tot gevolg sal hê. As 'n indeks gebruik word om die kwalifiserende rye te vind, word die indeks die dryfveer van die deursoek ("scan") genoem. As 'n ontbondelde indeks gebruik word, maak die optimeerder die aanname dat die bladsye wat elke kwalifiserende ry bevat, waarskynlik nie dieselfde bladsye sal wees as die voorafgaande bladsye met kwalifiserende rye nie. In die geval van 'n gebondelde indeks sal die volgende ry waarskynlik op dieselfde bladsye as die vorige ry voorkom omdat die blaarnodusse die data self bevat. Indien die laaste ry van 'n bladsy verkry is en nog 'n ry verkry moet word, sal die volgende bladsy gelees word. Die moontlikheid bestaan vir ontbondelde indekse dat die bladsy alreeds in die kasgeheue is, maar 'n logiese lees-aksie is steeds nodig. 'n Fisiese lees/skryf-aksie is baie duurder as 'n logiese lees/skryf-aksie. Die optimeerder probeer om so min as moontlik bladsy-lees/skryf-aksies te doen en gevolglik word die logiese en fisiese lees/skryf-aksies verminder. Die koste van verskillende indeksstrukture word in tabel 3.3 gelys.

Tabel 3.3 – Die koste van data-toegang met verskillende indeksstrukture (SQL Server)

Toegangsmetode	**Berekende koste (in logiese lees-aksies en naaste heelgetal)**
Tabeldeursoek	Die totale aantal databladsye van die tabel
Gebondelde indeks	Die aantal vlakke in die indeks plus die aantal databladsye wat deursoek moet word (aantal kwalifiserende rye / rye per databladsy).
Ontbondelde indeks vir 'n "heap"	Die aantal vlakke in die indeks plus die aantal kwalifiserende rye ('n logiese lees-aksie vir die databladsy van elke kwalifiserende ry).
Ontbondelde indeks vir 'n tabel	Die aantal vlakke in die indeks plus die aantal kwalifiserende rye vermenigvuldig met die koste om elke gebondelde indekssleutel te soek.
Ontbondelde-oordekkingsindeks	Die aantal vlakke in die indeks plus die aantal blaarnodusse (kwalifiserende rye / rye per blaarbladsy). Die databladsy word nie gelees nie, aangesien al die inligting in die indekssleutel is.

Die navraagoptimeerder mag meer as een ontbondelde indeks gebruik om die navraag te bevredig. Die kosteberaming vir meervoudige indekse verskil van die berekenings in tabel 3.3.

Soms kan die optimeerder nie die statistiese inligting van die kolomme gebruik nie. Dit is moontlik dat die inligting nie bestaan nie, maar meer waarskynlik dat die waardes in die soekargument veranderlikes is (as 'n spesiale geval wat buite die bestek van hierdie verhandeling val). Die optimeerder gebruik in so geval vasgestelde persentasies om die kwalifiserende rye te bereken, wat baie onakkuraat kan wees. Die vergelykoperator (=) kom neer op 10% van totale aantal rye, kleiner as (<) ongeveer 30%, groter as (>) ook 30% en die "BETWEEN"-operator is ongeveer 10% van die totale rye.

3.2.7.3.3 Verbindingseleksie

As die navraag verskeie tabelle gebruik of die navraag dieselfde tabel verskeie kere gebruik (self-verbinding), bepaal die navraagoptimeerder 'n verbindingstrategie met die laagste koste. Verskeie faktore speel 'n rol in die bepaling, waarvan die verwagte aantal lees-aksies en die hoeveelheid geheue benodig, deel is. Die optimeerder kan tussen drie strategieë kies, naamlik geneste herhaalverbindings ("nested loop joins"), saamvoegverbindings ("merge join") en "hash"-verbindings.

3.2.7.3.3.1 Geneste herhaalverbindings ("nested loop joins")

Hierdie tipe verbindings maak gebruik van geneste iterasies ('n stel herhalings) om deur die data van een tabel te gaan en vir elke ry dan die rekord te verkry wat ooreenstem met die verbindingskriteria in die ander tabel. Die aantal iterasies deur enige van die herhalings is gelyk aan die aantal soektogte wat gedoen moet word. 'n Tabeldeursoek is nie altyd nodig nie, aangesien 'n indeks gebruik kan word. Die algoritme volg in figuur 3.5.

Figuur 3.5 – Aangepaste pseudokode-algoritme van geneste herhaalverbindings ("nested loop joins") gebaseer op Delaney (2001:840)

```
// Lys van tabelle om te verbind word voorgestel deur L
// Lys van kriteria vir elke twee tabelle word voorgestel deur K

Funksie Geneste_herhaal (L, K) verskaf Resultaat_Tabel
   As net een tabel in L is, verskaf die tabel;
   Kry eerste twee tabelle van L genoem A en B;
   Itereer deur tabel A se rye:
      Vir elke ry R_A, itereer deur tabel B se rye:
         As R_B volgens K_AB voldoen,          // Kan indekse gebruik vir hierdie stap
            Voeg R_B by tydelike tabel T;
   Vervang tabelle A en B met T in L;
   Verwyder kriteria K_AB uit K;
   verskaf (Geneste_herhaal(L, K));
Einde van funksie
```

'n Voorbeeld van hierdie verbinding word deur Delaney (2001:840) beskryf.

3.2.7.3.3.2 Saamvoegverbindings ("merge join")

Die voorvereiste vir hierdie verbindingstegniek is dat die tabelle wat verbind moet word, op die verbindingskolom gesorteer is. Elke tabel se gesorteerde rye word dan ineengevleg om een lys te vorm. Die algoritme word in figuur 3.6 getoon.

Figuur 3.6 – Die aangepaste pseudokode-algoritme van 'n saamvoegverbinding gebaseer op Delaney (2001:840-841)

```
Funksie Saamvoeg (Tabel A, Tabel B, kriteria vir verbinding K) verskaf Resultaat_Tabel
   Verkry eerste rye (R_A en R_B) vir tabel A en B;
   Terwyl A en B nog rye oor het:
      As R_A se waarde = R_B se waarde volgens K,
         Voeg R_A en R_B in tydelike tabel T;
         As R_A se opvolger (indien die bestaan) se waarde gelyk is aan R_A se waarde,
            Skuif R_A aan na volgende ry in A;
            &R_B = begin_wyser                    // Duplikate
         Anders,
            Skuif R_B aan na volgende ry in B;
      Anders, As R_B se waarde < R_A se waarde volgend K,
         Skuif R_B aan na volgende ry in B;
      Anders,
         Skuif R_A aan na volgende ry in A;
         begin_wyser = &R_B;
   Einde van terwyl;
   verskaf T;
Einde van funksie
```

In sommige gevalle bereken die optimeerder dat die koste van sortering van een of beide tabelle steeds laer is as die alternatiewe, en gevolglik word sortering en samevoeging eerder gebruik. As duplikate voorkom in beide tabelle wat verbind moet word, moet een van die tabelle herhaaldelik teruggaan na die begin van die tabel se duplikate soos wat elke duplikaat in die ander tabel verwerk word.

Delaney (2001:840-841) beskryf 'n voorbeeld van hierdie verbinding.

3.2.7.3.3.3 "Hash"-verbindings

Die verbinding gebruik dieselfde beginsels as die "hash"-indeks (sien paragraaf 2.4.1.2). 'n Eienskap word benodig om die data in verskillende gleuwe te verdeel, wat beheerbare groottes en eweredige verspreiding verseker. Wanneer twee tabelle verbind word met hierdie verbindingsmetode word een van die tabelle gebruik om die "hash"-gleuwe te bou. Die ander tabel word nou ry vir ry ondersoek en 'n ooreenstemmende waarde word in die bestaande gleuwe gesoek. SQL Server probeer altyd die kleinste tabel gebruik om die gleuwe te bou. Die gleuwe word gestoor in 'n geskakelde lysstruktuur en die elemente bevat slegs die kolomme van

die boutabel wat benodig word. Die versameling geskakelde lyste word die "hash"-tabel genoem. Die algoritme word in figuur 3.7 vertoon.

Figuur 3.7 – Die aangepaste pseudokode-algoritme vir "hash"-verbindings gebaseer op Delaney (2001:842-843).

```
Funksie Hash_verbinding(Tabel A, Tabel B, kriteria vir hash K) verskaf Resultaat_Tabel
   Skep nuwe leë Hash tabel H;
   Verkry die kleinste tabel van A en B, genoem S (die ander is G);
   Itereer deur die rye R_S van S:          // Indien nodig, plaas gedeelte vanaf skyf in geheue
      Bereken hash-gleuf E wat binne H is, volgens hash-funksie (gebruik K);
      Voeg relevante kolomme van R_S in element in E by;
   Itereer deur die rye R_G van G:          // Indien nodig, plaas gedeelte vanaf skyf in geheue
      Bereken hash-gleuf E wat binne H is, volgens hash-funksie (gebruik K);
      Itereer deur E se elemente:
         As element ooreenstem met R_G se relevante kolomme,
            Voeg element in T;
   Verwyder H;
   Verskaf T;
Einde van funksie
```

Die "hash"-verbinding het 'n hoë geheue-koste. Die ideaal sal wees as die hele "hash"-tabel in die geheue kan pas. As dit nie die geval is, verdeel SQL Server beide tabelle in gedeeltes, elkeen met 'n stel gleuwe, en skryf die gedeeltes na die skyf. Soos wat elke gedeelte benodig word, word dit in die geheue gelaai. Die tipe "hash"-bewerking staan bekend as 'n "Grace hash".

SQL Server besit 'n alternatief waar gedeeltes slegs uitgeskryf word soos benodig, wat sommige gedeeltes in die geheue en ander op die skyf tot gevolg sal hê. Indien die keuse van die kleinste tabel gemaak kan word, word die effektiwiteit van die "hash"-verbinding verhoog. Die situasie ontstaan waar SQL Server tydens die uitvoer van die verbinding agterkom dat die keuse foutief was en die tweede tabel eintlik die kleinste was. SQL Server besit die vermoë om die twee tabelle dan om te ruil ("role reversal").

"Hash"- en saamvoegverbindings kan slegs toegepas word indien die verbinding 'n gelykheidsverbinding is ("equijoin"), met ander woorde 'n verbinding wat gelykheidstoets op die relevante kolomme doen. Die "hash"- en samevoegingmetodes gebruik baie meer geheue as die geneste herhaalmetode en die optimeerder sal die geneste herhaalmetode kies indien die geheue 'n beperkende faktor op die stelsel is.

3.2.7.3.4 Hantering van "GROUP BY", "DISTINCT", "UNION"

Buiten die indeks- en verbindingstrategieë-keuse moet die optimeerder besluit op die metode om "GROUP BY", "DISTINCT" en "UNION" (sien bylaag A) te hanteer. Die drie bewerkings "GROUP BY", "DISTINCT" en "UNION" word in die volgende paragrawe bespreek.

3.2.7.3.4.1 "GROUP BY"-bewerking

SQL Server 2000 kan die bewerking op twee maniere uitvoer. Die een manier is om die data (nadat soekargumente toegepas is) te sorteer en dan groepe te vorm van die gesorteerde data. Die tweede manier is om "hash"-funksies te gebruik om die data in groepe te allokeer en opsommende funksies ("aggregates") soos byvoorbeeld SUM (sommeer die kolom en gee die totaal terug) te bereken. Die algoritme neem elke rekord van die invoertabel en bepaal vir elkeen die gleuf. Die gleuf word deursoek vir 'n ooreenstemmende waarde. As die waarde bestaan, word die opsommende funksie toegepas op beide die nuwe element en die ooreenstemmende element in die gleuf, en die nuwe verkrygde element word in die gleuf in die plek van die ou ooreenstemmende element geplaas. As die waarde nie bestaan nie, word die element net tot die gleuf bygevoeg. Die verkrygde "hash"-tabel word as resultaat gegee. Die "hashing" metode sorteer nie die data nie, maar die data se orde is afhanklik van die orde van die elemente in die "hash"-gleuwe.

3.2.7.3.4.2 "DISTINCT"-bewerking

Dieselfde twee metodes as vir "GROUP BY"-bewerkings is ook vir die "DISTINCT"-bewerking beskikbaar. Die verskil is: as die waarde bestaan, word die nuwe element wat vergelyk word, verwyder, en as die waarde nie bestaan nie, word die nuwe element in die gleuf gevoeg en as uitvoer teruggestuur.

3.2.7.3.4.3 "UNION"-bewerking

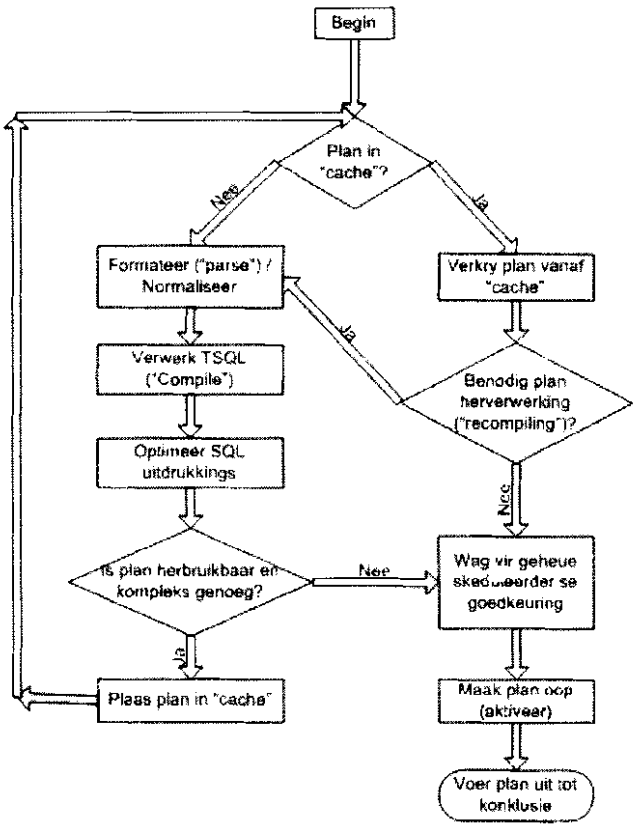
Die bewerking neem die resultate van twee verskillende navrae en verenig die resultate in 'n enkele resultaat. As "UNION ALL" gebruik word, word al die rye van beide navrae geneem as die finale resultaat (duplikate ingesluit). As "ALL" nie gespesifiseer word nie, word die duplikate verwyder voor die enkele resultaat teruggestuur word. SQL Server ondersteun drie maniere om die duplikate te verwyder. Die eerste is om "hashing" te gebruik soos vir die "DISTINCT"-bewerking. Die tweede manier is om die navrae se resultate te sorteer en dan 'n samevoegingsproses ("merge") te gebruik om een resultaat te verkry. Die laaste metode is om die resultate van die twee navrae saam te stel en dan te sorteer om die duplikate te verwyder.

Statistiese inligting op beide indekskolomme en nie-indekskolomme word baie deur SQL Server se optimeerder gebruik om die beste strategie vir vertaling te bereken. Wanneer 'n indeks geskep word, word die statistiese inligting bereken en gestoor. Die statistiese inligting word bygewerk wanneer die optimeerder besluit dat die statistiese inligting verouderd is, of wanneer die statistiese inligting benodig word. Die optimeerder bepaal veroudering ten opsigte van die aantal ry-bywerkings wat sedert die laaste bywerking van die statistiese inligting plaasgevind het. Die bevele "UPDATE STATISTICS" en "CREATE STATISTICS" word gebruik om die statistiese inligting onderskeidelik by te werk en te skep as 'n handtransaksie (Delaney, 2001:848).

3.2.7.4 Navraaguitvoerplan

Die eindproduk van die vertalings en optimeringsfase is die navraagplan wat gewoonlik in die prosedurekasgeheue geplaas word. Planne wat nie herbruikbaar is nie, word gewoonlik nie in die prosedurekasgeheue geplaas nie, sodat die kasgeheue eerder vir ander herbruikbare planne gebruik kan word. As die plan ten opsigte van hulpbronne baie goedkoop is om te verwerk, is dit onnodig om die plan in die prosedurekasgeheue te plaas. Figuur 3.8 toon die stappe waarvolgens die uitvoerplan hanteer word.

Figuur 3.8 – Stappe van die ontwikkeling van die navraaguitvoerplan gebaseer op Delaney (2001:850).



As die uitvoerplan nie in die kasgeheue bestaan nie, word een geskep. Indien die plan geskep moet word, word die linkerhelfte van die figuur, wat ooreenstem met die fases van vertaling en optimering, gebruik. As die plan nie herbruikbaar is nie, word die plan voorgelê vir die geheue-skeduleerder ("memory grant scheduler"). Indien die plan herbruikbaar is, met ander woorde die plan is nie te eenvoudig en sal weer gebruik kan word, word die plan in die kasgeheue gestoor. SQL Server verseker dat die plan se toestand nog dieselfde is. As die toestand verander het, moet die plan weer verwerk word. 'n Paar oorsake vir die hervertaling ("recompilation") van die plan is byvoorbeeld die verandering van indekse of die statistiese inligting rondom die kolomme. Die plan sal nie herverwerk word indien omgewingsveranderings (soos byvoorbeeld 'n verandering in die hoeveelheid beskikbare geheue) plaasvind nie.

Sommige navrae benodig 'n groot hoeveelheid geheue en die effektiewe allokering van hierdie geheue kan probleme veroorsaak. SQL Server bepaal tydens optimering twee aspekte oor die geheue vir die navraag, naamlik die minimum hoeveelheid geheue benodig vir die navraag om effektief te kan uitvoer (gestoor saam met navraaguitvoerplan) en die maksimum hoeveelheid geheue wat navraag kan bevoordeel (byvoorbeeld 100 megagrepe vir 'n sorteringsproses wat op 'n tabel van grootte 100 megagrepe toegepas word). Wanneer die uitvoerplan by die geheue-skeduleerder ("memory grant scheduler") kom, word bepaal of 'n sortering- of "hash"-bewerking benodig word. Indien nie, kan die plan dadelik uitgevoer word.

In die geval van hulpbronintensiewe navrae, word net 'n paar gelyktydig uitgevoer om sodoende voldoende geheue te verseker. As ten minste die helfte van die maksimum geheue wat vir die navraag benodig word, beskikbaar is, word die navraag uitgevoer, anders wag die navraag in die geheue-toekenningslys. SQL Server gee nooit meer as die helfte van die oorblywende navraaggeheue aan 'n enkele navraag nie. Na die geheue-toekenning, word die plan tot voltooiing uitgevoer.

3.2.7.5 Prosedurekasgeheue

Die prosedurekasgeheue is nie 'n aparte geheue-area nie. Die bufferkasgeheue stel die geheue beskikbaar vir die prosedurekasgeheue. Uitsonderlike gevalle bestaan wel waar SQL Server geheue direk uit die bedryfstelselgeheue allokeer, maar hierdie gevalle is baie skaars. Die planne word, saam met die koste om die plan te vertaal ("compile"), in die kasgeheue gestoor. As 'n "ad hoc"-navraag in die kasgeheue gestoor word, is die koste 0, wat beteken dat die navraag dadelik uitgevoer mag word. Indien geheue egter 'n probleem begin raak, word die "ad hoc"-navrae eerste verwyder. Die koste van vertaling word in eenhede van skyf-lees/skryf-aksies gestoor. As die "ad hoc"-navrae gebruik word, word die koste elke keer met een eenheid vermeerder. Hierdie vermeerdering hou aan totdat die werklike vertalingskoste van die "ad

hoc"-navraag bereik word, waarna dit nie meer kan vermeerder nie. Die gevolg van hierdie vermeerdering is dat die "ad hoc"-navraag langer in die geheue sal bly. As die navraag nie 'n "ad hoc"-navraag is nie (met ander woorde 'n gestoorde prosedure, 'n voorbereide navraag of 'n outo-geparametriseerde navraag (SQL Server maak 'n raaiskoot oor watter konstantes moontlike parameters kan wees)), word die koste na gebruik herstel na die oorspronklike vertalingskoste. Die algemene beginsel is: elke keer wat 'n plan in die kasgeheue gesoek word, word die koste van die ander planne verminder.

Die "lazywriter" (sien paragraaf 3.2.4.3) hanteer die koste van die planne en is verantwoordelik daarvoor om die planne uit die kasgeheue te verwyder, indien nodig. Die "lazywriter", wat eintlik deel vorm van die stoorenjin, gebruik dieselfde metode van geheuehantering vir navraagplanne, data- en indeksbladsye, omdat die planne in die normale bufferkasgeheue gestoor word. Die "lazywriter" kyk deur al die bufferopskrifte ("headers") in die stelsel. As geheue nie 'n probleem is nie, is die "lazywriter" se spoed stadig, maar soos wat geheue meer in aanvraag raak, loop die "lazywriter" meer gereeld. Die "lazywriter" kyk na die huidige koste vir die bladsy in die buffer. As die koste 0 is, word die bladsy verwyder om geheue oop te maak en as die buffer 'n prosedureplan bevat het, word die SQL-bestuurder geroep om bykomende skoonmaak te doen. Die buffer word dan vir hergebruik op die vry-lys geplaas. As die koste groter as 0 is, verminder die "lazywriter" die koste en evalueer dan ander buffers. Die "lazywriter" onderskei dus nie tussen navraagplanne in die kasgeheue en data- of indeksbladsye in die kasgeheue nie.

Gestoorde prosedures en kasbergingmetodes – Gestoorde prosedures is die mees koste-effektiewe metode om SQL-bevele uit te voer. Die prosedures benodig nie vertaling wanneer die prosedure uitgevoer word nie, behalwe wanneer hervertaling nodig is. Hervertaling moet waar moontlik vermy word, aangesien dit bykomende vertaling- en optimeringstyd bylas. Gestoorde prosedures kan outomaties of met 'n hand-aksie herverwerk word. In sekere gevalle kan hervertaling beter navraaguitvoerplanne lewer en is hervertaling voordelig. Gestoorde prosedures is 'n goeie opsie wanneer verskeie toepassings navrae uitvoer waarvan die parameters bekend is. SQL Server bied, buiten gestoorde prosedures, vier moontlikhede om die herverwerkproses uit te skakel:

- "Ad hoc"-kasberging – Die "ad hoc"-navrae word in die kasgeheue geplaas en indien die navrae herhaaldelik gebruik word, sal dit in die kasgeheue bly.

- Outo-parametrisering ("Autoparameterization") – SQL Server skep 'n basiese formaat ("template") vir eenvoudige navrae waar konstantes gesien word as parameters. Opeenvolgende navrae wat in dieselfde formaat is, kan dan dieselfde plan volg, met net die

parameter wat verander. As voorbeeld sal die volgende twee navrae dieselfde plan gebruik (sien bylae A vir SQL sintaksis) :

SELECT naam, van, titel FROM werknemer WHERE werknemer_id = 6;
SELECT naam, van, titel FROM werknemer WHERE werknemer_id = 2;

SQL Server hanteer hierdie twee navrae deur die konstantes met 'n parameter te vervang:
SELECT naam, van, titel FROM werknemer WHERE werknemer_id = @p;

- Die "SP_EXECUTESQL"-prosedure – Die spesifieke gestoorde prosedure se werking is tussen "ad hoc"-kasberging en gestoorde prosedures. Die sintaksis lyk soos volg:

  SP_EXECUTESQL *@batch_text, @batch_parameter_definitions,*
  param1, ..., paramN

  Die volgende vertoon 'n paar voorbeelde om hierdie metode te gebruik:

  EXEC SP_EXECUTESQL N'SELECT naam, van, titel FROM werknemer WHERE werknemer_id = @p', N'@p tinyint', 6
  EXEC SP_EXECUTESQL N'SELECT naam, van, titel FROM werknemer WHERE werknemer_id = @p', N'@p tinyint', 2

  Dieselfde gekaste ("cached") plan word vir beide voorbeelde gebruik. Die metode word gewoonlik gebruik vir 'n enkele gebruiker wat navrae verskeie kere uitvoer en waar die parameters bekend is.

- Die voorberei-en-uitvoer-metode – Die metode werk op dieselfde beginsel as "SP_EXECUTESQL", maar die SQL-uitdrukking ("batch_text") word nie elke keer saamgestuur nie. Die uitdrukking word gestuur in die voorberei-stadium en 'n beheerwyser ("handle") word verkry om die proses uit te voer. Die metode word gebruik vir verskeie gebruikers wat navrae uitvoer waarvan die parameters bekend is of vir enkelgebruikers wat dieselfde navrae verskeie kere sal uitvoer.

Die bevel "DBCC FREEPROCCACHE" word gebruik om alle gekaste planne uit die geheue te verwyder. Die bevel "DBCC FLUSHPROCINDB(<databasis>)" word gebruik om in 'n spesifieke databasis die planne uit die kasgeheue te verwyder. 'n Verdere bevel "DBCC DROPCLEANBUFFERS" bestaan wat skoon buffers vanuit die bufferarea verwyder sonder om

SQL Server af te skakel en weer aan te skakel. Die bevele moet egter net tydens toetsing gebruik word, aangesien dit produktiwiteit affekteer.

Delaney (2001:862-863) stel voor dat gestoorde prosedures so ver as moontlik gebruik word. Die prosedures bied die voordele van voorafverwerkte navrae, verminderde netwerkvloei (aangesien die SQL-uitdrukking nie oor die netwerk gestuur word nie) en 'n sekuriteitsmeganisme om die toegang tot data te beheer. Gestoorde prosedures bied 'n vlak van abstraksie tussen die toepassing en die databasisontwerp.

3.2.7.6 Uitvoering

Die finale stap in die navraagvertalingsproses neem die plan wat die optimeerder voorstel en voer dit uit. Die uitvoer-enjin skeduleer "threads" vir die prosesse en verskaf kommunikasie tussen die verskillende "threads".

Die gebruiker van 'n databasis stel gewoonlik daarin belang om die data so gou as moontlik te verkry en nie noodwendig in hoe lank dit neem om al die data te verkry nie. Delaney (2001, 874-875) verduidelik hoe SQL Server hierdie situasie hanteer. Gestel SQL Server moet tussen 'n ontbondelde indeks (geen gebondelde indeks) en 'n tabeldeursoek besluit om die navraag se resultaat te verkry en dat die resultaat 'n groot gedeelte van die tabel se rekords bevat. Normaalweg sal SQL Server die tabeldeursoek bo die indeks kies, aangesien die aantal rye so baie is. Die deursoek mag baie minder lees/skryf-aksies verlang as die indeks. Die keuse word egter beïnvloed deur die tyd wat die eerste rye as resultaat gelewer kan word. Die deursoekmetode sal gewoonlik eers teen die einde van die verwerking resultate begin lewer, terwyl die indeks die eerste rye baie vinniger beskikbaar stel. 'n Navraagleidraad "FAST" (sien bylaag A) bestaan om SQL Server te dwing om die eerste *n* rye vinniger terug te stuur, al is dit ten koste van die totale tyd.

3.2.8 Toepassings en gereedskap in SQL Server-pakket

SQL Server bied 'n hele paar bykomende toepassings en gereedskap om navrae te monitor en te optimeer. 'n Indeksoptimeringsghoeroe ("Index Tuning Wizard") bestaan wat die keuse van indekse vergemaklik. Die SQL Query Analyzer-program vergemaklik die skryf van navrae en bied statistiese inligting oor die betrokke navrae om sodoende te bepaal of die navrae effektief is of nie. Microsoft SQL Enterprise Manager is 'n kragtige administrasieprogram wat verskeie take soos replikasie, roltoekenning en administrasie van databasisse met 'n grafiese koppelvlak hanteer. 'n Kort beskrywing van 'n paar van hierdie toepassings word hier gegee:

3.2.8.1 SQL Profiler en SQL Trace.

SQL Server bied twee naspeurkomponente ("trace components"), naamlik die "SQL Trace"-komponent op die bediener se kant en SQL Profiler op die kliënt se kant. Die SQL Trace komponent beheer lyste van gebeurtenisse wat deur gebeurtenisproduseerders geskep word. Gebeurtenisse is die hoofkomponent van naspeuring en kan bewerkings binne SQL Server of tussen SQL Server en 'n kliënt wees. 'n Stelsel gestoorde prosedures bestaan wat bepaal watter gebeurtenisse nagespeur word, watter data vir elke gebeurtenis belangrik is en waar die inligting gestoor moet word.

SQL Profiler hou tred met elke uitdrukking na SQL Server wat uitgevoer (ook die in gestoorde prosedures) is, elke tabel wat gebruik word, elke slot wat benodig word en ook elke fout wat voorkom. SQL Profiler neem baie hulpbronne tydens gebruik in beslag. Die toepassing kan die naspeurdata in 'n lêer stoor wat 'n werksladinglêer genoem word. Die lêer word byvoorbeeld deur die "Index tuning wizard" gebruik om effektiewe indeks-implementerings voor te stel.

3.2.8.2 SQL Query Analyzer

Die navraaganaliseerder van SQL Server laat die gebruiker toe om "ad hoc"-navrae, gestoorde prosedures en ook SQL Server-bevele te roep. Die toepassing bied opsies waarmee die werkverrigting van 'n navraag beoordeel kan word en bied 'n toets-area waar alle navrae vir latere implementering eers getoets kan word. Delaney (2001:889-907) bespreek 'n paar van die moontlikhede wat SQL Query Analyzer bied, en dit word kortliks hier verduidelik:

- **Statistiese inligting oor lees/skryf-aksies ("STATISTICS IO"-bevel)** – Die statistiese inligting is nie histogramme en digtheidsinligting nie, maar dui die hoeveelheid werk aan wat SQL Server moet doen om die navraag te verwerk. As hierdie opsies aan is, word bykomende inligting vertoon, soos die aantal logiese[15] en fisiese[16] lees-aksies, die aantal bladsye wat vooruit in die kasgeheue ingelees word[17], die aantal kere wat die tabel deursoek is en die naam van die tabel.
- **Statistiese inligting oor tyd ("STATISTICS TIME"-bevel)** – Die statistiese inligting wys die tyd wat die navrae geneem het om uit te voer. Twee stelle waardes word vertoon, naamlik die tyd wat die ontleding ("parse") en vertaling neem, en die SVE-tyd en totale uitvoertyd wat die navraag neem om uit te voer. In sekere gevalle word twee stelle data vir die ontledings-

[15] Aantal bladsye gelees vanaf die datakasgeheue.

[16] Aantal databladsye gelees vanaf die hardeskyf in die datakasgeheue in.

[17] Aantal bladsye wat in die kasgeheue geplaas word vir die navraag.

en vertalingstyd getoon. Dit gebeur wanneer die navraag in die navraagkasgeheue geplaas word en geformateer word en die tweede tyd dui op die onttrekking van die navraag uit die kasgeheue. As die navraag alreeds in die kasgeheue is, word die onttrekkingstyd se waardes getoon en die ontleding- en vertalingstyd se waardes is nul. Die tweede stel waardes se totale uitvoertyd mag moontlik baie wissel, afhangende van hoe besig die bediener is. Die SVE-tyd van die tweede stel waardes is egter redelik konstant en ook onafhanklik van hoe besig die bediener is, en kan eerder as 'n maatstaaf gebruik word (Brad M. McGehee, 2005).

- **"Showplan"** – SQL Query Analyzer verskaf twee benaderings om die uitvoerplan van 'n navraag te vertoon. Die uitvoerplan kan teks of grafika wees en is opgedeel in verskeie fases. Die uitvoer van "showplan" vertoon inligting oor die metode van prosessering van 'n navraag. Die uitvoer toon die indekse wat gebruik word en die volgorde waarin die verskillende optimeringsprosesse plaasvind. Twee opsies, "SHOWPLAN_TEXT" en "SHOWPLAN_ALL", bestaan om die navraaguitvoerplan te verkry sonder om die navraag uit te voer. Geen resultate, behalwe vir die uitvoerplan, word dus vertoon nie. "SHOWPLAN_TEXT" vertoon die stappe om die navraag te verwerk, insluitende die tipe verbinding, die orde van die tabeltoegange, watter indekse vir elke tabel gebruik is, en die interne sorterings wat toegepas is. "SHOWPLAN_ALL" verskaf dieselfde inligting as "SHOWPLAN_TEXT", met die bykomende inligting van die verwagte rye wat aan die soekkriteria voldoen, die beraamde grootte van die resultaatrye, die beraamde SVE-tyd en die totale koste wat intern gebruik word om hierdie plan met ander planne te meet. Hierdie opsies word gebruik om 'n navraag te optimeer deur indekse by te voeg en te kyk of die indeks deur SQL Server gebruik is.

  Die grafiese "showplan" toon 'n verkleinde weergawe van dieselfde inligting wat "SHOWPLAN_ALL" vertoon. Aangesien die grafiese voorstelling in die eksperimentele gedeelte in hoofstuk 6 gebruik gaan word, word die argumente van die grafiese voorstelling hier gelys:

  - Fisiese bewerking – Die fisiese algoritme wat gebruik is om die resultaat van die spesifieke fase van 'n navraag te kry.
  - Logiese bewerking – Die logiese algoritme wat gebruik is om die resultaat van die spesifieke fase van 'n navraag te kry.
  - Ry aantal – Die aantal rye wat verkry is.
  - Beraamde rygrootte – Die gemiddelde grootte van die ry in grepe.
  - Lees/skryf-aksie-koste – Die beraamde lees/skryf-aksie-koste vir die bewerking.
  - SVE-koste – Die beraamde SVE-koste vir die bewerking.

- Aantal uitvoerings – Die aantal kere wat die bewerking vir die navraag uitgevoer sal word.
- Koste – Die koste ten opsigte van die totale bewerking.
- Subboom ("subtree") koste – Die kumulatiewe koste van al die vorige bewerkings tot by die spesifieke bewerking.
- Verwagte aantal rye – Die raaiskoot wat die optimeerder waag oor die verwagte aantal rye van die fase wat ondersoek word.

Die fisiese en logiese bewerkings kan verskillende prosesse voorstel. Die prosesse word in tabel 3.4 gelys, maar sekere prosesse word nie ingesluit nie (soos byvoorbeeld "trigger"-prosesse, "cursor"-prosesse en data-ontginningsprosesse) aangesien die prosesse nie in die praktiese gedeelte ondersoek word nie en buite die bestek van die verhandeling val. Die prosesse word verduidelik deur die webbladsy van Microsoft Corporation (2005e) en die skakels wat daaruit volg. Normaalweg vertoon die grafiese uitvoerplan die invoertabelle van bo na onder in die volgorde waarin die tabelle in die proses gebruik word. Sekere van die prosesse in tabel 3.4 kom voor in die eksperimentele gedeelte in hoofstuk 6 en 'n kort beskrywing van hierdie prosesse word hier gegee om die leser vertroud te maak met hierdie prosesse. Die prosesse is die kern van die wyse waarop SQL Server optimering toepas.

Tabel 3.4 – Fisiese en logiese prosesse wat SQL Server gebruik om navrae uit te voer

Proses	**Is fisies?**	**Is logies?**	**Beskrywing**
"Aggregate"	Nee	Ja	Groepeer die invoer volgens 'n stel kolomme en pas saamgestelde funksies, soos byvoorbeeld "SUM", op die kolomme toe.
"Assert"	Ja	Ja	Die proses verifieer 'n sekere voorwaarde en indien nodig word 'n foutboodskap verskaf.
"Bookmark Lookup"	Ja	Ja	Die proses gebruik 'n boekmerk (ry-id of gebondelde sleutel) om 'n ooreenstemmende ry in 'n tabel of gebondelde indeks op te spoor.
"Clustered Index Delete"	Ja	Nee	Die proses verwyder gespesifiseerde rye uit 'n gebondelde indeks.
"Clustered Index Insert"	Ja	Nee	Die proses voeg gespesifiseerde rye in 'n gebondelde indeks in.
"Clustered Index Scan"	Ja	Ja	Die proses deursoek 'n hele gebondelde indeks vir gespesifiseerde rye.
"Clustered Index Seek"	Ja	Ja	Die proses soek net vir die gespesifiseerde rye in 'n gebondelde indeks.
"Clustered Index Update"	Ja	Nee	Die proses wysig gespesifiseerde rye in 'n gebondelde indeks.
"Collapse"	Ja	Ja	Die proses optimeer wysigbewerkings deur tydelike en onnodige tussenfase veranderings te verwyder.

"Compute Scalar"	Ja	Ja	'n Uitdrukking word ontleed en 'n skalaar word verkry.
"Concatenation"	Ja	Ja	Deursoek verskeie invoertabelle en verskaf alle rye wat deursoek is.
"Constant Scan"	Ja	Ja	Die proses bring 'n konstante ry in 'n navraag in. Die proses verskaf nul of net een ry wat normaalweg geen kolomme bevat nie. 'n "Compute Scalar"-proses word normaalweg gebruik om kolomme tot die ry van die "Constant Scan"-proses te voeg.
"Cross Join"	Nee	Ja	Die proses verbind elke ry van die eerste (boonste) invoertabel met elke ry van die tweede (onderste) invoertabel.
"Delete"	Nee	Ja	Die proses verwyder rye van 'n objek (wat byvoorbeeld tabelle en indekse kan wees).
"Distinct Sort"	Nee	Ja	Die proses sorteer die rye en verwyder duplikate.
"Distinct"	Nee	Ja	Die proses deursoek al die rye in die invoertabel en verwyder duplikate.
"Distribute Streams"	Nee	Ja	Die proses word in parallelle navraagplanne gebruik. Die proses neem 'n enkele invoerstroom van rekords en verdeel die stroom in verskeie uitvoerstrome sonder om die inhoud of formaat van die rekords te verander.
"Eager Spool"	Nee	Ja	Die proses stoor al die invoer-rye in 'n tydelike tabel wat hergebruik kan word vir verbindings (tensy die rye se data intussen verander het). Die data hoef dus nie weer deursoek te word nie. As die spoel ("spool") se ouer-proses 'n versoek rig vir 'n ry, lees die spoelproses al die rye van die invoertabel en stoor die rye in die spoel.
"Filter"	Ja	Ja	Die invoer word deursoek en die rye wat aan die filtervoorwaarde voldoen, word teruggestuur.
"Flow distinct"	Nee	Ja	Die proses deursoek die invoer en soos wat die ry ondersoek word, word bepaal of die ry 'n duplikaat is. Die rye wat nie duplikate is nie, word teruggegee.
"Full Outer Join"	Nee	Ja	Die proses verbind die eerste invoertabel met die tweede invoertabel volgens die verbindingskriteria. Die uitvoer bevat ook rye van beide invoertabelle wat nie met die ander tabel ooreengestem het nie. Die kolomme van die invoertabel wat nie ooreengestem het nie bevat "NULL"-waardes.
"Gather Streams"	Nee	Ja	Die proses word in parallelle navraagplanne gebruik. Die proses neem verskeie invoerstrome en verskaf 'n enkele uitvoerstroom sonder om die rekords se inhoud of formaat te verander.
"Hash Match Root"	Ja	Nee	Die proses koördineer die bewerkings van ander "Hash Match Team"-prosesse wat onder hierdie proses loop.
"Hash Match Team"	Ja	Nee	Die proses vorm deel van ander "hash"-prosesse wat ooreenstemmende "hash"-funksies en verdelingstrategieë bevat.
"Hash Match"	Ja	Nee	Die proses skep 'n "hash"-tabel vanaf die eerste invoertabel. Die tweede invoertabel se rye word met die rye in die "hash"-tabel vergelyk en resulterende rye (volgens spesifieke kriteria) word as uitvoer verskaf.
"Index Delete"	Ja	Nee	Die proses verwyder rye uit 'n ontbondelde indeks.
"Index Insert"	Ja	Nee	Die proses voeg rye in 'n ontbondelde indeks in.
"Index Scan"	Ja	Ja	Die proses deursoek 'n ontbondelde indeks vir gespesifiseerde rye.
"Index Seek"	Ja	Ja	Die proses soek vir gespesifiseerde rye in 'n ontbondelde indeks.
"Index Spool"	Ja	Nee	Die proses deursoek die invoer-rye en stoor kopieë van elke ry in 'n

			tydelike spoellêer (wat net vir die leeftyd van die navraag bestaan), en bou 'n indeks op die rye.
"Index Update"	Ja	Nee	Die proses wysig rye in 'n ontbondelde indeks.
"Inner Join"	Nee	Ja	Die proses verbind die eerste tabel met 'n tweede tabel, volgens spesifieke verbindingskriteria.
"Insert"	Nee	Ja	Die logiese proses voeg 'n ry in 'n objek (byvoorbeeld 'n tabel of indeks) in.
"Lazy Spool"	Nee	Ja	Die proses stoor al die invoer-rye in 'n tydelike tabel wat hergebruik kan word vir verbindings (tensy die rye se data intussen verander het). Die data hoef dus nie weer deursoek te word nie. As die spoel se ouer-proses 'n versoek rig vir 'n ry, lees die spoelproses 'n ry van die invoertabel en stoor die ry in die spoel.
"Left Anti Semi Join"	Nee	Ja	Die proses verbind die rye van die eerste tabel wat nie 'n ooreenstemmende ry in die tweede tabel het nie volgens spesifieke verbindingskriteria.
"Left Outer Join"	Nee	Ja	Die proses verbind volgens spesifieke verbindingskriteria die eerste invoertabel met die tweede invoertabel. Die rye van die eerste tabel wat nie ooreenstemmende rye in die tweede tabel besit nie, word ook in die resultaat gegee.
"Left Semi Join"	Nee	Ja	Die proses verbind volgens spesifieke verbindingskriteria die eerste invoertabel met die tweede invoertabel.
"Log Row Scan"	Ja	Ja	Die transaksielog word deursoek.
"Merge Interval"	Ja	Ja	Die proses voeg verskeie (moontlik oorvleuelende) intervalle (voorgestel as kolomme in 'n ry) bymekaar om die kleinste moontlike nie-oorvleuelende intervalle te lewer wat gebruik word om indekselemente te soek.
"Merge Join"	Ja	Nee	Die fisiese proses voer die "Inner Join", "Left Outer Join", "Left Semi Join", "Left Anti Semi Join", "Right Outer Join", "Right Semi Join", "Right Anti Semi Join" en "Union" logiese prosesse uit. Die proses vereis dat beide invoertabelle gesorteer is. Sien paragraaf 3.2.7.3.3.2 vir 'n vollediger beskrywing van die saamvoegverbinding.
"Nested Loop"	Ja	Nee	Die fisiese proses voer die "Inner Join", "Left Outer Join", "Left Semi Join" en "Left Anti Semi Join" logiese prosesse uit. Sien paragraaf 3.2.7.3.3.1 vir 'n vollediger beskrywing van die geneste herhaalverbinding metode.
"Parallelism"	Ja	Nee	Die fisiese proses voer die "Distribute Streams", "Gather Streams" en "Repartition Streams" logiese prosesse uit.
"Parameter Table Scan"	Ja	Ja	Die proses deursoek 'n tabel wat as 'n parameter in die huidige navraag gebruik word. Die proses kom vir byvoeginstruksies in gestoorde prosedures voor.
"Remote Delete"	Ja	Ja	Verwyder rekords vanaf 'n objek in 'n afgeleë databasis.
"Remote Insert"	Ja	Ja	Voeg rekords by 'n objek in 'n afgeleë databasis.
"Remote Query"	Ja	Ja	Die navraag word aan 'n afgeleë databasis gerig.
"Remote Scan"	Ja	Ja	'n Afgeleë objek word deursoek.
"Remote Update"	Ja	Ja	'n Afgeleë objek word bygewerk.
"Repartition	Nee	Ja	Die proses word in parallelle navraagplanne gebruik. Verskeie strome

Streams"			word gelees en verskeie strome van rekords word as uitvoer gegee sonder om die inhoud of formaat van die rekords te verander.
"Right Anti Semi Join"	Nee	Ja	Die proses is dieselfde as "Left Anti Semi Join", behalwe dat die onderste tabel die eerste tabel en die boonste tabel die tweede tabel in die verbinding voorstel.
"Right Outer Join"	Nee	Ja	Die proses is dieselfde as "Left Outer Join", behalwe dat die onderste tabel die eerste tabel en die boonste tabel die tweede tabel in die verbinding voorstel.
"Right Semi Join"	Nee	Ja	Die proses is dieselfde as "Left Semi Join", behalwe dat die onderste tabel die eerste tabel en die boonste tabel die tweede tabel in die verbinding voorstel.
"Row Count Spool"	Ja	Nee	Die proses deursoek die invoer en tel die hoeveelheid rye in die invoertabel, en verskaf dieselfde aantal rye sonder data. Die proses word gebruik om die bestaan van rye te verifieer.
"Sequence"	Ja	Ja	Die proses is die dryfveer van 'n groep bywerkplanne. Die proses voer elke invoer in volgorde uit waar elke invoer normaalweg 'n bywerkbewerking van 'n ander objek is. Die laaste ry se uitvoer word gegee.
"Sort"	Ja	Ja	Sorteer die invoer.
"Split"	Ja	Ja	Die proses optimeer bywerkings deur elke bywerkbewerking te verdeel in 'n verwyder en byvoegbewerking.
"Stream Aggregate"	Ja	Nee	Die fisiese proses groepeer die invoer volgens bepaalde kolomme en bereken een of meer saamgestelde funksies.
"Table Delete"	Ja	Nee	Die proses verwyder rye uit 'n tabel.
"Table Insert"	Ja	Nee	Die proses voeg rye in 'n tabel in.
"Table Scan"	Ja	Ja	Al die rye van die invoertabel word deursoek.
"Table Spool"	Ja	Nee	Die invoer word deursoek en 'n kopie van elke ry word in tydelike tabel gestoor vir moontlike gebruik in verbindings.
"Table Update"	Ja	Nee	Die proses wysig rye in 'n tabel.
"Top"	Ja	Ja	Die invoer word deursoek en net 'n sekere aantal rye word teruggestuur.
"Union"	Nee	Ja	Die proses deursoek verskeie invoertabelle en stuur elke ry wat nie 'n duplikaat is nie as uitvoer terug.
"Update"	Nee	Ja	Die logiese proses wysig 'n ry in 'n objek.

"Showplan" is nuttig aangesien dit aantoon waarom 'n bepaalde indeks gebruik is. As die verwagte aantal rye baie verskil van die werklikheid, kan dit beteken dat die statistiese inligting nie bestaan nie of nie op datum is nie. Dit kan ook wees dat die "WHERE"-gedeelte nie 'n soekargument bevat nie. Die navraagoptimeerder (sien paragraaf 3.2.7) kies gewoonlik die plan wat volgens spesifieke kriteria goed genoeg is. 'n Beter plan mag wel bestaan, maar die optimeerder mag langer neem om die plan te kry.

- **Bedienernaspeurinligting** – Die inligting is baie dieselfde as die tipe inligting wat SQL Profiler bied.
- **Kliënt statistiese inligting** – Die inligting fokus op die hoeveelheid werk wat die spesifieke SQL Server kliënt moes doen om die navraag te stuur en die resultate vanaf SQL Server

terug te ontvang. Tipiese inligting is byvoorbeeld die aantal grepe wat oor die netwerk beweeg het.

3.2.8.3 Indeksoptimeringsghoeroe ("Index tuning wizard")

Hierdie toepassing word beskryf aan die hand van Delaney (2001:887-888). Die keuse van indekse is 'n belangrike en moeilike proses (sien paragraaf 4.4). Die ghoeroe kom dus handig te pas tydens die optimering van 'n databasis waarvan 'n groot gedeelte met indekse hanteer kan word.

Hierdie toepassing optimeer 'n enkele databasis per keer en maak gebruik van 'n werksladinglêer, wat inligting bevat soos SQL Server se afgeleëprosedure-opdragte ("remote procedure calls"), SQL-bondel se begin- en eindopdragte, die teks van die afgeleëprosedure-opdragte of bondels of ook SQL-uitdrukkings. As SQL se navraaganaliseerder gebruik word, bestaan die opsie om net die huidige navrae in die analiseerder te optimeer. SQL Profiler skep 'n werksladinglêer wat al die SQL-uitdrukkings deur verskillende gebruikers oor 'n sekere tyd bevat. Die toepassing neem die toegangspatrone van die betrokke gebruikers en tabelle in ag en stel indekse daarvolgens voor.

Volgens Shasha en Bonnet (2003:109) het die ghoeroe twee belangrike negatiewe punte, naamlik die feit dat daar nie prioriteit aan die verskillende uitdrukkings in die werksladinglêer toegeken is nie. Die moontlikheid bestaan dat die ghoeroe nie die beste indeks, wat 'n goeie balans tussen werkverrigting en onderhoud bied, voorstel nie.

3.3 MySQL AB se MySQL 4.1

Die beskrywing van hierdie relasionele databasisbestuurstelsel volg aan die hand van die boeke geskryf deur Balling en Zawodny (2004) en die PDF-handleiding van MySQL 4.1 (MySQL AB, 2005). MySQL 5.0 is tydens die ondersoek van die verhandeling vir produksie beskikbaar gestel en die besluit is geneem om met MySQL 4.1 voort te gaan. Die PDF-lêer reflekteer die webbladsye wat die nuutste inligting oor die spesifieke weergawe van MySQL bevat. Hierdie databasisbestuurstelsel is deel van die sogenaamde oopbrongemeenskap ("open-source community") en bevat bydraes deur verskeie persone. Die inherente probleem wat met enige oopbrongemeenskap gepaardgaan is dat die meeste van die dokumentasie nie noodwendig volwaardige akademiese werk is nie. Volgens Oates (2006:74) word handleidings normaalweg nie algemeen gebruik vir literatuurstudies nie. Die navorser is egter genoodsaak om hierdie bronne van MySQL te gebruik aangesien hierdie inligting die nuutste en akkuraatste bron van inligting oor die spesifieke MySQL-onderwerpe is.

3.3.1 Die ontstaan van MySQL

Volgens MySQL AB (2005:6) sou die databasisbestuurstelsel mSQL (Mini SQL is tans die eiendom van Hughes Technologies) aanvanklik saam met MySQL AB se vinnige laevlakroetines (ISAM) gebruik word. Na toetsing is gevind dat mSQL nie vinnig of buigbaar genoeg was nie. 'n Nuwe SQL-koppelvlak na die databasis wat baie dieselfde toepassingsprogrammeringskoppelvlak (API) as mSQL se koppelvlak besit, is geskep. Die koppelvlak is ontwerp om derdepartykode wat vir mSQL geskryf is maklik oor te dra na MySQL. Die oorspronklike bronkode het sy oorsprong in die vroeë tagtigerjare, en bied 'n stabiele kodebasis (MySQL AB, 2005:9).

Onsekerheid bestaan oor die ontstaan van die naam MySQL. Baie van MySQL AB se produkte sluit die voorvoegsel "my" in, en een van die stigters van MySQL AB, Michael "Monty" Widenius, se dogter se naam is ook "My". Die maatskappy MySQL AB bestaan uit MySQL-stigters en hoofprogrammeerders. MySQL AB is oorspronklik in Swede gestig deur David Axmark, Allan Larsson en Michael "Monty" Widenius. Die besigheid TcX (TcX DataKonsult AB, 2005) is die voorganger van MySQL AB (MySQL AB, 2005:9).

'n Kort lys van weergawes en die kenmerke wat bygevoeg is, word hier gegee:

- MySQL 3.23 – Vreemde sleutels (vir InnoDB-stoorenjin)
- MySQL 4.0 – "Unions"
- MySQL 4.1 – Subnavrae
- MySQL 4.2 – R-bome (vir MyISAM-stoorenjin)
- MySQL 5.0 – Gestoorde prosedures
- MySQL 5.0 – Skyntabelle
- MySQL 5.0 – "Cursors"
- MySQL 5.1 – Vreemde sleutels (alreeds geïmplementeer in 3.23 vir InnoDB)
- MySQL 5.0 en 5.1 – "Triggers"
- MySQL 5.1 – Volledige buiteverbinding ("Full outer join"), beperkings ("constraints"), verdeling ("partitioning") en ry-gebaseerde replikasie

3.3.2 'n Oorsig oor die elemente in MySQL en die wyses waarop MySQL sekere bewerkings hanteer

MySQL is in die programmeertale C en C++ geskryf en met verskeie vertalers getoets. MySQL is vir verskeie bedryfstelsels, soos byvoorbeeld Windows en Linux, beskikbaar. Die toepassingsprogrammeringskoppelvlak (API) is vrylik beskikbaar vir verskeie programmeertale,

naamlik C, C++, Eiffel, Java, Perl, PHP, Python, Ruby en Tel. Die ondersteuning vir volledige "multi-threaded threads" is ingebou en indien dit beskikbaar is, ondersteun MySQL meer as een SVE.

MySQL bied beide transaksiegerigte en nie-transaksiegerigte stoorenjins. Die moontlikheid bestaan om stoorenjins by te voeg. 'n Vinnige "thread"-gebaseerde geheue-allokeringstelsel bestaan en tydelike "hash"-tabelle (wat in die geheue pas) word toegeken. Verbindings ("joins") word behartig met 'n enkeldeurgang-multiverbinding ("single-sweep multi-join"). Vir die verbinding lees MySQL een ry van die eerste tabel, verkry dan 'n ooreenkomstige ry in die tweede tabel, en dan 'n ooreenkomstige ry in die volgende tabel totdat al die tabelle deursoek is. As al die tabelle verwerk is, word die geselekteerde kolomme van die ry vertoon. Die tabellys word van agter af gevolg, totdat 'n tabel verkry word wat nog ooreenkomstige rye bevat. Die volgende ry word van hierdie tabel gelees en die proses herhaal na die volgende tabel. As al die ooreenkomstige rye van die ry in die eerste tabel gevind is, word die volgende ry in die eerste tabel verwerk. SQL-funksies word geïmplementeer met 'n geoptimeerde klasbiblioteek ("class library").

Die MySQL-kode word met "Purify" ('n kommersiële geheuelekkasieverkryger) getoets en met "Valgrind", wat 'n GPL ("General public license") toepassing is. Die bediener is as 'n aparte program vir gebruik in 'n kliënt/bediener-netwerkomgewing beskikbaar, en as 'n biblioteek wat gekoppel kan word met toepassings.

MySQL bied verskeie buigsame en veilige regtetoekenningsmoontlikhede. Wagwoorde word met enkripsie beveilig.

MySQL kan groot databasisse met soveel as 64 indekse hanteer. Elke indeks mag tussen 1 en 16 kolomme of gedeelde kolomme bevat. Die maksimum indeks-wydte is 1000 grepe. Vanaf MySQL 4.1 ondersteun die Windows MySQL-weergawe se bedieners gedeeldegeheue-konneksies. MySQL verskaf ondersteuning vir toepassings wat ODBC-konneksies ("Open database connectivity") gebruik. Foutboodskappe kan in verskeie tale gegee word en sortering word volgens die huidige karakters-stel behartig. MySQL bied funksies om tabelle te ondersoek, te optimeer en te herstel.
MySQL 3.22 het 'n beperking van 4 gigagrepe ("gigabyte") op tabelgrootte gehad. Die MyISAM-stoorenjin ondersteun 'n tabel met 'n grootte van ongeveer 65536 terragrepe ("terrabyte"). Die InnoDB-stoorenjin stoor die InnoDB-tabelle in 'n tabelarea ("tablespace") wat uit verskeie lêers bestaan.

Van die kenmerke in MySQL 4.1 word beskryf deur MySQL AB (2005:18-20). MySQL 4.1 bevat funksionaliteit om subnavrae en afgeleide tabelle[18] te hanteer. B-boomindekse word vir "heap"-tabelle ondersteun. Die "NDB Cluster"-stoorenjin word deur MySQL Cluster ('n weergawe wat spesiaal vir groepering ("clustering") gebruik word) gebruik. Die "ARCHIVE"-stoorenjin word gebruik om groot hoeveelhede data sonder indekse te stoor. Die CSV-stoorenjin stoor data in 'n komma-verdeelde formaat. Die "BLACKHOLE"-stoorenjin stoor nie data nie, stuur 'n leë resultaat terug en word gebruik in replikasie.

MySQL ondersteun ANSI/ISO SQL-standaard mits daar geen negatiewe impakte op spoed en kwaliteit van die kode as gevolg van die standaard ontstaan nie (MySQL AB, 2005:29). Volgens MySQL AB (2005:34-36) ondersteun MySQL transaksies met die InnoDB- en BDB-transaksie-stoorenjins. MyISAM-tabelle ondersteun atomiese bywerkings wat altyd alle bywerkings bevestig ("commit"), en geen ander kliënt kan die bywerking beïnvloed nie. Die moontlikheid bestaan om verskillende stoorenjins vir elke tabel te gebruik. MySQL bied transaksievlakbetroubaarheid en -integriteit, selfs vir nie-transaksiegerigte tabelle. As die tabelle gesluit word met "LOCK TABLES", word alle bywerkings vertraag totdat integriteitstoetse gedoen is. As 'n "READ LOCAL"-slot verkry word op 'n tabel wat gelyktydige byvoegbewerkings aan die einde van die tabel toelaat, word leesinstruksies toegelaat, asook byvoeging deur ander gebruikers. Die rekords wat deur die ander gebruikers bygevoeg word, kan nie deur die gebruiker met die leesslot gesien word alvorens die slot vrygestel word nie. As 'n slot op die tabel bestaan, laat die "INSERT DELAYED"-bevel toe dat byvoeging in 'n waglys geplaas word sonder dat die gebruiker hoef te wag vir die byvoegbevel om te voltooi.

Die InnoDB-stoorenjin ondersteun vreemdesleutelbeperkings (MySQL AB, 2005:36-37). Die tipe beperkings laat toe dat verwyder- en bywerkbewerkings op 'n tabel, deur al die afhanklike tabelle van die betrokke tabel, gereflekteer word (met "CASCADE"). Een van die voordele van hierdie beperkings is dat die toepassings nie oor kennis van die afhanklike tabelle hoef te beskik nie. Die rekord word gewysig of verwyder en die databasis self hanteer die impak van die bewerking. Die beperkings het egter 'n koste ten opsigte van addisionele verwerking deur die databasisbestuurstelsel.

Die wyse waarop MySQL beperkings op data hanteer, word deur MySQL AB (2005:39-40) beskryf. Aangesien MySQL beide transaksiegerigte tabelle[19] en nie-transaksiegerigte tabelle hanteer, word die beperkings anders geïmplementeer as ander databasisbestuurstelsels.

[18] Subnavraag wat in die "FROM"-gedeelte volgens MySQL AB (2005:743) voorkom.

[19] Hanteer transaksie- en terugrolbewerkings.

MySQL probeer om 'n foutboodskap te verskaf vir enige fout wat opgespoor word tydens die ontleding ("parsing") van die uitdrukking wat uitgevoer moet word. MySQL poog om te herstel na elke fout wat tydens die uitvoeringsfase van die uitdrukking plaasvind. Hier volg MySQL se hantering van 'n paar beperkings:

- **Primêre sleutel en unieke indeksbeperking (MySQL AB, 2005:39)** – Normaalweg word 'n foutboodskap vertoon wanneer probeer word om 'n ry in te voeg waarvan die bepaalde sleutel reeds bestaan.

  Vir 'n transaksiegerigte stoorenjin soos InnoDB, sal MySQL outomaties die transaksie terugrol. MySQL stop die verwerking van die uitdrukking by die ry waar die fout voorkom vir 'n nie-transaksiegerigte stoorenjin, en die oorblywende rye word nie verwerk nie. Die moontlikheid bestaan om die sleuteloortredings te ignoreer en verwerking te laat voortgaan.

- **Beperkings op ongeldige data (MySQL AB, 2005:39)** – Voor MySQL 5.0.2 laat MySQL ongeldige data toe deur die data in geldige data te omskep. Vanaf MySQL 5.0.2 bestaan die opsie dat navrae wat ongeldige data in kolomme besit, verwerp kan word. Die omskakeling verkry die beste moontlike waarde en vervang die ongeldige waarde. Die omskakeling gebeur as gevolg van sommige tabelle wat nie terugrolbewerkings ondersteun nie. Die navrae vir so 'n tabel sal net gedeeltelik verwerk word, wat 'n ongewenste toestand is.

  'n Lys van hierdie omskakelings van foutiewe data word hier gegee:

  - MySQL stoor 'n nul, die kleinste of die grootste toelaatbare getal as die waarde buite die toelaatbare grense van 'n numeriese kolom is.
  - MySQL stoor 'n leë string, of soveel van die karakterstring as wat in die kolom kan kom, as die karakterstring foutief is.
  - MySQL stoor 0 indien 'n string wat nie met 'n syfer begin nie in 'n numeriese kolom gestoor moet word.
  - MySQL laat foutiewe datumwaardes soos byvoorbeeld "2000-02-31" toe en stoor "0000-00-00" as die datum glad nie in 'n herkenbare formaat is nie.
  - Die poging om 'n "NULL"-waarde in 'n kolom te stoor wat nie "NULL"-waardes toelaat nie, veroorsaak 'n foutboodskap vir net een byvoegbewerking en stoor verstekwaardes in die kolom as meer as een byvoegbewerking vereis word.
  - As 'n "INSERT"-uitdrukking nie 'n waarde vir 'n kolom spesifiseer nie, word die verstekwaarde vir die kolom gestoor.

- **"ENUM"- en "SET"-beperkings (MySQL AB 2005:40)** – Die twee beperkings verskaf 'n benadering om 'n kolom te dwing om net waardes uit 'n sekere voorafbepaalde stel waardes te kan kry. As 'n ongeldige waarde in 'n kolom met 'n "ENUM"-opstelling ingevoeg word, word die waarde 0 (leë string vir karakterstringe) ingevoeg. Die "SET"-tipe ignoreer die ongeldige waarde.

Die datatipes van die kolomme wat ondersoek word, stem ooreen met die kolomme van SQL Server (sien paragraaf 3.2.5.1 en 3.2.5.2). Die *varchar*-tipe word gestoor met 1 ekstra greep wat die lengte van die kolom aandui.

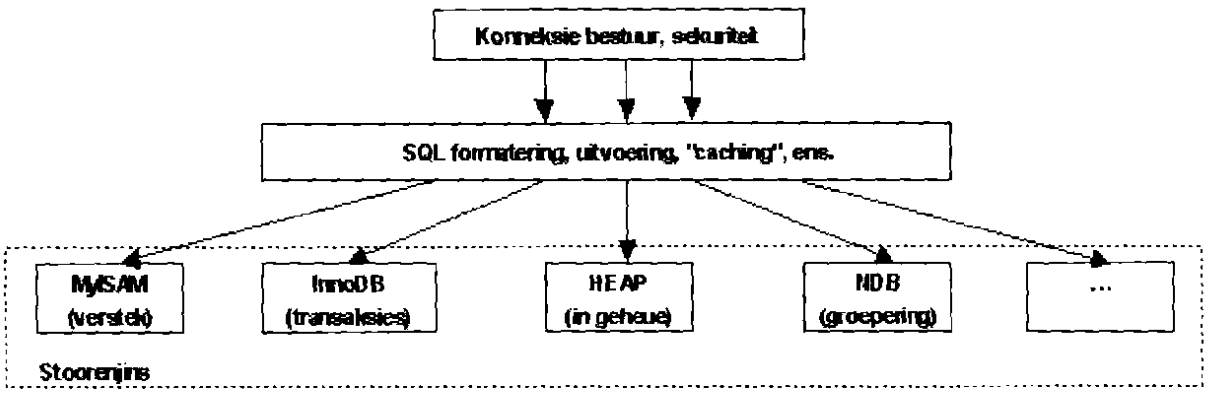
3.3.3 Die interne argitektuur van MySQL 4.1

Die interne argitektuur word aan die hand van Balling en Zawodny (2004) se boek en MySQL AB (2005) se PDF-lêer beskryf.

Volgens Balling en Zawodny (2004:16) is een van die kragtige aspekte wat MySQL van ongeveer al die ander databasisbestuurstelsels onderskei, die aantal keuses en opsies wat MySQL vir die gebruikersomgewing bied. Die verstekkonfigurasie kan aangepas word vir 'n wye verskeidenheid hardeware. Verskeie datatipes is beskikbaar vir die skep van tabelle en die tipe van die tabel self kan gekies word. Dit is moontlik om verskeie tipes tabelle in dieselfde databasis te huisves. Stoorenjins word gebruik om hierdie tipes te hanteer.

Figuur 3.9 toon 'n vereenvoudigde logiese voorstelling van die MySQL-argitektuur, gebaseer op Balling en Zawodny (2004:16).

Figuur 3.9 – 'n Logiese voorstelling van MySQL se argitektuur.



Die boonste vlak in figuur 3.9 bevat dienste wat nie uniek aan MySQL is nie. Hierdie vlak besit netwerkgebaseerde kliënt- of bedienerhardeware, soos die hantering van konneksies en

sekuriteit. Die middelste vlak in figuur 3.9 bevat elemente wat navraagontleding en -analise, kasberging en ingeboude funksies soos wiskundige funksies en datumfunksies hanteer. Die vlak bevat funksionaliteit wat oor verskillende stoorenjins voorkom. Gestoorde prosedures sal vanaf MySQL 5.0 in die middelste vlak voorkom.

Die onderste vlak bestaan uit stoorenjins. Die stoorenjins is verantwoordelik vir die stoor van data in MySQL en elkeen besit voor- en nadele. Die koppelvlak tussen die middelste en onderste vlak is 'n toepassingsprogrammeringskoppelvlak (API) wat nie uniek aan enige stoorenjin is nie, maar eerder met elemente van elke stoorenjin kan kommunikeer.

Die toepassingsprogrammeringskoppelvlak bevat ongeveer 20 laevlakfunksies wat bewerkings soos die begin van 'n transaksie of die verkryging van 'n ry met 'n spesifieke primêre sleutel behartig. Die stoorenjins hanteer nie SQL of kommunikasie met mekaar nie, maar reageer net op versoeke vanaf die hoër vlakke in MySQL.

Die kwessie van gelyklopendheid en sluiting word aan die hand van Balling en Zawodny (2004:18-22) beskryf. Die tipe slotte wat voorkom, stem baie ooreen met SQL Server se slotte, naamlik eksklusiewe slotte en gedeelde slotte. Die grein van die slotte is ook dieselfde as SQL Server se grein, naamlik tabelslotte, bladsyslotte en ry-slotte.

'n Tegniek genaamd multiweergawe-gelyklopendheidbeheer ("multi-version concurrency control (MVCC)") bestaan wat soortgelyk aan ry-slotte is en in die InnoDB-stoorenjin (Balling & Zawodny, 2004:20-22) beskikbaar is. Die tegniek laat nie-sluitende lees-aksies toe, terwyl die nodige rekords steeds gesluit word vir skryf-aksies. Die navraag sal 'n momentopname ("snapshot") van die data sien soos wat die data aan die begin van die navraag was, ongeag die uitvoertyd. In 'n stelsel wat van hierdie tegniek gebruik maak, word twee bykomende verskuilde waardes, wat die skeptyd en verwydertyd voorstel, saam met elke ry gestoor. Die waardes is egter nie 'n tydwaarde nie, maar eerder 'n weergawe-waarde. Die weergawe vermeerder elke keer wat 'n transaksie begin.

Die selekteerbevel verkry alle rye waarvan die skeptyd kleiner as of gelyk aan die huidige stelselweergawe-waarde is en alle rye waarvan die verwydertydweergawe "NULL" of groter as die huidige stelselweergawe is. Die skeptyd van die ry mag nie in die lys van lopende navrae vir die selekteerbevel wees nie. Die byvoegbevel stoor die huidige stelselweergawe-waarde saam met die nuwe ry as die skeptyd. Die verwydering van 'n ry veroorsaak dat die huidige weergawe-waarde in die verwydertyd-kolom van die ry gestoor word. As 'n ry gewysig word, word die veranderde ry met die huidige weergawe-waarde as die skeptyd gestoor en die weergawe-nommer word gestoor as die ou ry se verwydertyd. Die voordeel van hierdie tegniek

is dat navrae geen slotte benodig nie, en die nadele is dat die bediener meer data moet stoor en dat verwerking van die rye vermeerder.

Balling en Zawodny (2004:22-29) se gedeelte oor transaksies word kortliks hier beskryf. Transaksies moet voldoen aan vier kriteria om sogenaamde AKID-transaksies (meer bekend as "ACID transactions") te wees (sien ook Atzeni *et al.* 1999:285):

- Atomisiteit ("Atomicity") – Transaksies moet as een onskeibare eenheid van werk funksioneer. Die hele transaksie word aanvaar of die hele transaksie word teruggerol.
- Konsekwentheid ("Consistency") – Aangesien foutiewe transaksies nie bevestig ("commit") word nie, gaan die transaksie se foutiewe veranderings nie in die databasis geplaas word nie en bly die databasis konsekwent.
- Isolasie ("Isolation") – Die resultaat van transaksies moet normaalweg nie sigbaar vir ander transaksies wees nie totdat die transaksie voltooi word.
- Duursaamheid ("Durability") – As die transaksie bevestig is, moet die data gestoor wees sodat data nie verlore gaan as die stelsel afgaan nie.

Hierdie tipe transaksie vereis meer werk van die databasisbediener. MySQL verseker met die stoorenjins dat net sekere tabelle kan spesialiseer in hierdie tipe transaksies, en die res kan normaal hanteer word. Die transaksie-isolasievlakke van MySQL stem ooreen met SQL Server se isolasievlakke, wat in paragraaf 3.2.3.6.1 verduidelik word.

MySQL kan opsioneel van transaksielogboeke gebruik maak, wat eers bywerkings van tabelle stoor om later die bywerkings aan die tabel self te maak. Die log bevorder die duursaamheid van die transaksies, aangesien die veranderings in die transaksielog behou word as die stelsel sou afgaan. MySQL bevestig elke navraag soos die navraag uitgevoer word aangesien die verstekwaarde elke navraag as 'n enkele transaksie beskou.

Die keuse van die regte stoorenjin is vir elke tabel belangrik. Elke tabel se funksie gaan bepaal watter tipe stoorenjin die beste vir die tabel sal pas. Balling en Zawodny (2004:29-34) bied 'n paar riglyne oor die keuse:

- Transaksies en gelyklopendheid – Die eerste faktor wat die keuse van 'n stoorenjin beïnvloed, is transaksies en gelyklopendheid. As die toepassing transaksies met 'n hoë lees- of skryf-gelyklopendheid benodig, word InnoDB voorgestel. As die toepassing transaksies met 'n middelmatige lees/skryf-gelyklopendheid benodig, kan óf BDB- óf InnoDB-tabelle gebruik word. MyISAM-tabelle word normaalweg gebruik as transaksies onbelangrik is.

- Rugsteun – Die tweede faktor wat die keuse kan beïnvloed, het te make met rugsteun, en word deur die gedeelte van Balling en Zawodny (2004:189-190) verduidelik. ISAM- en MyISAM-tabelle moet gesluit word om 'n konsekwente rugsteun te gee. InnoDB se datalêers bevat 'n groot hoeveelheid ongebruikte spasie en die rugsteun is kleiner as die databasis se grootte. Die kopieer van die datalêers is 'n baie effektiewe tegniek vir rugsteun en kan vinniger wees as die gebruik van "mysqldump" wat al die data as SQL-uitdrukkings na 'n lêer toe uitskryf. ISAM- en MyISAM-tabelle moet die data-, indeks- en tabeldefinisielêers kopieer vir rugsteun. BDB- en InnoDB-tabelle moet transaksielogboeke en data kopieer vir rugsteun. InnoDB besit rugsteuntoerusting wat rugsteun kan doen al is die bediener af of die tabel gesluit vir veranderings. InnoDB vereis dat die stelsel af is voor die tabelarealêers herstel word. ISAM en MyISAM word net gekopieer om die tabelle te herstel.
- Funksies van elke stoorenjin – Nog 'n faktor is die funksies wat elke stoorenjin aan die gebruiker bied. As die toepassing akkurate en vinnige ry-tellings benodig, is MyISAM die beste enjin. InnoDB moet die rye tel, terwyl MyISAM die presiese aantal rye stoor. As vreemde sleutels benodig word, is InnoDB die enigste stoorenjin wat die sleutels ondersteun.
- Tipe navraag – As die tipe navraag 'n groot aantal byvoegings met spoed as primêre doelwit behels, sal MyISAM aanvanklik die beste opsie wees aangesien die enjin baie byvoegings teen 'n relatief lae koste kan hanteer. As opsommings op die byvoegings toegepas word, gaan MyISAM egter nie so effektief wees nie. As daar op die tabel baie meer leesbewerkings as bywerkings geskied, sal MyISAM weereens 'n goeie opsie wees. As die integriteit van die data belangrik is, moet 'n transaksiegerigte stoorenjin soos InnoDB of BDB oorweeg word.

Die tipe stoorenjins word in die volgende paragraaf ondersoek.

3.3.3.1 Stoorenjins

Elke tabel kan 'n spesifieke stoorenjin gebruik. Die stoorenjin is afhanklik van die funksie van die tabel. Die stoorenjin bepaal die wyse waarop die data gestoor word. Die "SHOW TABLE STATUS"-bevel word gebruik om (saam met verskeie ander inligting) die stoorenjintipe van 'n tabel te vertoon. 'n Beskrywing van die verskillende stoorenjins volg aan die hand van Balling en Zawodny (2004:35-44), MySQL AB (2005) se PDF-lêer, Kruckenberg en Pipes (2005:153-188) en figuur 3.9.

3.3.3.1.1 ISAM-stoorenjin

Die stoorenjin word volgens MySQL AB (2005:843-844) bespreek. Die stoorenjin is die oorspronklike stoorenjin wat MySQL gebruik het, en ook die enigste stoorenjin tot en met MySQL 3.23. Elke ISAM-tabel word in drie lêers gestoor. Die lêername van hierdie drie lêers bestaan uit die tabel se naam, en die uitbreiding ("extension") toon die tipe van die lêer aan. Die ".frm"-tipe stoor die tabel se definisie. Die ".ISD"-tipe stoor die data, en die indekslêer word met 'n ".ISM"-uitbreiding gestoor. ISAM gebruik B-boomindekse. Die stoorenjin mag slegs 16 indekse per tabel bevat en elke sleutel mag hoogstens 256 grepe groot wees. Die data word in 'n bedryfstelselafhanklike formaat gestoor, wat redelik vinnig is. ISAM kan nie tabelle groter as 4 gigagrepe hanteer nie. "MERGE"-tabelle word nie ondersteun nie. Die ISAM-stoorenjin is nog in MySQL 3.23 – MySQL 4.1 beskikbaar, maar spesiale metodes moet toegepas word om die stoorenjin te kan gebruik.

3.3.3.1.2 MyISAM-stoorenjin

Die stoorenjin het ISAM vanaf MySQL 3.23 as die verstekstoorenjin vervang. MyISAM word in twee lêers gestoor. Die datalêer bevat die uitbreiding ".MYD" en die indekslêer bevat die uitbreiding ".MYI". Die MyISAM-formaat is platform-onafhanklik en die lêers kan sonder probleme vanaf Intel-bedieners na Macintosh Powerbook of Sun SPARC gekopieer word.

MyISAM stoor rye as dinamies of staties (vaste lengte), wat tydens die skep van die tabel bepaal word. Die grootte van die tabel is afhanklik van die beskikbare skyfspasie en die grootste enkele lêer wat die bedryfstelsel kan skep. Die verstekwaarde vir die MyISAM-lêers is gestel op 4 gigagrepe, maar dit kan verstel word na die verlangde grootte.

MyISAM hanteer slotte op die tabelvlak. 'n Opsie ("--myisam-recover") bestaan waarmee MySQL die MyISAM-tabel toets om te sien of die tabel behoorlik toegemaak is. As die tabel weens byvoorbeeld hardewareprobleme nie behoorlik toegemaak is nie, sal MySQL die tabel deursoek vir foute, en die foute herstel. Die toepassing moet tydens die foutopsporing- en -herstelproses wag. Die foutopsporing- en -herstelproses kan met die hand geloop word deur die bevele "CHECK TABLE" en "REPAIR TABLE" te gebruik. As MyISAM-tabelle geen afgeleë rye besit nie, mag byvoegbewerkings plaasvind terwyl navrae op die tabel uitgevoer word. MyISAM laat indeksering van kolomme toe wat "NULL" bevat. MyISAM-tabelle wat met die opsie DELAY_KEY_WRITE geskep is, verhoog produktiwiteit in stelsels wat baie tabelveranderings behartig. Die indeksveranderings word nie dadelik na die skyf geskryf nie, maar word in die geheue-buffer gehou wanneer die tabel toegemaak word of die sleutelbuffer verwerk word.

MyISAM-tabelle kan saamgedruk ("compress") word om die spasie te verminder en navrae is gevolglik vinniger. Kruckenberg en Pipes (2005:158) stel dat veranderings nie op die data kan plaasvind nie. Die verkryging van die saamgedrukte rekords se koste is weglaatbaar klein, aangesien slegs die ry wat gesoek word weer verwerk moet word om die kompressie te verwyder.

MyISAM RAID[20]-tabelle is spesiale tabelle wat spesiaal vir gebruik vertaal moet word. Die tabelle is nie eintlik 'n aparte tabeltipe nie, maar eerder net 'n uitbreiding op MyISAM waar die datalêer in verskeie lêers verdeel is. Skryf-aksies na die tabel word tussen die verskillende datalêers verdeel. Die uitbreiding is nuttig in gevalle waar die bedryfstelsel nie groot lêers kan hanteer nie. Die indekslêer word egter nie verdeel nie, en die grootte van die lêer moet gemonitor word.

MySQL AB (2005:841-850) en Kruckenberg en Pipes (2005:161) stel dat MyISAM-tabelle 64 indekse vir die huidige stabiele weergawe van MySQL kan besit. Die maksimum aantal kolomme per indeks is 16. Die maksimum sleutelgrootte is 1000 grepe. Die indekslêer van MyISAM is kleiner as ISAM-tabelle se indekslêer. As die tabel nie oop spasie in die middel van die datalêer bevat nie, kan byvoegbewerkings gelyktydig saam met leesbewerkings plaasvind. As daar egter spasie oop is as gevolg van die verwydering van rye of die bywerking van veranderlikelengtekolomme, is die byvoeging nie gelyklopend met leesbewerkings nie.

MySQL AB (2005:824) verduidelik dat MyISAM-tabelle van B-boomindekse (ook R-boom en "FULLTEXT" volgens Kruckenberg en Pipes (2005:161)) gebruik maak en dat die indekslêer se grootte benader kan word as *(sleutel_lengte + 4) / 0.67* vir elke sleutel en dan gesommeer word oor al die sleutels vir die totale grootte. Die berekening dui die slegste geval van al die sleutels aan wat in gesorteerde volgorde bygevoeg is, en 'n tabel wat nie kompressie van sleutels ondersteun nie. MyISAM ondersteun ook 'n mate van kompressie vir karakterstringkolomme in indekse en kan selfs 'n gedeeltelike kompressie gebruik as die karakterstringkolom 'n deel van die indeks uitmaak.

Drie tipes stoorformate word volgens MySQL AB (2005:824-827) gebruik om MyISAM-tabeltipes te stoor, naamlik vaste formaat, dinamiese formaat en saamgedrukte formaat. Die verstekwaarde is die vaste of statiese formaat en word gebruik vir tabelle wat geen veranderlikelengtekolomme bevat nie. Elke ry word met 'n vaste aantal grepe gestoor. Hierdie formaat is die eenvoudigste, die veiligste en die vinnigste van die drie formate. Die formaat ondersteun effektiewe herstel na 'n faling van die bediener deurdat al die rekords behalwe die gedeeltelik geskryfde een herstel kan word.

[20] Normaalweg "Redundant Array of Inexpensive Disks" vir hardeskywe wat dui op verskeie oortollige hardeskywe.

Die dinamiese stoorformaat word gebruik vir MyISAM-tabelle wat veranderlikelengtekolomme bevat. Elke ry bevat 'n opskrif wat die lengte van die ry aandui. 'n Rekord kan moontlik oor 'n paar databladsye as gevolg van lengte versprei word. Elke rekord word voorafgegaan deur 'n "bitmap" wat aandui of die kolom 'n leë string (vir karakterstringe) of nul (vir numeriese kolomme) bevat. Die formaat benodig minder spasie as die vaste formaat.

Die laaste formaat is 'n saamgedrukte formaat waarvan die tabelle se data nie verander mag word nie. Die tabelle neem baie min spasie in beslag. Die formaat kan vastelengte- of dinamieselengterekords hanteer.

3.3.3.1.3 MyISAM Merge-stoorenjin

Die ineengevlegte tabel word gevorm deur verskeie tabelle te kombineer in een virtuele tabel. Gestel verskeie MyISAM-tabelle is geskep. Dit is moontlik om 'n MyISAM Merge-tabel te skep wat navrae kombineer wat oor die verskillende MyISAM-tabelle strek. Die ineengevlegte tabel word hanteer soos 'n kombinasie van al die tabelle waarop die enkele tabel toegespits is. Die MyISAM-tabelle moet net dieselfde definisie besit, en vanaf MySQL 4.1.1 af kan verskeie databasisse gebruik word. Die MyISAM-tabelle kan saamgedrukte tabelle wees, wat beteken dat ouer tabelle saamgedruk kan word omdat daar in elk geval nie na die ou tabelle geskryf word nie.

3.3.3.1.4 BDB-stoorenjin

Balling & Zawodny (2004:44) stel dat MySQL se eerste transaksiegerigte stoorenjin, BDB, gebaseer is op die Berkeley DB-databasisbiblioteek wat ondersteun word deur Sleepycat Software. BDB-tabelle gebruik bladsyvlaksluiting om hoër gelyklopendheid as MyISAM-tabelle te kry.

MySQL AB (2005:836-838) dui aan dat die BDB-tabelle op twee lêers gestoor word. Die lêername bestaan uit die tabel se naam, en die lêers met uitbreidings ".frm" en ".db" bevat onderskeidelik die tabeldefinisies en die tabeldata saam met die indeksdata. Die tabelle kan soveel as 31 indekse per tabel, met 16 kolomme per indeks, hanteer. Die maksimum sleutellengte is 1024 grepe. 'n Primêre sleutel word vir elke ry benodig en word geskep met grootte van 5 grepe indien die sleutel nie bestaan nie. Reekssoektogte vir die BDB-stoorenjin is stadiger as MyISAM se reekssoektogte omdat data in die BDB-tabel in B-bome gestoor word en nie in 'n aparte datalêer nie. Loglêers word onderhou om terugroltransaksies te ondersteun. BDB-tabelle stoor die absolute pad van die datalêer in die datalêer self, wat veroorsaak dat die

lêer nie van een databasis na 'n ander geskuif kan word nie, tensy die absolute pad dieselfde vir beide is nie.

3.3.3.1.5 InnoDB-stoorenjin

Hierdie stoorenjin word volgens Balling en Zawodny (2004:42-44) en gedeeltes uit MySQL AB (2005:845-901) beskryf. Die stoorenjin is deur Heikki Tuuri van Innobase Oy in Helsinki, Finland ontwikkel. Die stoorenjin is transaksiegerig en is op Oracle gebaseer. Kruckenberg en Pipes (2005:166) stel dat hierdie stoorenjin die beste keuse is, indien AKID-transaksies (sien paragraaf 3.3.3) belangrik is.

Die tabelle se data word in 'n reeks van een of meer datalêers gestoor wat saam as 'n tabel-area bekend staan. InnoDB gebruik multiweergawe-gelyklopendheidbeheer (sien paragraaf 3.3.3) om 'n baie hoë vlak van gelyklopendheid te verkry en die verstekvlak vir isolasie is die herhaalbarelees-vlak (sien paragraaf 3.2.3.6.1). Een van die grooste kenmerke van InnoDB is die gebruik van vreemdesleutelbeperkings.

Volgens MySQL AB (2005:845) hanteer InnoDB sluiting op die ry-vlak. SQL-navrae mag InnoDB-tabelle met ander tipe tabelle meng. InnoDB kan groot hoeveelhede data hanteer, en 'n aparte bufferarea word in die hoofgeheue vir die kasgeheue van data en indekse onderhou. MySQL AB (2005:871-872) stel dat InnoDB multigrein-sluitbewerkings ondersteun wat gelyktydig beide rekordslotte en tabelslotte toelaat. 'n Nuwe slot, naamlik intensieslotte, word saam met gedeelde en eksklusiewe slotte gebruik. Die transaksie dui aan watter tipe slot (gedeel of eksklusief) tot 'n spesifieke ry in die tabel verlang word; die transaksie dui met ander woorde 'n intensie aan.

3.3.3.1.6 "Heap"-stoorenjin

Die stoorenjin word aan die hand van MySQL AB (2005:832-834) bespreek. Die tabelle word in die geheue gestoor. Elke geheue-tabel word geassosieer met een lêer waarvan die lêernaam met die tabelnaam ooreenstem en waarvan die uitbreiding ".frm" is. Die lêer dui die tabeldefinisie aan. Die tabelle gebruik normaalweg "hash"-indekse (verstekwaarde), wat vinnig is, maar B-boomindeks is ook moontlik. Die tabelle is normaalweg tydelike tabelle en die inligting gaan verlore indien die bediener afgaan. Die spasie vir die tabelle word in klein blokke geallokeer en gebruik dinamiese "hashing" vir byvoeging. Geen oorvloei-areas en geen ekstra ruimte word vir sleutels of vry-lyste benodig nie. Afgeleë rye word in 'n geskakelde lys geplaas en die spasie word weer gebruik wanneer nuwe data bygevoeg word. Die tabelle laat 32 indekse en 16 kolomme per indeks toe. Die tabelle maak gebruik van die vastelengteformaat

en kan nie-unieke sleutels bevat. Die geheue-tabelle word tussen alle kliënte van MySQL gedeel. Die "heap"-tabelle word nie na die hardeskyf geskryf nie en voorsorg moet getref word dat hierdie tabelle nie te groot raak nie.

3.3.3.1.7 ARCHIVE-stoorenjin

Hierdie stoorenjin word volgens MySQL AB (2005:840-841) bespreek. Die stoorenjin word gebruik om groot hoeveelhede data sonder indekse en met 'n klein verwerkingskoste te stoor. Die tabelle word op drie verskillende lêers gestoor waarvan almal dieselfde lêernaam (stem ooreen met tabelnaam) het en onderskeidelik uitbreidings ".frm" (tabeldefinisie), ".ARZ" (datalêer) en ".ARM" (metadatalêer) besit. Die enjin ondersteun net seleksie- en byvoegbewerkings. Sortering word wel hanteer en ry-vlakslotte word gebruik. Rekords word saamgedruk ("compress") soos wat die rekords bygevoeg word. 'n Seleksiebewerking maak slegs van tabeldeursoeke gebruik, aangesien geen indekse gebruik word nie.

3.3.3.1.8 BLACKHOLE-stoorenjin

MySQL AB (2005:841-843) stel dat hierdie stoorenjin data aanvaar, maar nie die data stoor nie. Die enigste lêer wat geskep word, is die tabeldefinisielêer, met tabelnaam as die lêernaam en uitbreiding ".frm". Alle tipes indekse word op hierdie tipe tabel toegelaat. Byvoegings word nie gestoor nie, maar as die binêre log geaktiveer is, word die SQL-uitdrukkings gelog. Die tipe stoorenjin kan gebruik word om al die binêre logdata na 'n slaafbediener oor te dra. Metings kan geneem word om die koste van binêre logprosedures te meet, en ook van werkverrigtingsprobleme wat nie verwant aan die stoorenjin is nie.

3.3.3.1.9 CSV-stoorenjin

Hierdie stoorenjin word volgens MySQL AB (2005:841) bespreek. Die enjin stoor data in tekslêers met 'n komma-verdeelde formaat. Twee lêers met lêername gelyk aan die tabelnaam word met uitbreidings ".frm" (tabeldefinisie) en ".CSV" (datalêer) geskep. Die tipe lêer kan byvoorbeeld in Microsoft Excel ingetrek word vir verwerking.

3.3.3.1.10 EXAMPLE-stoorenjin

Die stoorenjin word volgens MySQL AB (2005:839-840) bespreek. Die stoorenjin word gebruik as 'n voorbeeld vir die skryf van nuwe stoorenjins en het geen verdere doel nie.

3.3.4 MySQL-indekse

Hierdie gedeelte word met die hulp van Balling en Zawodny (2004:61-78) beskryf. Die basiese beginsel van MySQL se indekse stem baie ooreen met SQL Server se indekse. Indekse is 'n manier om data-opsporing te versnel en verruil stoorspasie vir produktiwiteit. MySQL verskaf 'n tegniek waar gedeeltelike indekse op kolomme geskep kan word om sodoende die gebruik van spasie te beperk. MySQL se indekse kan, soos in die geval van die meeste relasionele databasisbestuurstelsels, uit verskeie kolomme bestaan. Een van die belangrikste aspekte van die manier hoe MySQL indekse hanteer, is dat MySQL net een indeks per tabel per navraag gebruik, al is daar meer indekse beskikbaar om te kan gebruik. Die enigste uitsondering op die reël is "UNION", wat in elk geval twee aparte navrae is wat saamgevoeg word. MySQL optimeer self gevalle waar sortering swak produktiwiteit lewer. MySQL kan indekse van agter na voor deursoek om beter werkverrigting vir navrae, wat sortering toepas, te lewer (byvoorbeeld "DESC"-argumente).

Unieke indekse word gebruik om duplikate uit te skakel. MySQL laat toe dat "NULL"-waardes in 'n unieke indeks gestoor word sonder om die werkverrigting van die indeks te beïnvloed, maar die "NULL"-waarde mag net een keer as 'n primêre sleutel volgens die SQL-standaard voorkom. MyISAM se unieke indeks en primêre sleutels stem ooreen, maar die primêre sleutel mag nie "NULL"-waardes bevat nie. InnoDB en BDB vereis primêre sleutels vir elke tabel, en MySQL sal 'n onsigbare primêre sleutel (teller wat vermeerder) byvoeg indien een nie bestaan nie.

MySQL maak van gebondelde (slegs vir InnoDB-stoorenjin) en ontbondelde indekse (soms sekondêre indekse genoem) gebruik. MyISAM-tabelle stoor die indekslêer apart van die datalêer en die indekslêer bestaan uit 'n lys van primêre sleutels en wysers na rekords. MyISAM stoor die indekse op hierdie wyse aangesien die rekords in willekeurige volgorde is. InnoDB maak van gebondelde indekse gebruik waar die data-elemente deel uitmaak van die indeks. MySQL AB (2005:887) stel dat die InnoDB-tabel outomaties 'n gebondelde indeks aan die primêre sleutel toeken – of 'n unieke indeks met geen "NULL"-waardes nie indien die primêre sleutel nie bestaan nie. As nie een van die twee indekse bestaan nie, word die gebondelde indeks aan die onsigbare primêre sleutel toegeken.

MySQL maak van drie tipes indekse, naamlik B-boom, "Hash"- en R-boomindekse gebruik. Die B-boom en "Hash"-indeks is vergelykbaar met die ekwivalente indekse in SQL Server (sien paragraaf 3.2.6). Die B-boom is die verstekindeks in MySQL. 'n Algemene "hash"-funksie wat deur MySQL gebruik word, is die MD5-enkripsie wat 'n 128 bis-waarde verskaf. Die funksie verskaf $3.4 * 10^{38}$ moontlike waardes, en baie min rekenaars besit genoeg spasie om soveel gleuwe te hanteer. 'n Algemene tegniek bestaan waar die aantal gleuwe vooraf gekies word,

soos byvoorbeeld 35149 wat 'n priemgetal is. Die resultaat van die "hash"-funksie word deur die priemgetal gedeel en die res bepaal die gleuf.

Die ander indeks, R-bome, is die eerste keer in MySQL 4.1 gebruik. MySQL verkry die *n-dimensionele* objek se kleinste begrensde boks wat die objek kan bevat en stoor die koördinate van die reghoek of boks. Die reghoek word gebruik om objekte of figure in 'n spesifieke area te verkry.

Die MySQL-indekse se implementerings verskil effens vir die verskillende stoorenjins. Die impak van 'n paar stoorenjins ten opsigte van indekse word hier bespreek:

- MyISAM-tabelle – Die stoorenjin maak van B-boomindekse en R-boomindekse gebruik. MyISAM verskaf twee addisionele kenmerke tot B-boomindekse, naamlik voorvoegselkompressie ("prefix compression") en saamgedrukte ("packed") sleutels. Die voorvoegselkompressie word gebruik om algemene voorvoegsels in string sleutels uit te skakel. As voorbeeld sal MySQL-tabelle wat 'n "URL" ("Uniform Resource Locater") of internetadres moet stoor, se eerste gedeelte wat hoofsaaklik "http://" of "https://" is, saamdruk en dan gestoor word.

  Saamgedrukte sleutels ("packed keys") dui op 'n kompressie van heelgetalsleutels. Die boonste grepe (ten opsigte van bisse) van die heelgetalsleutel word eerste gestoor en aangesien groot hoeveelhede sleutels se boonste bisse nie baie verander nie, kan hierdie metode gebruik word.

  Volteksindeks ("fulltext") is 'n spesiale indeks vir MyISAM-tabelle wat die posisie van elke unieke woord in 'n kolom verkry. Die indeks word geskep deur 'n normale MyISAM B-boomindeks met twee kolomme te skep, waar die eerste kolom 'n veranderlikelengtestring is en die tweede kolom 'n reële getal bevat. Die eerste kolom bevat die geïndekseerde woord en die tweede kolom bevat die lokale gewig van die woord in die ry.

- "Heap"-tabelle – Die geheue-tabel kon oorspronklik net "hash"-indekse gebruik, maar MySQL besit die funksionaliteit dat B-boomindekse in die tipe tabelle gebruik mag word.

- BDB-tabelle – Slegs B-boomindekse word ondersteun. Voorvoegselkompressie word ondersteun en gebondelde indekse mag gebruik word. Die BDB-tabelle benodig 'n primêre sleutel.

- InnoDB-tabelle – B-boomindekse word ondersteun met geen voorvoegselkompressie of saamgedrukte sleutels nie. 'n Primêre sleutel is verpligtend en MySQL verskaf 'n 64 bis-waarde as die primêre sleutel nie bestaan nie. Die indeks word in die InnoDB-tabel-area gestoor. Gebondelde indekse word wel toeglaat, en MyISAM-tabelle bied die opsie om indeksdata se skryf na die skyf per tabel of vir die databasis te vertraag. Dit verhoog produktiwiteit, maar die indeks se data stem dalk nie meer presies ooreen met die werklike data nie.

Gevalle bestaan waar MySQL weier om indekse te gebruik. 'n Paar van hierdie gevalle word bespreek (Balling & Zawodny, 2004:75-76):

- "Wildcard"-vergelykings – 'n Navraag wat alle rekords versoek wat 'n spesifieke woord bevat, vereis dat MySQL elke ry in die tabel deurgaan. Die oplossing is die volteksindeks. Gevalle waar net gedeeltelike woorde gesoek word, sal nie van die volteksindeks gebruik kan maak nie aangesien die indeks net volledige woorde hanteer.
- "Regular expressions" – MySQL AB (2005:1648-1651) verduidelik "regular expressions" as 'n benadering om 'n patroon vir 'n komplekse soektog te spesifiseer. Geen indeks kan hierdie metode ondersteun nie en geen optimering word op hierdie tipe navrae toegepas nie. Die enigste wyse wat oorbly, is 'n tabeldeursoek.
- Foutiewe statistiese inligting – As die statistiese inligting van MySQL foutief is, mag indekse dalk nie gebruik word nie en normale tabeldeursoeke word toegepas.
- Aantal rye oorskry 'n sekere perk – Volgens MySQL AB (2005:436) baseer MySQL die besluit om eerder 'n tabelsoektog te doen op die vooruitskatting van rye (wat as resultaat verkry word), die tabelgrootte en die lees/skryf-blok se grootte.

3.3.5 MySQL-optimering

MySQL AB (2005:421) stel dat die ontwerp van 'n stelsel bepaal hoe vinnig die stelsel is. Dit is belangrik om te weet waar die moontlike probleemareas kan voorkom. 'n Paar algemene probleemareas word genoem:

- Hardeskyfsoektogte – Die data moet gevind word.
- Hardeskyf-lees- of -skryf-aksies – Tyd word gebruik om data te skryf of te lees.
- SVE-siklusse – Die data moet verwerk word om die korrekte resultaat te verkry.
- Geheue-bandwydte – Data word vanaf die skyf na die geheue gelaai om verwerk te word, en meer bandwydte sal meer data kan dra.

Die optimering van MySQL word aan die hand van die boek deur Balling en Zawodny en die webbladsye van MySQL AB (2005) verduidelik.

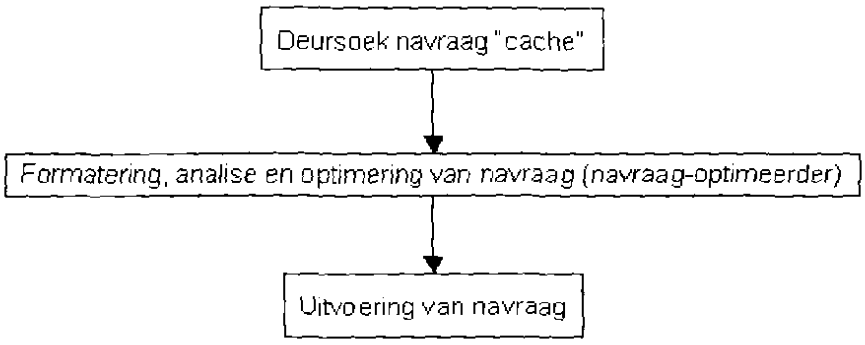
3.3.5.1 Navraagverwerking in MySQL

Die gedeelte word aan die hand van Balling en Zawodny (2004:79-95) verduidelik. MySQL kan bepaal as die "WHERE"-gedeelte 'n onuitvoerbare versoek rig, en lewer dadelik 'n resultaat van geen rekords sonder verwerking. 'n Tipiese voorbeeld is "WHERE id < 5000 and id > 30000".

MySQL AB (2005:425) verduidelik dat regte elke navraag beïnvloed. Die regte van 'n gebruiker moet dus so eenvoudig moontlik gehou word sodat die verwerking wat nodig is vir die regte nie 'n groot impak op die navraag se uitvoering sal maak nie.

MySQL hanteer die verwerking van navrae in 'n paar fases, wat in figuur 3.10 getoon word.

Figuur 3.10 – Die fases van navraagverwerking vir MySQL.



Die onderskeie fases word in die volgende paragrawe bespreek.

3.3.5.1.1 Navraagkasgeheue

Voor enige navraagverwerking gedoen word, probeer MySQL om die resultaat van enige seleksienavraag in die navraagkasgeheue te vind. 'n "Hash"-funksie word op die navraag toegepas en die "hash"-waarde word gebruik om te soek vir die navraag in die kasgeheue. Die navraag moet presies dieselfde as die vorige navraag wees om dieselfde "hash"-waarde te kry. Navrae wat nie gereeld uitvoer nie, moet gestel word sodat die navraag nie na die kasgeheue toe geskryf word nie, om sodoende die kasgeheue beskikbaar te stel vir navrae wat gereeld voorkom. Die kasgeheue is by verstek in gebruik. As die navraag in die kasgeheue bestaan, word die resultaat na die kliënt toe gestuur sonder enige verdere verwerking. Die MySQL-bevel "FLUSH TABLES" (MySQL AB, 2005:802) maak die kasgeheue skoon en die

"SQL_NO_CACHE"-bevel (MySQL AB, 2005:385) wat in 'n seleksiebevel gebruik word, keer dat die navraag in die kasgeheue geplaas word.

3.3.5.1.2 Formatering ("parsing"), analise en optimering

As die navraag nie in die kasgeheue gevind word nie, moet die navraag eers geformateer word en in verskillende dele verdeel word. Die navraag word getoets om te sien of die sintaksis geldig is, en sekere basiese inligting word verkry – soos byvoorbeeld die tipe van die navraaguitdrukking (seleksie, bywerk of byvoeg), watter tabelle gebruik word (ook aliasse), wat die "WHERE"-gedeelte bevat, en of die navraag leidrade of ander opsies bevat.

Sodra die formatering en verdeling plaasgevind het, word die navraagoptimeerder geroep. Die taak van die navraagoptimeerder is om die navraag op die mees effektiewe manier uit te voer. Die basies beginsel is om die aantal rye waarop gewerk word te minimeer. Die optimeerder volg 'n stel reëls om die besluite te neem. Die optimeerder toets vir die bestaan van indekse wat moontlik gebruik kan word om die rye vinniger te kry. As meervoudige indekse beskikbaar is, word die "beste" (volgens MySQL) indeks gekry vir die navraag en ook die spesifieke tabel. Die optimeerder besluit tydens verbindingbewerkings watter tabelle afhanklik van ander tabelle is. Die optimale verbindingsplan word bereken vir tabelle wat verbind moet word. MySQL konsentreer hoofsaaklik op die keuse van indekse en die orde waarin verbindings plaasvind. Die twee faktore is nie die enigste faktore nie, maar geniet prioriteit in MySQL se optimeringsfase. MySQL se optimeerder neem besluite volgens die statistiese inligting wat beskikbaar is. Die statistiese inligting verskil vir verskillende stoorenjins, en navrae se uitvoer verskil vir die stoorenjins.

Gedurende verbindingbewerkings moet MySQL besluit in watter volgorde die verbindings moet plaasvind om so min as moontlik rye te lees. Die keuse van die volgorde is 'n swakpunt van MySQL, en die optimeerder probeer elke moontlike kombinasie om die volgorde te bepaal. Die aantal tabelle wat verbind word, moet dus tot 'n minimum beperk word.

3.3.5.1.3 Uitvoering

MySQL gebruik die plan wat uit die vorige fase verkry is en verkry dan die kwalifiserende rye. Gedurende hierdie proses kan MySQL van 'n tydelike tabel (in geheue of op skyf) gebruik maak om die resultate te stoor. Die resultaat word dan aan die gebruiker gestuur. Terwyl die navraag uitgevoer word, verkry MySQL sekere inligting – byvoorbeeld oor die eienaar van die navraag, die tydsduur van die proses en die hoeveelheid rye verkry, en stoor dan die inligting in die stadige navraaglog indien die navraagtyd 'n sekere perk oorskry.

3.3.5.2 Die optimering van sekere navraaggedeeltes in MySQL

Die eerste navraaggedeelte wat ondersoek word, is die "WHERE"-gedeelte wat met behulp van MySQL AB (2005:435-437) verduidelik word. 'n Paar van die tegnieke wat gebruik word, word hier getoon:

- MySQL verwyder onnodig hakies; ((a AND b)) word byvoorbeeld verander na (a AND b).
- Konstante uitdrukkings se formaat word verander; (a<b AND b=c) AND a = 5 word byvoorbeeld verander na b > 5 AND b=c AND a=5
- Konstante voorwaardes word verwyder:
  (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6) word verander na B=5 OR B=6
- Konstante uitdrukkings wat deur indekse gebruik word, word net een keer verwerk.
- Die "HAVING"-gedeelte word ineengevleg met die "WHERE"-gedeelte as "GROUP BY"- of groepfunksies nie gebruik word nie.
- Indien moontlik, word 'n eenvoudiger "WHERE"-gedeelte vir elke tabel in 'n verbinding verkry.
- Alle konstante tabelle word eerste in navrae gelees. 'n Konstante tabel is 'n leë tabel, 'n tabel met een ry, of 'n tabel wat 'n "WHERE"-gedeelte het wat 'n primêre of unieke indeks gebruik en waar al die indeksgedeeltes met konstante uitdrukkings vergelyk word en nie "NULL"-waardes kan bevat nie.
- Die beste verbindingkombinasies word verkry deur al die moontlike opsies te deursoek. As alle kolomme in die "ORDER BY"- en "GROUP BY"-gedeelte van dieselfde tabel kom, geniet die tabel voorrang tydens verbinding.
- As die "ORDER BY"- en die "GROUP BY"-gedeelte verskil, of die twee gedeeltes kolomme van ander tabelle as die eerste tabel in die verbindingslys bevat, word 'n tydelik tabel geskep.
- MySQL kan in sommige gevalle rye vanaf die indeks lees sonder om na die datalêer te kyk.
- Die rye wat nie aan die "HAVING"-gedeelte voldoen nie, word oorgeslaan voor die ry as resultaat uitgegee word.

Twee spesiale optimeringstegnieke word op die "WHERE"-gedeelte toegepas, naamlik reeksoptimering en indekssaamvoegoptimering.

- Reeksoptimering – Die optimering word toegepas op indekse waarvan een of meer gedeeltes aan 'n spesifieke reeks (met ander woorde die gedeelte moet groter en kleiner as sekere waardes wees) moet voldoen. Die reeksvoorwaardes moet uit die "WHERE"-gedeelte verkry word en vereenvoudig word (byvoorbeeld voorwaardes wat leë reekse gee).

- Indekssaamvoegoptimering – Die indekssaamvoegmetode word gebruik om rye te verkry van verskeie "ref"-, "ref_or_null"- of reekssoektogte (sien paragraaf 3.3.6.1 vir 'n verduideliking van die tipes) en die resultaat saam te voeg in een resultaat. Die metode gebruik verskeie toegangalgoritmes, naamlik deursnee ("intersection"), "union" en "sort-union". Die metode is vanaf MySQL 5.0.0 beskikbaar en word nie verder hier bespreek nie.

Die "IS NULL"-uitdrukking word op dieselfde wyse geoptimeer as wat vir konstante waardes gedoen word. As die uitdrukking toegepas word op 'n kolom wat nie "NULL"-waarde toelaat nie, word die uitdrukking uit die navraag verwyder. Die "DISTINCT"-gedeelte benodig in baie gevalle 'n tydelike tabel (veral saam met "ORDER BY"). Die "DISTINCT"-gedeelte word normaalweg gesien as 'n spesiale geval van "GROUP BY". Die optimering hiervan is dus baie dieselfde as "GROUP BY" se optimering. Die algemene metode vir die optimering van "GROUP BY"-gedeelte is om die hele tabel te deursoek en 'n tydelike tabel te skep waar al die rye van elke groep op mekaar volg. Die groepfunksies word dan op hierdie tydelike tabel toegepas. In sommige gevalle kan MySQL indekse gebruik om die gedeelte beter te hanteer. 'n Minder streng indekssoektog kan gebruik word om die groepe direk te verkry, of 'n strenger indekssoektog kan gebruik word – wat 'n volle indekssoektog of reeks indekssoektog behels.

3.3.5.3 Die optimering van databasisstrukture in MySQL

MySQL AB (2005:458-459) beskryf 'n paar metodes om spasie te bespaar, wat beter produktiwiteit lewer aangesien meer rye per bladsy kan voorkom. Die kleinste datatipes moontlik moet gebruik word. Heelgetalle moet byvoorbeeld eerder van "MEDIUMINT" as "INT" gebruik maak. Kolomme moet verkieslik nie "NULL"-waardes bevat nie, omdat ekstra inligting gestoor moet word om die "NULL"-waardes te akkommodeer (soos by SQL Server in paragraaf 3.2.5.1). Indekse moet nie onnodig geskep word nie en die sleutels van die indekse moet so klein as moontlik wees.

Die MyISAM-stoorenjin implementeer 'n tegniek waar tabelblokke wat meer gereeld gebruik word, langer in die geheue bly (MySQL AB, 2005:463-468). Indeksblokke gebruik 'n spesiale sleutelbufferstruktuur (of sleutelkasgeheuestruktuur) en datablokke gebruik die bedryfstelsel se lokale lêerstelselkasgeheue. As die sleutelbufferstruktuur nie gebruik word nie, word die indeksblokke dieselfde as die datablokke hanteer. Alle bufferblokke in 'n sleutelbufferstruktuur het dieselfde grootte. Die grootte van die bufferblokke kan kleiner as, gelyk aan of groter wees as die grootte van 'n tabel se indeksblok, maar die twee waardes is gewoonlik veelvoude van mekaar. As data vanaf die tabel se indeksblok verkry moet word, word eerstens in die bufferblokke van die sleutelbuffer vir die data gesoek. As die inligting in die sleutelbuffer beskikbaar is, word die data gebruik, anders vervang die tabel se indeksblok een van die

bestaande bufferblokke in die sleutelbuffer. As die blok wat vervang moet word, gewysig is, word die blok as vuil ("dirty") beskou en die blok word eers uitgeskryf na die tabelindeksblok waar dit vandaan kom. MySQL volg 'n strategie wat die blokke vervang wat die langste gelede gebruik is ("least recently used"). 'n Spesiale lys van gebruikte blokke word in die sleutelbuffer onderhou waar elke blok wat onlangs gebruik is agter in die lys geplaas word. Die blokke aan die begin van die lys word kandidate om vervang te raak.

MySQL 4.1.0 ondersteun gedeelde toegang tot die sleutelbuffer, waar buffers wat nie gewysig word nie, deur verskeie "threads" gebruik kan word. Buffers wat gewysig word, veroorsaak dat "threads" wat die buffer wil gebruik, moet wag totdat die bywerkbewerking afgehandel is. (MySQL AB, 2005:464). Volgens MySQL AB (2005:464-466) besit MySQL 4.1.1 'n addisionele tegniek om die sleutelbuffer se mededinging om toegang te verminder. Die tegniek behels dat verskillende tabelindekse tot verskillende sleutelbuffers toegeken kan word.

'n Metode bestaan wat, in die plek van die strategie om die verouderde (lank terug gebruik) blokke te vervang, gebruik kan word in die keuse van blokke wat in die sleutelbuffer vervang mag word. Die metode word die middelpuntbyvoegstrategie genoem (MySQL AB, 2005:466-467). Die lys van blokke word in twee dele verdeel, naamlik 'n warm subketting ("hot sub-chain") en 'n lou subketting ("warm sub-chain"). Die sleutelbuffer se bestuurstelsel verseker dat die lou subketting nie te kort is nie. As 'n indeksblok vanaf die tabel gelees en in die sleutelbuffer geplaas word, word dit aan die einde van die lou subketting geplaas. As die blok 'n sekere aantal kere gebruik is, word die blok in die warm subketting geplaas. Die blok wat oorgedra word, word aan die einde van die warm subketting geplaas. As 'n blok 'n geruime tyd by die begin van die warm subketting bly, word die blok terugverplaas na die lou subketting. Die moontlikheid bestaan om 'n hele indeks se blokke in die sleutelbuffer te laai as die spasie voldoende is (MySQL AB, 2005:467). Die voordeel van hierdie manier is dat die blokke sekwensieel vanaf die skyf na die sleutelbuffer geskryf word.

3.3.6 Toepassings en gereedskap gebaseer op MySQL

MySQL is, soos reeds genoem, 'n oopbronprojek en die toepassings wat vir die databasisbestuurstelsel beskikbaar is, word deur verskeie programmeerders ontwerp. MySQL AB bied MySQL Administrator, wat ooreenstem met Microsoft SQL Enterprise Manager (nie noodwendig al die funksionaliteit, maar beide gebruik dieselfde konsepte (sien paragraaf 3.2.8)), en MySQL Query Browser, wat ooreenstem met SQL Query Analyzer (weereens het MySQL se weergawe dalk nie al die funksionaliteit nie, maar die beginsel bly dieselfde). Die Index Tuning Wizard word deur die "EXPLAIN"-bevel in MySQL vervang. Die bevel verskaf

egter net inligting oor die betrokke navrae en maak nie voorstelle vir indekse nie. Die "EXPLAIN"-bevel word aan die hand van MySQL AB (2005:425-434) beskryf.

3.3.6.1 Die "EXPLAIN"-bevel van MySQL

Die bevel se resultaat kan gebruik word om 'n tabel se kolomme, of die wyse waarop MySQL 'n navraag uitvoer, te beskryf. Die resultaat kan gebruik word om die keuse van indekse te vergemaklik. Die "ANALYZE TABLE"-bevel (MySQL AB, 2005:773) word gebruik om die statistiese inligting vir 'n spesifieke tabel te korrigeer sodat die "EXPLAIN"-bevel die korrekte inligting vertoon. Die "EXPLAIN"-bevel toon aan of die optimeerder tabelle op 'n optimale wyse ten opsigte van die volgorde van die tabelle verbind. 'n Ry inligting word vir elke tabel vertoon wat deel uitmaak van 'n seleksie-uitdrukking wat met die "EXPLAIN"-bevel ondersoek word.

Die bykomende "EXTENDED"-argument veroorsaak dat "EXPLAIN" meer inligting lewer wat met die bevel "SHOW WARNINGS" vertoon word. Die ekstra opsie vertoon inligting oor die wyse waarop die optimeerder tabel- en kolomname in die seleksie-uitdrukking kwalifiseer, hoe die seleksie-uitdrukking lyk nadat die optimeerder die uitdrukking herskryf en geoptimeer het, en nog ander inligting oor die optimeerproses. Die verduideliking van die "EXPLAIN"-bevel steun baie op die SQL-taal, en 'n goeie begrip van die taal sal die verduideliking vereenvoudig (sien bylae A). Die "EXPLAIN"-bevel word baie in hoofstuk 6 as deel van die eksperimentele gedeelte van die verhandeling gebruik.

Die resultaat van die "EXPLAIN"-bevel se kolomme word in tabel 3.5 beskryf.

Tabel 3.5 – Kolomme en beskrywings van die "EXPLAIN"-bevel se resultaat

Kolom	**Moontlike Waarde(s)**	**Beskrywing**
id	Getal	Die seleksie- uitdrukking se unieke id.
Seleksie-tipe ("select_type")	• "SIMPLE" – 'n Eenvoudige seleksie. • "PRIMARY" – Die buitekantste seleksie. • "UNION" – Die tweede seleksie-uitdrukking in 'n "UNION"-bewerking. • "DEPENDANT UNION" – Die tweede seleksie-uitdrukking in 'n "UNION"-bewerking, wat afhanklik is van 'n buitenste navraag van 'n verbinding. • "UNION RESULT" – Die resultaat van 'n "UNION"-bewerking. • "SUBQUERY" – Die eerste seleksie in 'n subnavraag. • "DEPENDENT SUBQUERY" – Die eerste seleksie in 'n subnavraag, wat afhanklik is van 'n buitenste navraag van 'n verbinding. • "DERIVED" – 'n Afgeleide tabel se seleksie (subnavraag in die	Die tipe seleksie, wat enige een van die moontlike waardes kan aanneem.

	"FROM"-gedeelte)	
Tabel ("table")	Lys van tabel(le)	Die tabel waarheen die ry vir die bevel se uitvoer verwys
Verbindingstipe ("type")	• "system" – Die tabel bevat net een ry. • "const" – Die tabel het hoogstens een passende ry wat aan die begin van die navraag gelees word. Die waardes van die een ry word as konstantes deur die res van die optimeerder beskou. Navrae op die konstante tabelle is baie vinnig aangesien die tabel net een keer gelees word. 'n Voorbeeld van hierdie tipe is wanneer al die dele van 'n primêre of unieke indeks met konstante waardes vergelyk word. • "eq_ref" – 'n Enkele ry vanaf die tabel word vir elke kombinasie van rye vir die vorige tabelle gelees. Die tipe word gebruik wanneer al die dele van 'n indeks in die verbinding gebruik word en die indeks 'n primêre sleutel of unieke indeks is. • "ref" – Alle rye wat 'n indekswaarde besit wat pas, word vanaf die verwysde tabel vir elke kombinasie van die rye van die vorige tabelle gelees. Die verbindingstipe word gebruik as net die linkerkantste deel van die sleutel gebruik word of as die sleutel nie 'n primêre sleutel of unieke indeks is nie (met ander woorde die verbinding kan meer kolomme op grond van die sleutel selekteer). • "ref_or_null" – Soortgelyk aan "ref"-tipe, behalwe dat MySQL ook soektogte doen vir rye wat "NULL" bevat. • "unique_subquery" – Die tipe vervang "ref" vir sommige "IN"-subnavrae (sien paragraaf 3.4 in bylaag A) waar die subnavraag se waardes uniek is. Die "unique_subquery" behels 'n indekssoektog-funksie ("lookup function") wat die subnavraag heeltemal vervang. • "index_subquery" – Die tipe is soortgelyk aan die "unique_subquery"-tipe. Subnavrae met "IN"-gedeeltes word deur hierdie tipe vervang. • "range" – Slegs rye wat in 'n spesifieke reeks is, word met behulp van die indeks verkry. Die "key"-kolom noem die indeks wat gebruik is. Die "key_len" bevat die langste sleutelgedeelte wat gebruik is. Die "ref"-kolom is "NULL" vir hierdie tipe. • "index" – Die tipe is dieselfde as "ALL", behalwe dat net die indeksboom deursoek word. Die deursoek is normaalweg vinniger as die "ALL"-tipe, aangesien die indekslêer kleiner is as die datalêer. MySQL gebruik hierdie verbindingstipe waar die navraag net kolomme gebruik wat deel van 'n enkele indeks (oordekkingsindek) is. • "ALL" – Die tipe gebruik 'n volle tabelsoektog vir elke kombinasie van rye van die vorige tabelle.	Die verbindingstipe wat gebruik word. Die moontlike waardes is in volgorde van die beste tipe tot die swakste
Moontlike sleutels ("possible_keys")	Lys van indeks(e)	Die kolom lys die moontlike indekse wat MySQL kan gebruik om die rye in die tabel te vind. As die kolom "NULL" is, is daar geen bruikbare indekse nie.

Sleutel gebruik ("key")	Indeks se naam. As geen indeks gekies is nie, bevat die kolom "NULL".	Die kolom dui die indeks aan wat gebruik is.
Sleutellengte ("key_len")	Lengte van sleutel	Die kolom dui die lengte van die sleutel wat MySQL gekies het aan.
Verwysing ("ref")	Kolom(me)	Die kolom vertoon die kolomme of konstantes aan wat saam met die "key"-kolom gebruik word om die rye te selekteer.
Rye ("rows")	Getal	Die aantal rye wat MySQL verwag om te ondersoek om die navraag uit te voer. Die produk van totale aantal verwagte rye van elke ry in die resultaat geen 'n aanduiding van die effektiwiteit van 'n verbinding (MySQL AB, 2005:463-464).
Ekstra ("Extra")	• "Distinct" – MySQL hou op om te soek nadat die eerste ry vir die huidige ry-kombinasie gevind is. • "Not exists" – 'n "LEFT JOIN"-optimering (sien paragraaf 3.3 in bylaag A) was moontlik en geen verdere rye word vir die vorige ry-kombinasies ondersoek nadat een ooreenkomstige ry verkry is wat voldoen aan die "LEFT JOIN"-kriteria nie. • "range checked for each record (index map: #)" – Aanvanklik kan geen goeie indekse gebruik word nie, maar sommige indekse mag dalk gebruik word soos wat kolomwaardes van vorige tabelle bekend raak. Vir elke vorige ry-kombinasie van die vorige tabelle in 'n verbinding word die moontlikheid van die gebruik van 'n "range"- of "index_merge"-toegangsmetode ondersoek. Die metode is nie so vinnig nie, maar is steeds vinniger as 'n verbinding sonder 'n indeks. • "Using filesort" – MySQL moet 'n ekstra rondte doen om uit te vind hoe om die rye in gesorteerde volgorde te verkry. Die sorteerproses beweeg deur al die rye, volgens die verbindingstipe, en stoor die sorteringsleutel en wyser na die ry vir alle rye wat aan die "WHERE"-gedeelte voldoen. Die sleutels word gesorteer en die rye word met behulp van die wysers in gesorteerde volgorde verkry. • "Using index" – Die kolom-inligting word vanaf die tabel verkry deur net die inligting in die indeksboom te gebruik. • "Using temporary" – 'n Tydelike tabel word gedurende die verwerking en uitvoer van 'n navraag geskep. • "Using where" – Die "WHERE"-gedeelte word gebruik om rye te beperk wat met die volgende tabel verbind moet word. • "Using index for group-by" – Die inligting dui aan dat MySQL 'n indeks gevind het om al die kolomme van 'n "GROUP BY"- of "DISTINCT"-navraag te verkry, sonder enige verdere skyf-aksies.	Addisionele inligting oor die manier waarop MySQL navrae hanteer, word hier vertoon.

'n Aanduiding van die gehalte van 'n verbinding word verkry deur die "rows"-kolomme met mekaar te vermenigvuldig, wat ongeveer die aantal rye aandui wat MySQL moet deursoek vir die navraag.

3.4 Slotparagraaf

Dit blyk duidelik dat die onderskeie databasisbestuurstelsels 'n redelike breë onderwerp is, en die verhandeling het maar net 'n kort beskrywing oor die onderwerpe verskaf. Elke databasisbestuurstelsel moet individueel ondersoek word indien die bestuurstelsel as 'n produksie-oplossing oorweeg word. In hierdie hoofstuk is die interne werking van die twee gekose databasisbestuurstelsels bestudeer. Die belangrikste aspekte in hierdie hoofstuk is die besprekings oor die navraagverwerker, indekse en die onderskeie toepassings van elke databasisbestuurstelsel. Die volgende hoofstuk bespreek algemene optimeringstegnieke en -riglyne sowel as sekere tegnieke wat net vir die onderskeie databasisbestuurstelsels geld.

HOOFSTUK 4

FORMULERING VAN RIGLYNE VIR NAVRAAG- EN DATABASISOPTIMERING

4.1 Inleiding

Die proses van navraag- en databasisoptimering is nie 'n proses wat maklik en vinnig geskied nie. Die proses vereis deurlopende aandag, d.w.s. deur die hele ontwikkelingsfase van die databasis. Die kuns van optimering is nie iets wat net uit teoretiese agtergronde alleen geleer kan word nie. Dit bevat 'n sterk element van ondervinding. Die volgende riglyne is aanduidings van wat tydens die optimeringsproses as belangrik beskou word. Die riglyne word, waar nodig, aan die hand van SQL Server en MySQL verduidelik. Die verskillende optimerings is egter afhanklik van die spesifieke geval wat ondersoek word, en kan nooit volledig veralgemeen word nie.

Delaney (2001:867) beweer dat navraagoptimering normaalweg bewerkstellig word deur databasisontwerp of indekskeuses te verander, eerder as die sintaksis van die navraag self. Delaney (ook Shasha en Bonnet (2003:124)) stel dat dit beter is om die basiese strukture van die begin af reg te kry as om later verskeie tegnieke toe te pas om die produktiwiteit te verbeter.

Verskeie tegnieke, riglyne en probleemsituasies word bespreek. Die boeke van Delaney (2001:867-942) en Balling en Zawodny (2004:45-128) word as basis vir hierdie riglyne gebruik. Die optimeringstegnieke oor indekse in hierdie hoofstuk word in hoofstuk 6 toegepas.

Paragraaf 4.2 konsentreer op optimering deur databasisontwerp en -administrasie. Die volgende paragraaf (4.3) dui die nut van werkverrigtingstoetse ("benchmark") aan. Die belangrikste paragraaf in die hoofstuk is paragraaf 4.4, wat die keuses van indekse bespreek. Paragraaf 4.5 bespreek kortliks die aspekte van "gelyklopendheid" ("concurrency") teenoor "konsekwentheid" ("consistency"). Paragraaf 4.6 bespreek die konsepte van "versperring" ("blocking") en "dooiepunte" ("deadlocks"). Paragraaf 4.7 hanteer die tegnieke "ladingbalansering" en "groepering" ("clustering"). Die opstelling van navrae om optimering te verbeter, word in paragraaf 4.8 bespreek. Die tweede laaste paragraaf (4.9) bespreek hardeware en sagteware ten opsigte van optimering.

4.2 Databasisontwerp en -administrasie

Die algemene begrip bestaan dat nuwe hardeware die werkverrigting van die toepassing sal verbeter. Hardeware-opgraderings het – indien die fondse beskikbaar is – normaalweg 'n verbetering in werkverrigting tot gevolg, alhoewel die verbetering soms klein kan wees. Verandering ten opsigte van die databasis en toepassing se ontwerp kan potensieel groter produktiwiteit lewer as hardeware-opgraderings. 'n Paar benaderings word in die volgende paragrawe bespreek.

4.2.1 Normalisering

Normalisering is een van die belangrikste tegnieke vir databasisontwerp. Rob en Coronel (2007:148) definieer normalisering as 'n proses wat data-oorbodigheid verminder en sodoende data-anomaliteite verwyder. 'n Voorbeeld van die data-anomaliteite is 'n tabel wat elke werknemer se inligting gesamentlik met die werknemer se departement inligting bevat. Die departement inligting word dus herhaal vir alle werknemers wat in dieselfde departement is. Indien die departement se naam sou verander, moet die inligting van elke werknemer van die betrokke departement verander. Rob en Coronel (2007:148) stel dat normalisering uit 'n reeks vlakke bestaan wat "normaalvorme" genoem word. Die onderskeie vlakke word beskryf:

- Eerste normaalvorm – Die eerste normaalvorm word verkry deur die verwydering van herhalende groepe (Rob & Coronel, 2007:155). Herhalende groepe dui op data-oorbodigheid. 'n Voorbeeld van die groep is waar 'n projek aan verskeie werknemers gekoppel is en die inligting van die projek en die werknemer in één tabel gestoor word. Die herhalende groepe moet verwyder word sodat elke ry slegs één entiteit definieer. 'n geskikte waarde word met ander woorde in ten minste die primêre sleutel-kolom vir die rye bygevoeg.
- Tweede normaalvorm – Die tweede normaalvorm word bereik as die tabel alreeds in eerste normaalvorm is en geen gedeeltelike afhanklikhede bestaan nie (Rob & Coronel, 2007:157). Gedeeltelike afhanklikhede is volgens Rob en Coronel (2007:154) kolomme wat net van 'n gedeelte van die saamgestelde primêre sleutel afhanklik is. Die gedeeltelike afhanklike kolomme word geneem om nuwe tabelle te skep wat bestaan uit die betrokke kolom, of kolomme, en die primêre sleutelgedeelte of -gedeeltes. Die primêre sleutel van die oorspronklike tabel bly onveranderd.
- Derde normaalvorm – Rob en Coronel (2007:158) definieer die derde normaalvorm as 'n tabel wat alreeds in die tweede normaalvorm is en geen transitiewe afhanklikhede besit nie. Transitiewe afhanklikhede is die afhanklikheid van een nie-primêre kolom van 'n ander nie-primêre kolom (Rob & Coronel, 2007:154). Die derde normaalvorm word bereik deur

kolomme wat van ander nie-primêre kolomme afhanklik is na 'n aparte tabel te skuif. Die sleutel van die nuwe tabel is die kolomme waarop die afhanklike kolomme afhanklik is. In die ou tabel bly die kolomme waarop die ander kolomme afhanklik was as 'n verwysing na die nuwe tabel agter.

- Ander normaalvorme bestaan, maar val buite die bestek van hierdie verhandeling.

Delaney (2001:870) noem normalisering ook "logiese ontwerp". Normalisering vergemaklik die bywerk van die databasis. Die probleem ontstaan dat normalisering die aantal verbindingsaksies van tabelle verhoog. As die stelsel eerder op vinnige navrae konsentreer, moet normalisering liefs nie oorweeg word nie (of dalk net gedeeltelike normalisering). Indien bywerkings gereeld plaasvind en navrae nie so krities is nie, bied normalisering 'n verhoogde logiese voorstelling van die data.

4.2.2 Identifiseer kritiese transaksies

Die vasstelling van die kritiese transaksies speel 'n groot rol in die ontwerp van die databasis. Transaksies in hierdie geval is nie net navrae nie, maar eerder enige bywerking op die databasis. Delaney (2001:871) stel dat oorbodigheid van data oorweeg moet word indien kritiese navrae meer as vier verbindings behels. Oorbodigheid van data bemoeilik die bywerk van data, aangesien verskeie velde of rekords moet verander. Opsommende tabelle is 'n moontlikheid wat oorweeg kan word indien opsommende berekenings van belang is. 'n Voorbeeld hiervan is die balans van 'n kliënt. In plaas daarvan om die balans elke keer uit die data te bepaal, kan nuwe data net by die opsommende waarde gevoeg word. Hierdie strategie benodig 'n klein hoeveelheid addisionele stoorplek, maar mag die werkverrigting verhoog.

4.2.3 Lengte van 'n tabel se kolomme en sleutels

Die lengte van kolomme en die tipes is baie belangrike aspekte tydens databasisontwerp. Kolomme waarvan die waardes se lengtes baie mag verskil, moet gebruik maak van veranderlikelengte-tipes. 'n Tipiese voorbeeld sal 'n persoon se naam wees. Die kolom sal gewoonlik van tipe "VARCHAR" wees, wat 'n veranderlikelengtekarakterveld is. As die kolom egter karakters bevat wat nie wissel in lengte nie, kan die tipe "CHAR" 'n beter passing wees. Die strewe is om die ry-lengtes te beperk sodat meer rye op 'n gegewe bladsy kan kom. Die hoeveelheid bladsye bepaal die hoeveelheid soektogte. Die ry-lengte van die primêre sleutel en die kolomme waarop indekse toegepas is, moet so klein as moontlik gehou word ten einde die werkverrigting ten opsigte van die indeks te verminder. Gestel slegs sekere kolomme verkry gereelde verwysings in die seleksiegedeelte van navrae. 'n Metode bestaan waarvolgens hierdie kolomme geneem word en 'n nuwe tabel geskep word om sodoende 'n kleiner tabel te

verkry wat gereeld gebruik word, en waarvan meer rye op 'n databladsy kan pas. Die metode se verbetering in produktiwiteit moet opgeweeg word teen die koste van bywerkings op die twee tabelle. As 'n tabel se ry-lengte klein is, verhoog die waarskynlikheid dat die data eerder vanaf die kasgeheue as vanaf die skyf gelees word.

4.2.4 Identifiseer piektye

Die produktiwiteit van 'n databasis is afhanklik van die tye van gebruik. 'n Databasis word verskillend ontwerp vir 'n stelsel wat transaksies eweredig versprei en 'n stelsel wat transaksies op 'n sekere tyd van die dag laat piek. Die tipe transaksies gedurende hierdie piektye het ook 'n impak. As die piektyd byvoorbeeld meer bywerktransaksies verkry, sal 'n genormaliseerde model die stelsel moontlik beter pas. Piektyd word normaalweg deur kritiese transaksies veroorsaak. Whitehorn en Marklyn (2003:328) beskryf die tegniek waar indekse afgehaal word wanneer die invoerfrekwensie die navraagfrekwensie oorskry. Sodra die navraagfrekwensie weer die invoerfrekwensie oorskry, word die indekse weer opgestel.

4.2.5 Identifiseer die wyse waarop die data gebruik word

Balling en Zawodny (2004:132) bespreek 'n metode genaamd rolgebaseerde verdeling. Die metode verdeel die navrae volgens hul spesifieke rolle en skep dan databasisse vir elke rol. Die databasisse kan selfs op verskillende bedieners geplaas word. Dit sluit aan by die metode waar twee tabelle vir verskillende kolomme geskep word volgens die mate van gebruik van die kolomme. As voorbeeld kan data oor studente en data oor elke student se verhandeling geneem word. Die navrae op die verhandeling sal groter koste dra as gevolg van die grootte van die data. Dit kan dus voordelig wees om die verhandelingdata van die student se data te skei. Nog 'n metode wat deur Balling en Zawodny (2004:181) beskryf word, is datagebaseerde verdeling. Die metode is baie dieselfde as rolgebaseerde verdeling, behalwe dat die verdeling geskied op grond van 'n eienskap van die data. As voorbeeld kan data verdeel word volgens vanne wat begin met die letters *a* tot *m* en vanne wat begin met die letters *n* tot *z*.

4.3 Werkverrigtingstoetse ("Benchmark")

Werkverrigtingstoetse is handig in dié opsig dat veranderings se impak op produktiwiteit gesien kan word. Werkverrigtingstoetse moet verkieslik so eenvoudig moontlik wees en so na as moontlik aan die werklike stelsel. Een van die eenvoudigste maniere is om 'n prototipe databasis met 'n groot hoeveelheid willekeurige data te skep. As die data 'n beter voorstelling van die werklike data kan wees, sal die toets 'n beter resultaat kan lewer. Die hoeveelheid data moet 'n merkbare verskil kan aandui tussen 'n navraag van 'n enkele ry met 'n indeks en 'n

navraag van 'n enkele ry met 'n tabeldeursoek. Hoë kasgeheue-tref-verhoudings ("cache-hit ratios") moet in die algemeen nie as 'n maatstaf gebruik word nie. Dit is beter om na elke navraag die kasgeheue skoon te maak om sodoende die slegste geval (waar data vanaf die skyf verkry moet word) te ondersoek.

Werkverrigtingstoetse behels die vergelyking van vorige resultate met mekaar. Balling en Zawodny (2004:46-48) stel dat werkverrigtingstoetse in twee groepe gedeel kan word, naamlik toetse wat alternatiewe vergelyk en toetse wat die grense (maksimum en minimum) van die databasis toets (volgens huidige konfigurasie). 'n Paar voorstelle wat oorweeg moet word:

- Verander net een aspek op 'n slag om sodoende die effek van elke veranderlike duidelik te kan sien.
- Toetse moet met ander waardes herhaal word sodat die beste moontlik stelselkonfigurasie verkry kan word. Toetse moet ook gereeld geloop word om buite-invloede se uitwerking uit te skakel.
- Gebruik so ver moontlik werklike data, aangesien optimering 'n moeilike proses is as die toetsdata nie 'n akkurate beeld van die werklike data is nie.
- Gebruik verskeie kliënte om die toetse te doen, maar moenie te veel gebruik nie. Die voorspelde aantal kliënte moet as 'n maatstaf gebruik word vir die werkverrigtingstoetse.
- Skei die kliënt van die bediener sodat die kliënt se eie verwerkingsprosesse nie die resultaat van die databasisbestuurstelsel se prosesse beïnvloed nie. Die data word dan oor 'n netwerk gestuur wat die normale gebruik van 'n databasis volg.

Sekere elemente van die werkverrigtingstoetsprosedure word in Hoofstuk 5 en Hoofstuk 6 toegepas.

4.4 Die keuse van indekse

Indekse is al redelik uitvoerig bespreek in die vorige hoofstukke (sien paragrawe 2.4, 3.2.6 en 3.3.4). Indekse kan navrae bespoedig, maar sal 'n nadelige uitwerking op bywerkings hê as gevolg van die feit dat die indekselemente sowel as die data-elemente onderhou moet word. Delaney (2001:880) noem dat die suksesvolle keuse van indekse geskied deur die gebruik van die data te verstaan, die tipes en frekwensies van navrae te ken en te weet hoe indekse die databasisbestuurstelsel kan help om die data vinnig op te spoor.

Gebondelde indekse (sien paragraaf 2.4.1) werk goed vir reekssoektogte en vir navrae waar die data volgens die gebondelde sleutel gesorteer moet wees. Shasha en Bonnet (2003:93) stel dat gebondelde indekse ook goed werk vir gelykheidstoets wat meer as een resultaat lewer.

Gebondelde indekse se elemente sal altyd uniek wees. Gewoonlik het elke tabel 'n gebondelde indeks. Die primêresleutelkolom is per definisie normaalweg 'n gebondelde indeks. Ander kolomme mag 'n gebondelde indeks verkry, maar daar is net een gebondelde indeks per tabel moontlik. Die gebondelde indeks moet dus gekies word vir 'n kolom waarvoor die groepering van waardes 'n voordeel kan wees. Indien die behoefte bestaan om meer as een gebondelde indeks te skep, bied Shasha en Bonnet (2003:95) 'n oplossing van 'n oorbodige tabel. Die tabel word met dieselfde data gevul en 'n tweede gebondelde indeks kan op die nuwe tabel geskep word. Hierdie tegniek werk goed indien bywerkings nie gereeld geskied nie.

Die optimeerder gebruik eerder gebondelde indekse as ontbondelde indekse, aangesien die databladsye deel vorm van die gebondelde indekse. Ontbondelde indekse moet gekies word wanneer die navrae groot hoeveelhede rye elimineer. Die ontbondelde indeks sal byvoorbeeld nie 'n goeie keuse wees vir 'n soektog wat tussen manlik en vroulik onderskei nie. Die kandidaatkolom vir die ontbondelde indeks moet 'n redelike verskeidenheid van waardes bevat.

Bywerking moet tydens die keuse van indekse in gedagte gehou word, aangesien bywerkings 'n meer hulpbronintensiewe bewerking vir indekse is. As die kritiese transaksies meer na bywerkings neig, moet die indeks dalk nie gebruik word nie. Geleenthede moet gesoek word waar oordekkingsindekse gebruik kan word, met ander woorde waar al die kolomme in die selekteerbevel uit die indeks verkry kan word. Die oordekkingsindeks gebruik meer spasie en moet nie geforseer word nie. 'n Belangrike aspek en verskil tussen SQL Server en MySQL is dat MySQL net een indeks per navraag per tabel (sien paragraaf 3.3.4) kan gebruik, waar SQL Server meer as een indeks per tabel kan gebruik. MySQL besit gedeeltelike indekse (sien paragraaf 3.3.4), waar SQL Server nie hierdie funksionaliteit besit nie.

Unieke indekse (gebondel of nie) verskaf die beste selektiwiteit vir navrae. Indekse veroorsaak normaalweg meer bywerkings om die indeks te onderhou. Indekse help ook tydens bywerking om die rekords in die tabel op te spoor. MySQL se InnoDB- en BDB-tabelle (sien paragraaf 3.3.3.1 se gedeelte oor stoorenjins) vereis dat 'n primêre sleutel vir elke tabel bestaan, en een word geskep as geen primêre sleutel bestaan nie. MySQL se MyISAM-tabel vereis egter nie dat 'n primêre sleutel bestaan nie. SQL Server se tabelle vereis nie 'n primêre sleutel nie. SQL Server bied 'n metode vir indekse om groot kolomme te hanteer vir indekse deur 'n optelsom ("checksum") vir die kolom te bereken en die waarde in die indeks te stoor (Delaney, 2001:884-885). As 'n spesifieke waarde van die groot kolom gesoek word, word die optelsom van die waarde verkry en vergelyk met die optelsom in die indeks om die waarde in die indeks te kry.

Die orde van die kolomme in 'n indeks is belangrik, aangesien die indeks slegs gebruik word indien die linkerkantste kolomme van die indeks in die navraag voorkom. Gestel 'n indeks

bestaan uit die van en naam van 'n werknemer. As die navraag net op die naam toegepas word, sal die indeks nie gebruik word nie. As die navraag die van of die van én die naam in die "WHERE"-gedeelte gebruik, sal die indeks wel gebruik word. Dit is voordelig om die kolom of kolomme wat die meeste rye uitskakel as een van die eerste kolomme in die indeks te gebruik. Die linkerkantste kolomme van die indeks bepaal of die indeks bruikbaar is of nie.

Die verbindingskolomme is gewoonlik goeie kandidate vir indekse. As die primêre sleutel van 'n tabel gebruik word as 'n vreemde sleutel ("foreign key") van 'n ander tabel en die primêre sleutel groot is, kan 'n identiteitskolom vir die verbinding en indeks oorweeg word. Die kleiner identiteitskolom se indeks sal kleiner wees, wat veroorsaak dat meer elemente op die bladsye kan pas en die moontlikheid verhoog dat die elemente in die kasgeheue kan voorkom en bly. Verbinding van heelgetalkolomme is goedkoper as verbindings van karakterkolomme. Geneste herhaalverbindings (sien paragraaf 3.2.7.3.3.1 vir SQL Server se implementering) word gereeld vir verbindings gebruik, maar is eers vanaf MySQL 5.0.1 beskikbaar. Die verbinding kry vir die rye van een tabel die ekwivalente rye van 'n ander tabel (die binnenste tabel). Die binnenste tabel sal bevoordeel word indien bruikbare indekse hierop geskep word. Lei en Ross (1999:180) stel dat verbindings duur operasies is en dat dit belangrik is om op die verbindingskolomme te konsentreer tydens die keuses van indekse. Verbindings is veral duur wanneer die tabelle aansienlike groter as die hoofgeheue van die bediener is.

Die skep en verwyder van indekse is eenvoudige bewerkings. As die indeks nie gebruik word nie, moet die indeks verwyder word om spasie te bespaar. Tydelike indekse kan geskep word vir byvoorbeeld 'n bondeltransaksie, waarna die indeks weer verwyder word.

4.5 Gelyklopendheid ("concurrency") teenoor konsekwentheid ("consistency")

Gelyklopendheid dui op verskeie gebruikers wat gelyktydig navrae aan die databasisbestuurstelsel rig (Delaney, 2001:914). Konsekwentheid bestaan uit isolasievlakke waar die data met verskeie gebruikers dieselfde reageer as wanneer een gebruiker alleen die data gebruik. In die algemeen ding bywerktransaksies en navrae mee om hulpbronne. Die bywerktransaksies vereis normaalweg eksklusiewe (alleenreg-) slotte op die data, terwyl navrae net gedeelde (gewoonlik net leesregte) slotte nodig. Navrae word bevoordeel deur indekse en oorbodigheid, terwyl bywerktransaksies eerder 'n genormaliseerde model met minder indekse sal verkies. Die onderwerpe van gelyklopendheid en konsekwentheid is reeds in paragraaf 3.3.3 aangeroer.

4.6 Versperring ("blocking") en dooiepunte ("deadlock")

Dooiepunte ontstaan wanneer verskeie transaksies slotte op data verkry en twee of meer transaksies dieselfde slotte in verskillende volgordes op die data wil verkry (Balling & Zawodny, 2004:26). 'n Transaksie veroorsaak 'n versperring indien die spesifieke transaksie (eerste transaksie) wag om 'n slot te kry wat deur 'n ander transaksie (tweede transaksie) vasgehou word en die tweede transaksie het nie 'n slot van die eerste transaksie nodig nie (wat 'n ketting en dooiepunt tot gevolg sal hê). MySQL se InnoDB-tabelle bevat ingeboude opsporingstegnieke om hierdie kettings op te spoor en 'n foutboodskap aan die gebruiker oor te dra (Balling & Zawodny, 2004:26-27). Die ander tabelle van MySQL gebruik eindtye ("timeouts") vir die navrae. Delaney (2001:915-916) bied verskeie riglyne om die situasie te vermy:

- Transaksies moet so kort as moontlike gehou word, sodat die hulpbronne so vinnig as moontlik weer beskikbaar gestel kan word.
- Invoer van die gebruiker moet nie binne 'n transaksie verlang word nie. Die gebruiker kan moontlik lank neem om invoer te verskaf, terwyl die spesifieke hulpbronne die hele tyd vasgehou word.
- Verwerk resultate so vinnig as moontlik. As die resultaat nie verwerk word nie, word die bediener verhoed om nog resultate te stuur.

SQL Server spoor dooiepunte outomaties op en beëindig een van die prosesse om sodoende die dooiepunt op te los. Delaney (2001:919) stel dat twee algemene dooiepuntsituasies bestaan, naamlik siklus- ("cycle") en omskakeling-dooiepunte ("conversion"). Die geval van 'n siklus-dooiepunt ontstaan wanneer prosesse mekaar se hulpbronne vashou en elkeen wag vir die ander om die hulpbron te los voor die proses se eie slot vrygelaat word. Die omskakeling-dooiepunt geskied wanneer herhaalbarelees-isolasievlak gebruik word. Die transaksie lees die data terwyl 'n ander transaksie ook probeer om die data te lees. Elkeen van die transaksies bevat 'n bywerkopdrag wat direk op die navraag volg. Die leestransaksies hou dus die hulpbronne vas en die bywerkopdragte kan nie die gesogte eksklusiewe slot verkry nie.

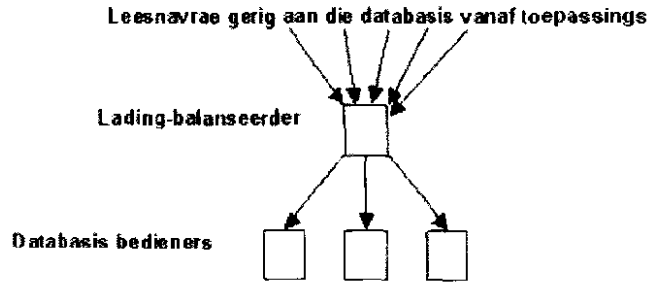
Een van die maniere om dooiepunte te voorkom is om gestoorde prosedures te gebruik. Gestoorde prosedures maak dit makliker om konsekwente reëls te gebruik vir die orde waarin hulpbronne gebruik word. Die voorkoming van dooiepunte verminder gelyklopendheid van transaksies as gevolg van die isolasievlakke wat dalk te hoog raak, en dit kan in sommige gevalle voordelig wees om eerder die dooiepunte te hanteer deur byvoorbeeld die transaksie te herhaal. Atzeni *et al.* (1999:306) stel die gebruik van 'n leeftyd vir transaksies. Sodra die leeftyd om is, word die hulpbronne vrygemaak.

4.7 Ladingbalansering en groepering ("clustering")

Balling en Zawodny (2004:169-170) beskryf ladingbalansering as 'n tegniek met 'n groepering van twee of meer bedieners wat werkslading eweredig deel. Die ladingbalanseerder verdeel inkomende konneksies tussen die bedieners wat nie so besig is nie. Figuur 4.1 dui die tegniek aan. MySQL ondersteun tans nie self hierdie funksionaliteit nie en SQL Server 2000 (en SQL Server 2005) ondersteun net "failover"-groepering, waar een faling van 'n bediener 'n oorskakeling na 'n ander bediener tot gevolg het (Microsoft Corporation, 2005f).

Daar bestaan hardeware om hierdie tipe balansering te behartig. Hierdie hardeware beskik egter nie oor 'n manier om die navrae te sorteer volgens koste nie, en veroorsaak 'n oneweredige verdeling van die lading. Groepering is in paragrawe 3.2.2.6 en 3.3.2 aangeroer.

Figuur 4.1 – Ladingbalansering van databasisse



Vier uitgangspunte van ladingbalansering word genoem:

- "Scalability" – Kapasiteit kan verhoog word deur nog bedieners by te voeg.
- Effektiwiteit – Ladingbalansering gebruik hulpbronne meer effektief omdat versoeke beheer kan word.
- Toeganklikheid ("Availability") – As een bediener ophou funksioneer, neem 'n ander bediener die oop plek in. Die gebruiker sien geen verskil in funksionaliteit nie.
- Deursigtigheid ("Transparency") – Die gebruikers is onbewus van die ladingbalansering-opstelling wat gebruik word.

4.8 Navrae

MySQL se navraagkasgeheue kan geaktiveer word om resultate van navrae te stoor sodat die navraag nie eers vertaal of uitgevoer moet word nie, aangesien die resultaat alreeds beskikbaar is. Die metode het egter geen buigbaarheid nie omdat die navraag se resultate vir akkuraatheid altyd dieselfde moet wees.

Beide MySQL en SQL Server se optimeerders is goeie ontleders en vind normaalweg sonder navraagleidrade (sien paragraaf 3.2.2.1) die beste uitvoerplan. Gevalle bestaan wel waar leidrade beter werkverrigting kan lewer. MySQL ignoreer normaalweg die volgorde van die tabelle in 'n verbindingsnavraag en besluit self in watter volgorde die tabelle verwerk moet word. Die volgorde van verwerking kan met die leidraad "STRAIGHT_JOIN" geforseer word volgens die volgorde in die navraag. Leidrade word gebruik om die verstekprosessering te vervang, en moet nie as die norm gebruik word nie. Die leidrade moet voortdurende gemonitor word om te verseker dat die leidraad nog steeds die beste werkverrigting lewer. MySQL bied 'n leidraad wat veroorsaak dat die resultaat van 'n navraag in 'n tydelike tabel gestoor word, sodat slotte vinniger gelos word (Balling & Zawodny, 2004:97).

Verskeie spesifieke optimeringsleidrade bestaan soos byvoorbeeld vir "union". As geen duplikate in die twee navrae voorkom nie, moet die "Union ALL"-gedeelte gebruik word, om die bykomende koste van die verwydering van die duplikate uit te skakel.

4.9 Hardeware en sagteware

Die moontlikheid bestaan om vanaf 'n databasis die gedeeltes wat meer met bywerktransaksies te make het in 'n aparte databasis te plaas en sodoende te skei van die gedeeltes wat meer met navrae te make het. Dit is een van die beginsels van datapakhuise ("data warehousing"). SQL Server se replikasie kan gebruik word om die twee databasisse te sinkroniseer. Wysigingstransaksies kan teruggehou word om die veranderings aan die databasis te doen wanneer die databasis nie te besig is nie, sodat eksklusiewe slotte verkry kan word. Hierdie metode verhoog produktiwiteit deur bestuur van slotte uit te skakel.

Balling en Zawodny (2004:103-109) beskryf 'n paar beperkende faktore ten opsigte van hardeware, naamlik hardeskywe, geheue en die netwerk.

- Hardeskywe – Die hardeskyf is gewoonlik die stadigste deel van die stelsel. Die meeste tyd word spandeer om die data te vind. Die spoed waarteen die lees/skryf-kop van die hardeskyf beweeg en die volgende data lees, word die soektyd genoem. Die soektyd bestaan uit twee faktore, naamlik die tyd benodig om die lees/skryf-kop van die hardeskyf na die nuwe posisie te beweeg en die skyf se rotasietyd (hoe vinnig die skyf draai). Die tyd wat data dan neem om van die skyf na die geheue oorgedra te word, word die oordragtyd genoem.
- Geheue – Geheue word gebruik as tussenganger tussen die skyf en die SVE. Groter geheue sal 'n goeie opsie wees indien die databasis se grootte nie kleiner as die bestaande geheue is nie. As meer toepassings as net die databasis op die bediener loop, sal die

bediener uitbreidings van die geheue kan benut. Die onderskeie databasisbestuurstelsels se opstellings ten op1sigte van die hoeveelheid geheue wat elkeen vir buffers en kasgeheue-areas gebruik, kan beheer word. Die spoed van die geheue het 'n impak op die toegang van data in die geheue. Sommige geheue-hardeware het ingeboude foutopsporing- en -regstellingfunksies sodat foute in die geheue opgespoor en herstel kan word. 'n Groter kasgeheue in die SVE sal 'n merkbare verskil (tot op 'n sekere punt) in produktiwiteit lewer.

- Netwerk – As die netwerk 'n groot hoeveelheid verkeer moet hanteer, sal die databasisverkeer moet meeding om gebruik van die netwerk. As die spesifieke netwerk onder bespreking aan hoë verborgenheid ("latency") ly, kan die produktiwiteit belemmer word. Verborgenheid kom normaalweg voor by netwerke wat oor groot afstande strek. Dit is voordelig om die bedieners en kliënte so naby as moontlik aan mekaar te hou. Gestel 'n relatief groot navraag word geïmplementeer. Terwyl die resultaat oor die netwerk beweeg, word die hulpbronne vasgehou en kan bywerktransaksies nie plaasvind nie.

Shasha en Bonnet (2003:59) stel dat selfs as die spesifieke hardewarebron oorlaai is, dit nie noodwendig beteken dat die byvoeging van hardeware die probleem sal oplos nie. Shasha en Bonnet (2003:66) noem drie hardeware-verbeterings: geheue, hardeskywe en verwerkers wat tot 'n stelsel toegevoeg kan word. Geheue is die goedkoopste verbetering aangesien meer geheue beteken dat minder hardeskyf-lees- en -skryf-aksies gaan plaasvind.

4.10 Slotparagraaf

Hoofstuk 4 het verskeie aspekte van optimering bespreek. Die aankoop van nuwer hardeware is normaalweg 'n oplossing wat eerstens duur is en tweedens net 'n tydelike oplossing bied. As die databasis van die begin af op die beste moontlike wyse ontwerp is en die regte indekse toegepas word, is die aankoop van beter hardeware normaalweg nie nodig nie. Hierdie hoofstuk sluit die literatuurstudie en teoretiese gedeelte van die verhandeling af. Die volgende twee hoofstukke bespreek die eksperimentele gedeelte, wat hoofsaaklik oor die gebruik van indekse handel.

HOOFSTUK 5

DIE BEDIENEROPSTELLING VIR DIE EKSPERIMENTELE GEDEELTE

5.1 Inleiding

Hoofstuk 4 het riglyne voorgestel wat gebruik kan word om 'n databasisbestuurstelsel te optimeer. As 'n databasis opgestel word, moet hierdie riglyne (maar nie slegs hierdie riglyne nie) in gedagte gehou word. In hierdie hoofstuk word die opstellings van die rekenaar ten opsigte van die spesifikasies van die rekenaar en 'n paar ander algemene punte bespreek.

Die opstelling van die rekenaar en die fisiese komponente van die bediener word in paragraaf 5.2 beskryf. Paragraaf 5.3 beskryf die databasis se opstelling ten opsigte van die tabelle wat gebruik word. Die hoofstuk word in paragraaf 5.4 opgesom.

5.2 Die opstelling van die rekenaar en databasis vir die ondersoek

Die bediener-rekenaar waarop die databasisbestuurstelsel loop se spesifikasies is soos volg:

- Pentium III 600 Megahertz
- 512 Megagreep geheue
- Windows 2000 Advanced Server
- Die spoed van die netwerk is 100 megabisse per sekonde.

Beide SQL Server en MySQL is op die bediener-rekenaar gelaai. Beter resultate word verkry indien die databasis en die kliënt nie op dieselfde rekenaar loop nie. Daarom word die bediener en die kliënt vir die ondersoek geskei. Die rede vir die gebruik van die relatief klein rekenaar is dat die teoretiese beginsels van optimering dieselfde vir enige grootte rekenaar behoort te wees. Die gebruik van die stadiger rekenaar behoort nie die resultate te beïnvloed nie aangesien die verskil in uitvoertyd (en nie die uitvoertyd self nie) vir elke navraag en die interne koste van die prosesse wat elke databasisbestuurstelsel gebruik om die navrae te ontleed en uit te voer die relevante faktore is (sien paragraaf 6.2 vir al die relevante faktore). Die bedryfstelsel Windows 2000 Advanced Server is vir prosesse soos die databasisbestuurstelsel se prosesse geoptimeer, en is 'n goeie keuse vir die ondersoek. Die databasis wat ondersoek word, is nie so groot ten opsigte van skyfspasie nie en baie van die navrae behoort relatief gemaklik gehanteer te kan word. Die spoed van die netwerk sal die uitvoertyd aansienlik vertraag aangesien groot hoeveelhede data lank neem om oor 'n netwerk te vloei. Die impak word egter geïgnoreer aangesien bedieners en kliënte nie normaalweg op dieselfde rekenaar bestaan nie en 'n

netwerk deel van die opstellings is. Hoe groter die hoeveelheid data wat oor die lyn beweeg, hoe langer neem die navraag om in SQL Query Analyzer of MySQL Query Browser te vertoon.

Die ondersoek is ingestel op 'n relatief statiese databasis, wat beteken dat die navrae wat daarop uitgevoer word, nie oor 'n tydperk ondersoek word nie; die databasis is nie deel van 'n werklike produksiestelsel nie. Elke navraag word apart hanteer asof net die een navraag op die databasis plaasvind. As 'n verskeidenheid van navrae tot die databasis gerig word, mag die indekse miskien anders toegepas word. Ten slotte word SQL Profiler met al die navrae en 'n paar bywerkings en byvoegings gevoer om 'n meer verteenwoordigende aanbeveling vir die algemene opstelling van die databasis te kry (sien paragraaf 3.2.8.1 vir beskrywing van SQL Profiler).

Die gebruik van gestoorde prosedures is normaalweg een van die voorgestelde tegnieke van optimering (sien paragraaf 3.2.2.3), waar die prosedure se optimale plan alreeds gevind is en elke keer net gebruik kan word sonder dat die optimale plan weer gesoek moet word. Die huidige weergawe van MySQL besit egter nie gestoorde prosedures nie. SQL Server besit wel die funksionaliteit, maar die tegniek word nie hier toegepas nie, aangesien die tegniek net 'n merkbare verskil op 'n groot hoeveelheid navrae toon en as verskeie keuses van indekse beskikbaar is. Die statiese databasis wat hier ondersoek word, baat nie by gestoorde prosedures nie, aangesien die aantal navrae en indekse beperk is.

5.3 Die opstelling van die databasis se tabelle vir SQL Server en MySQL

Die databasis bestaan uit vyf tabelle. Twee van hierdie tabelle bevat data wat onttrek is uit 'n produksiedatabasis. Die oorblywende tabelle is ontwerp met gegenereerde data vir toetsing. Die tabelle se skepprosedures word in bylaag C gelys. Alle tabelle begin sonder enige indekse. Die gemiddelde grootte van die data-rye word vir elke tabel in SQL Server bereken aangesien die inligting in die SQL Server se eksperimentele gedeelte gebruik word (sien paragraaf 3.2.5.2 en paragraaf 3.2.5.3). Die tabelle se uitleg lyk soos volg:

- **Tabel *internet*** – Die tabel bevat werklike data oor die internetgebruik van 'n gebruiker. Die datum (stygende volgorde), bron-ip-adres, diens ("http" of "ftp"), aantal grepe wat afgelaai is, gebruikersid, "url", webwerf en tyd wat die aflaai geneem het word in die tabel gestoor. Die tabel bevat 338859 rekords.

Die gemiddelde grootte van die *internet-tabel* se data-rye word vir SQL Server soos volg bereken:

Data-rygrootte	=	4 (Statusbis A, statusbis B en die vastelengtegrootte) +
		20 (kolomme *datetime, bytes* en *elapsed* se groottes) +
		3 (Aantal kolomme in totaal en "NULL bitmap") +
		2 (Aantal veranderlikelengtekolomme) +
		10 (Veranderlikelengtekolomme se eindpuntskikking) +
		112 (Gemiddelde grootte van veranderlikelengtekolomme wat verkry is met "SELECT AVG(LEN(1) +...+ LEN(*n*)) FROM internet" waar "..." al die veranderlikelengtekolomme voorstel en 1..*n* die veranderlikelengtekolomme en *n* die toelaatbare aantal kolomme minus die vastelengtekolomme)
	≈	151 grepe

Die minimum aantal databladsye wat vir die stoor van die *internet*-tabel benodig word, kan bereken word. Die berekening word egter net vir die *internet*-tabel in detail gedoen aangesien die berekening dieselfde vir die ander tabelle sal wees. 'n Databladsy bestaan normaalweg uit 8192 grepe (8 * 1024). 'n Gedeelte van 96 grepe word gebruik vir die bladsy-opskrif (sien paragraaf 3.2.5.3). Die oorblywende gedeelte 8096 grepe word vir die data-rye en die data oorsprongskikking gebruik. Die skikking is egter afhanklik van die aantal data-rye wat op die databladsy kan pas. 'n Formule om die aantal data-rye (*n*) wat op die databladsy pas, te bereken, lyk soos volg:

n * 151 + n * 2	=	8096	
n	≈	52	(Aangesien data-rye net vir "text"- en beeldkolomme oor databladsye kan strek)

Die totale grepe wat van 'n databladsy vir die *internet*-tabel vir die data-rye gebruik word, is 7852 grepe (52 * 151 grepe). Die oorsprongskikking beslaan 104 grepe. Aangesien 151 grepe 'n gemiddelde is, kan die waardes moontlik van die werklike waardes verskil.

Die aantal databladsy word verkry deur die aantal rekords in die tabel te vermenigvuldig met die aantal grepe, en dit dan deur 7852 grepe te deel. Die berekening lyk soos volg:

Databladsye	=	338859 * 151 grepe / 7852 grepe
	≈	6516

- **Tabel *persoon*** – Die tabel bevat werklike data oor die gebruikers. Die van, voorletters, voorname, noemname, titels, geboortedatums, taal, aktiewe status en netwerkid word gestoor. Die tabel bevat 69980 rekords.

  Die gemiddelde grootte van die *persoon*-tabel se data-rye word vir SQL Server soos volg bereken:

  Data-rygrootte = 4 (Statusbis A, statusbis B en die vastelengtegrootte) +
  9 (kolomme *geboortedatum* en *aktief* se groottes) +
  3 (Aantal kolomme in totaal en "NULL bitmap") +
  2 (Aantal veranderlikelengtekolomme) +
  14 (Veranderlikelengtekolomme se eindpuntskikking) +
  45 (Gemiddelde grootte van veranderlikelengtekolomme)
  ≈ 77 grepe

  Databladsye = 69980 • 77 / 7854 (n ≈ 102)
  ≈ 686

- **Tabel *tabel1*** – Die tabel het 'n verbindingskolom (verbindkolom) met die netwerkid van die *persoon*-tabel en die gebruikersid van die *internet*-tabel. Die volgende kolom (kolom1) bevat 'n teller wat uniek is. Die derde kolom (kolom2) bevat 'n karakterstring. Die tabel bevat 4741 rekords.

  Die gemiddelde grootte van die *tabel1*-tabel se data-rye word vir SQL Server soos volg bereken:

  Data-rygrootte = 4 (Statusbis A, statusbis B en die vastelengtegrootte) +
  14 (kolomme *kolom1* en *kolom2* se groottes) +
  3 (Aantal kolomme in totaal en "NULL bitmap") +
  2 (Aantal veranderlikelengtekolomme) +
  2 (Veranderlikelengtekolomme se eindpuntskikking) +
  7 (Gemiddelde grootte van veranderlikelengtekolom *verbind-kolom*)
  ≈ 32 grepe

  Databladsye = 4741 * 32 / 7616 (n ≈ 238)
  ≈ 19

- **Tabel *tabel2*** – Die tabel het 'n verbindingskolom (verbindkolom) met kolom1 van *tabel1*. Die volgende kolom (kolom1) bevat 'n reële getal. Die derde kolom (kolom2) bevat 'n karakterstring. Die tabel bevat 2000 rekords.

  Die gemiddelde grootte van die *tabel2*-tabel se data-rye word vir SQL Server soos volg bereken:

  Data-rygrootte = 4 (Statusbis A, statusbis B en die vastelengtegrootte) +
  12 (kolomme *verbindkolom* en *kolom1* se groottes) +
  3 (Aantal kolomme in totaal en "NULL bitmap") +
  2 (Aantal veranderlikelengtekolomme) +
  2 (Veranderlikelengtekolomme se eindpuntskikking) +
  5 (Gemiddelde grootte van veranderlikelengtekolom *kolom2*)
  ≈ 28 grepe

  Databladsye = 2000 * 28 / 7532 (n ≈ 269)
  ≈ 7

- **Tabel *tabel3*** – Die tabel het 'n verbindingskolom (verbindkolom) met kolom1 van *tabel1*. Die volgende twee kolomme (kolom1 en kolom2) bevat karakterstringe. Die tabel bevat 2000 rekords.

  Die gemiddelde grootte van die *tabel3*-tabel se data-rye word vir SQL Server soos volg bereken:

  Data-rygrootte = 4 (Statusbis A, statusbis B en die vastelengtegrootte) +
  259 (kolomme *verbindkolom* en *kolom1* se groottes) +
  3 (Aantal kolomme in totaal en "NULL bitmap") +
  2 (Aantal veranderlikelengtekolomme) +
  2 (Veranderlikelengtekolomme se eindpuntskikking) +
  96 (Gemiddelde grootte van veranderlikelengtekolom)
  ≈ 366 grepe

  Databladsye = 2000 * 366 / 8052 (n = 22)
  ≈ 90

Die tabelle vir MySQL is met MySQL se sintaksis geskep en die data is vanaf SQL Server met behulp van tekslêers oorgedra na MySQL. Die MyISAM en InnoDB-stoorenjins word onderskeidelik gebruik om twee databasisse, een vir elke tipe stoorenjin, in MySQL te skep.

5.4 Slotparagraaf

Hoofstuk 5 het kortliks die opstelling van die rekenaar, die databasisbestuurstelsels en die databasis self beskryf. Die volgende hoofstuk handel oor die praktiese gedeelte van die verhandeling. Verskeie navrae word ondersoek om die gebruik en invloed van indekse te bespreek en te analiseer.

HOOFSTUK 6

EKSPERIMENTE TEN OPSIGTE VAN NAVRAAGKOMBINASIES EN DIE GEPAARDGAANDE RESULTATE

6.1 Inleiding

Hellerstein (1998:147) stel dat implementering en eksperimentering twee kritiese boublokke van navraagoptimering is. Die uitvoerbaarheid van 'n tegniek kan slegs deur die implementering en vergelyking van verskeie optimeringstegnieke gemeet word. Hierdie hoofstuk bevat die eksperimentele gedeelte om die uitvoerbaarheid van indekse as optimeringstegniek te demonstreer.

Hoofstuk 4 het verskillende tegnieke bespreek waarmee navrae en die databasis self geoptimeer word. Die optimeringstegniek wat vir hierdie hoofstuk gebruik word, is hoofsaaklik gekoppel aan indekse. Indekse (sien paragraaf 2.4 vir die teorie gedeelte en paragraaf 4.4 vir die keuse van indekse) is een van die belangrikste tegnieke waarmee navrae geoptimeer kan word. Hoofstuk 5 het die basis gelê vir hierdie hoofstuk deur die opstelling van die bediener te bespreek. Hoofstuk 6 bespreek 'n paar navrae en die resultate ten opsigte van indekse vir beide SQL Server en MySQL (sien hoofstuk 3 vir 'n beskrywing van die twee databasisbestuurstelsels). Die resultate wat verkry is, mag van databasis tot databasis verskil indien verskillende rekenaars of verskillende data gebruik word. Die navrae is tot een databasis (enigste stoorenjin) in SQL Server en twee databasisse (een met MyISAM as stoorenjin en een met InnoDB as stoorenjin) in MySQL gerig.

Die model (of metode) wat vir elke navraag gevolg word, word in paragraaf 6.2 beskryf. Die onderskeie navrae word kortliks in paragraaf 6.3 bespreek en daarna word elkeen se resultate volgens die model in paragraaf 6.2 vertoon.

6.2 'n Oorsig oor die praktiese implementering van die navrae

Die hoofdoel met hierdie praktiese gedeelte is om te bepaal wat die invloed van die gebruik van indekse op die navrae is. Die invloed word gemeet ten opsigte van die verskil in uitvoertyd en die interne koste van die prosesse wat elke databasisbestuurstelsel gebruik om die navrae te verwerk en uit te voer. Twee bevele "CREATE STATISTICS" vir SQL Server (sien paragraaf 3.2.7.3.4.3) en "ANALYZE TABLE" vir MySQL (sien paragraaf 3.3.6.1) kan gebruik word om meer inligting oor die verspreiding van data in tabelle (wat die optimeerder gebruik vir besluite) te verkry. Verskeie transaksies, soos byvoorbeeld bywerkings, kan veroorsaak dat die

statistiese inligting van die databasis onakkuraat raak; in so 'n geval word die "CREATE STATISTICS"- en "ANALYZE TABLE"-bevele gebruik om die inligting weer by te werk sodat die optimeerder akkurate inligting vir gebruik beskikbaar het. Die statiese databasis (sien paragraaf 5.2 en 5.3) wat hier ondersoek word, verander egter nie merkbaar nie, en die inligting behoort redelik akkuraat te wees. Elke navraag word ten minste vyf keer uitgevoer vir elke opsie (kasberging of indeks) wat ondersoek word, sodat gemiddeldes vir die uitvoertyd bereken kan word. Die res van die waardes (soos byvoorbeeld die interne koste van die optimeerder) wat ondersoek word, bly konstant vir elke herhaling van die navraag, maar verskil indien die databasis oorgeskep word of die statistieke van die databasis bygewerk word met die "CREATE STATISTICS"- of "ANALYZE TABLE"-bevele. Die verskille is normaalweg nie groot genoeg om die keuses van indekse te beïnvloed nie.

Elke navraag word eers sonder kasberging en dan mét kasberging uitgevoer. SQL Server gebruik die twee bevele "DBCC DROPCLEANBUFFERS" en "DBCC FREEPROCCACHE" (sien paragraaf 3.2.7.5) om die kasgeheue skoon te maak. MySQL gebruik die "FLUSH TABLES"-bevel (sien paragraaf 3.3.5.1.1), wat die kasgeheue skoonmaak, asook die "SQL_NO_CACHE"-bevel (sien paragraaf 3.3.5.1.1) in die seleksie-sintaksis, wat verhoed dat die navraag se resultaat in die kasgeheue geplaas word.

Die sintaksis om 'n indeks vir MySQL te skep, bied 'n funksie om verskeie indekstipes te kies (MySQL AB, 2005:753). Die twee stoorenjins MyISAM en InnoDB (sien paragraaf 3.3.3.1.3 en 3.3.3.1.5) wat vir die twee databasisse in MySQL gebruik word, kan egter net B-boomindekse gebruik. Die twee stoorenjins word gebruik aangesien MyISAM die verstekstoorenjin is en InnoDB 'n transaksiegerigte stoorenjin is.

Die volgende model waarmee elke navraag ondersoek word, word hier puntsgewys bespreek:

1. Die navraag word met SQL Server (sien paragraaf 3.2) geïnterpreteer.

1.1 Die moontlike indekse, die wenke wat die indeksoptimeringsghoeroe ("Index Tuning Wizard", sien paragraaf 3.2.8.3) maak, en die indeks of indekse wat gekies is, word bespreek. Elke gebondelde indeks word met die verstek-"FILLFACTOR"-waarde (sien paragraaf 2.4.1 en 3.2.6.2) geskep aangesien die databasis as staties beskou word (die indeks se databladsye word volgemaak).

1.2 Die uitvoertyd, aantal rekords en die inligting wat verkry word deur die "STATISTICS IO"- en die "STATISTICS TIME"-opsies (sien paragraaf 3.2.8.2) aan te skakel, word vir die navraag met en sonder kasberging, en met en sonder indekse verkry, en die resultate word in 'n tabel gelys. Die klem word op die SVE-tye (beide vir vertaling- en uitvoertyd) van die navraag geplaas, en die tyd word in die verhandeling as 'n normalisering (of

fraksie) van die navraag sonder kasberging en indekse (wat as basis dien) se SVE-tye vertoon om 'n vergelyking tussen die tye te kan tref. Die normalisering vind plaas deur die SVE-tyd te deel deur die SVE-tyd vir die navraag sonder kasberging en indekse. Die navraag sonder kasberging en indekse behoort die langste SVE-tyd te hê. 'n Normaliseringswaarde word ook vir die uitvoertyd vertoon. Hoe kleiner die normaliseringswaardes, hoe beter is die indeks of kasberging. Die totale uitvoertyd word ook vertoon, maar die waardes kan wissel, afhangende van hoe besig die bediener is. 'n Kort beskrywing van die onderskeie resultate word gegee.

1.3 Die "DBCC SHOW_STATISTICS"-bevel (sien paragraaf 3.2.7.3.2) word gebruik om die digtheid en kardinaliteit van indekse te toon. Digtheid is 'n maatstaf van selektiwiteit (sien paragraaf 3.2.7.3.2).

1.4 Die uitvoerplanne van SQL Server word vir die spesifieke navraag beskryf (sien paragraaf 3.2.8.2). Die waardes van die uitvoerplan met en sonder indekse word in tabelle gelys en die wyse waarop SQL Server die navraag hanteer, word kortliks beskryf. Die koste vir sekere van die waardes word voorgestel as 'n eenheid wat intern deur SQL Server se optimeerder gebruik word. 'n Kort beskrywing van die bruikbaarheid van die indeks word hier hanteer.

1.4.1 Die verbetering in koste word in die verhandeling bereken met die formule: *(geen_indeks_koste_waarde – met_indeks_koste_waarde) / met_indeks_koste_waarde * (-100).* Die formule toon die verhouding van die koste wat die navraag geneem het sonder indekse teenoor die koste wat die navraag geneem het met die indekse. Die (-100)-waarde word gebruik aangesien die koste van die indeks normaalweg 'n verbetering tot gevolg het, wat 'n vermindering in die koste aandui.

2. Die navraag word met MySQL (sien paragraaf 3.3) geïnterpreteer.

2.1 Die moontlike indekse en die indeks of indekse wat gekies is, word bespreek.

2.2 Die uitvoertyd en aantal rekords in die resultaat van die navraag word met en sonder kasberging en met en sonder indekse verkry en die resultate word in 'n tabel gelys. Die uitvoertyd word ook as 'n normalisering (of fraksie) van die navraag sonder kasberging en indekse (wat as basis dien) se uitvoertyd vertoon om 'n vergelyking tussen die uitvoertye te kan tref. Die proses word vir beide 'n MyISAM-databasis en 'n InnoDB-databasis uitgevoer. 'n Kort beskrywing van die resultaat word gegee.

2.3 Die "EXPLAIN"-bevel (sien paragraaf 3.3.6.1) word met en sonder indekse in MySQL Query Browser (sien paragraaf 3.3.6) uitgevoer en die resultate word bespreek. 'n Kort beskrywing van die bruikbaarheid van die indeks word hier hanteer.

Ten slotte word al die navrae, saam met 'n aantal bywerk- en byvoegbewerkings, na mekaar uitgevoer en met SQL Profiler (sien paragraaf 3.2.8.1 en paragraaf 6.4) gemeet om sodoende

die werkverrigting van die databasis te monitor. Die werksladinglêer word aan die einde van al die navrae vir die indeksoptimeringsghoeroe gevoer, wat dan voorstelle ten opsigte van indekse vir die hele databasis maak.

6.3 Die navrae wat ondersoek word

Die navrae wat ondersoek word, bestaan net uit seleksienavrae, aangesien bywerkings dieselfde beginsels volg en byvoegings normaalweg nie die grootste koste van die verskillende bewerkings dra nie. Die aanname is gemaak dat navrae wat meer as 3 sekondes neem, afgerond word na die naaste sekonde, terwyl navrae wat minder as 3 sekonde neem, afgerond word tot drie desimale. Die tye wat vir SQL Server gebruik word, is die SVE-tyd en uitvoertyd volgens die "STATISTICS TIME"-opsie waar die twee verlooptye (wat die SVE-tyd insluit) saam die uitvoertyd lewer. MySQL bied slegs die uitvoertyd. Elke navraag se werklike invoerwaardes word in die verhandeling met letters vervang, aangesien die werklike tabelle (*internet* en *persoon*) met die voorwaarde van vertroulikheid verkry is. Die gemiddelde uitvoertyd vir normale tabeldeursoeke (SELECT * FROM *tabel*) van die verskillende tabelle in die databasis en die maksimum SVE-gebruikpersentasie (wat volgens Windows Task Manager gemeet is), word vir sowel afgeleë navrae[21] as lokale navrae[22] in tabel 6.1 getoon. MySQL se InnoDB-uitvoertyd word in hakies vertoon en die MyISAM-uitvoertyd word normaal vertoon.

Tabel 6.1 – Tabeldeursoek tye (sonder kasberging) vir die tabelle in die databasis.

Tabel	Gemiddelde uitvoertye (en SVE-tyd vir SQL Server) (sekondes)						SVE-persentasie (Windows Task Manager)			
	SQL Server (afgeleë)		SQL Server (lokaal)		MySQL (afgeleë)	MySQL (lokaal)	SQL Server (afgeleë)	SQL Server (lokaal)	MySQL (afgeleë)	MySQL (lokaal)
	SVE	Uitvoertyd	SVE	Uitvoertyd						
internet	1.728	181	2.433	22	140 (151)	586 (590)	6%	22%	20%	100%
persoon	0.196	22	0.381	3	10 (13)	22 (24)	6%	21%	22%	100%
tabel1	0.010	0.393	0.010	0.103	0.144 (0.400)	0.203 (0.258)	8%	9%	5%	31%
tabel2	0.005	0.203	0.004	0.091	0.052 (0.050)	0.056 (0.076)	7%	8%	4%	26%
tabel3	0.023	2.832	0.021	0.171	0.613 (0.846)	0.704 (0.883)	8%	9%	5%	97%

Die uitvoertye dui vir SQL Server aan dat die afgeleë navrae langer neem om uit te voer as die lokale navrae. MySQL neem langer om lokaal uit te voer as gevolg van die invloed van die

[21] Navrae gerig deur 'n ander kliëntrekenaar wat aan die bediener met 'n netwerk verbind is.

[22] Navrae op die bediener self.

MySQL Query Browser wat gereeld ongeveer 95% (volgens Windows Task Manager) hulpbronne van die bediener gebruik het en die MySQL-diens self met onvoldoende hulpbronne los.

MySQL se uitvoertye vir die afgeleë navrae is beter as SQL Server se afgeleë uitvoertye. Die tabel *tabel3* se fisiese grootte dra by tot die langer uitvoertyd, wat navrae op die tabel voortbring. Die resultate wat in die res van die hoofstuk verkry word, sal van afgeleë navrae gebruik maak aangesien die bediener- en kliëntrekenaar oor 'n netwerk gekoppel is. 'n Groot gedeelte van die uitvoertyd kan dus toegeskryf word aan die versending van die resultate oor die netwerk. As 'n groot aantal rekords vertoon word, sal die uitvoertyd langer neem. 'n Voorbeeld hiervan is die uitvoertyd van die *internet*-tabel in tabel 6.1. Die verskil in uitvoertye van byvoorbeeld die navraag in tabel 6.1 op die *internet*-tabel (181 sekondes) en Navraag 1 (op die *internet*-tabel) se uitvoertyd van 26 sekondes in tabel 6.3 (sien paragraaf 6.3.1.1.2) word toegeskryf aan die aantal rekords wat oor die netwerk moet vloei. Die navraag op die *internet*-tabel in tabel 6.1 verkry 338859 rekords, en Navraag 1 in tabel 6.2 verkry net 33586 rekords.

Die presiese SVE-persentasie kan met elke navraag wissel aangesien ander prosesse die SVE van die bediener kan vashou. Die persentasies wat in tabel 6.1 vertoon word, is die maksimum persentasie wat gedurende die vyf herhalings verkry is. Dit is dus beter om na die verskil in persentasies te kyk as na die presiese persentasie. Die bediener verskaf meer hulpbronne aan die lokale navraag as aan die afgeleë navraag aangesien die SVE-persentasie hoër is vir die lokale navrae, wat klop met die hoër SVE-tye vir SQL Server van die lokale navrae. Dit blyk dat MySQL 'n groter SVE-koste vir afgeleë navrae as SQL Server dra, wat klop met die beter uitvoertye van MySQL teenoor SQL Server.

Elke navraag wat hier ondersoek word, toon 'n spesifieke element van optimering (of van die navrae self) aan. Die groep navrae is gekies weens die navraagtipe se algemene gebruik in die praktyk en omdat sekere navrae bronintensiewe navrae is. Die rede vir elke navraag word kortliks beskryf:

- Navraag 1 – Die navraag is hoofsaaklik ter illustrasie om die gebruiker bekend te maak met die model waarvolgens die navrae ondersoek word.
- Navraag 2 – Die navraag is soortgelyk aan Navraag 1 en toon die effek van 'n sorteerargument aan.
- Navraag 3 – Die navraag dui op die invloed wat 'n subnavraag in die navraag sal bewerkstellig.

- Navraag 4 – Die uitwerking van die "union"-argument word met hierdie navraag getoon. Die doel is om aan te toon dat MySQL net met "union" twee indekse van dieselfde tabel kan gebruik.
- Navraag 5 – Die navraag illustreer 'n gekorreleerde subnavraag ("correlated subquery").
- Navraag 6 – Die navraag toon die gebruik van die "GROUP BY"-argument aan.
- Navraag 7 – Die navraag toon 'n verbinding van vyf tabelle aan.

Die navrae wat ondersoek word, word in die volgende gedeelte vollediger bespreek volgens die benaderings van paragraaf 6.2:

6.3.1 Navraag 1

Die eerste navraag is relatief eenvoudig en word gebruik om die leser vertroud te maak met die model wat vir die praktiese aspek van die verhandeling gevolg word.

```
SELECT *
FROM internet
WHERE userid = 'X';
```

6.3.1.1 SQL Server-interpretasie (Navraag 1)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 1) gegee is.

6.3.1.1.1 Moontlike indekse wat toegepas kan word

Die indeksoptimeringsghoeroe (sien paragraaf 3.2.8.3) verskaf 'n nuttige beginpunt vir die skep van indekse. Die indekse wat deur die ghoeroe voorgestel word, moet egter nie noodwendig altyd gevolg word nie. Die indeks wat die ghoeroe vir Navraag 1 voorstel, is 'n gebondelde B-boomindeks op die *userid*-kolom, wat stygend gesorteer is. Die ghoeroe voorspel dat die indeks ongeveer 'n 91%-verbetering op die huidige situasie van geen indekse sal lewer. Die moontlikheid bestaan dat 'n ontbondelde indeks geskep kan word, maar SQL Server kies 'n normale tabeldeursoek bo die gebruik van die ontbondelde indeks. Die SQL Server-navraag is ter illustrasie geforseer (sien "table_hint" in paragraaf 3.3 van bylaag A) om die ontbondelde indeks te gebruik, en die resultaat van die onderskeie gevalle word in tabel 6.2 getoon. Tabel 6.2 is gesorteer volgens die koste.

Tabel 6.2 – Moontlike indekse vir Navraag 1 (SQL Server)

Indeks(e)	Koste	Uitvoertyd (Sekondes)	SVE-tyd (Sekondes)	Logiese Lees-aksies
Gebondelde indeks *<userid>*	0.541	18	0.307	632
Geen indekse (slegs tabeldeursoek)	5.44	26	0.761	6797
Ontbondelde indeks *<userid>*	206	9	0.996	33689

Tabel 6.2 toon vir die ontbondelde indeks 'n koste van 206, wat hoër is as die koste wat verkry word vir 'n normale tabeldeursoek (5.44 volgens tabel 6.5). Die koste steun die besluit van SQL Server om nie die ontbondelde indeks te kies nie. Die uitvoertyd van die ontbondelde indeks is verkry as 9 sekondes, met die SVE-tyd as 0.996 sekondes. 'n Ondersoek is geloods om vas te stel waarom die uitvoertyd soveel beter as beide die tabeldeursoek en die gebondelde indeks is. Verskeie forums is genader (Microsoft Technet (2005), Sql Server Central.Com (2005), Sql Server Performance.Com (2005b) en Sql Team.Com (2005)) vir moontlike verduidelikings vir die verskynsel. Die beste antwoord was dat die stadiger bediener wat gebruik is die navraag anders hanteer as 'n vinniger bediener. Dit wil voorkom of die uitvoertyd (soos in paragraaf 3.2.8.2 genoem is) nie 'n goeie aanduiding van werkverrigting lewer nie, en dat daar eerder na die logiese lees-aksies en SVE-tyd gekyk moet word tydens optimering. 'n Vinniger bediener ("Dual 3.6 Gigahertz" met 4 gigagrepe geheue) lewer 'n SVE-tyd van 0.047 sekondes vir die gebondelde indeks, 0.297 sekondes vir die tabeldeursoek, en 0.203 sekondes vir die ontbondelde indeks. Die uitvoertye reflekteer die SVE-tye waar die gebondelde indeks die vinnigste is. Die logiese lees-aksies was min of meer dieselfde as die stadiger bediener (verskil van ongeveer 10 lees-aksies of minder vir die drie gevalle). Hieruit is dit duidelik dat die gebondelde indeks die beste resultate lewer vir hierdie navraag en dat die SVE-tyd en logiese lees-aksies meer beduidend is as die totale uitvoertyd. Die uitvoertyd is egter vir die gebruiker van belang. Die resultate behoort nader aan die teorie te beweeg indien die navrae se frekwensies toeneem as gevolg van die logiese lees-aksies wat verkry is.

As 'n ander gebondelde indeks alreeds op die tabel geskep was, is dit in hierdie geval beter om geen indeks te skep nie, as gevolg van die SVE-tyd en logiese lees-aksies van die ontbondelde indeks. Aangesien hierdie navraag net 'n voorbeeld is, word die aanname gemaak dat geen gebondelde indeks alreeds vir die *internet*-tabel bestaan nie, en die gebondelde indeks word geskep en gebruik.

Aangesien die logiese lees-aksies die SVE-tyd reflekteer, word slegs die logiese lees-aksies van die tabeldeursoek en die gekose indeks(e) vir elke navraag verder vertoon.

6.3.1.1.2 Die resultaat van die navraag ten opsigte van uitvoertyd, aantal rekords en die "STATISTICS IO"-opsie (Navraag 1)

Die waardes vir die gemiddelde uitvoertyd, die aantal rekords en die inligting verskaf deur die "STATISTICS IO"-opsie word in tabel 6.3 getoon. Dit wil voorkom of die gebruik van kasberging 'n verbetering in uitvoertyd en SVE-tyd lewer (aangedui deur 0.731 en 0.934 sonder indekse, en 0.667 en 0.334 met indekse). Die indeks bied 'n verbetering vir die uitvoertyd (volgens die normaliseringswaardes 0.731 en 0.667) en die SVE-tyd (volgens die normaliseringswaardes 0.403 en 0.334). Kasberging veroorsaak normaalweg dat die vooruitlees-aksies nie plaasvind nie (aangesien die bladsye alreeds in die kasgeheue bestaan) en dat die uitvoerplan nie weer uitgewerk word nie, maar uit die navraagkasgeheue verkry word. Geen fisiese lees-aksies vanaf die skyf het vir die normale tabeldeursoek plaasgevind nie, wat daarop dui dat al die databladsye in die geheue kon kom.

Tabel 6.3 – Die resultaat vir Navraag 1 ten opsigte van uitvoertyd, rekords en "STATISTICS IO".

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	26	19	18	14
Normalisering	1.000	0.731	0.731	0.667
SVE-tyd (sekondes)	0.761	0.711	0.307	0.254
Normalisering	1.000	0.934	0.403	0.334
Aantal rekords	33586	33586	33586	33586
Logiese lees-aksies	6797	6797	632	632
Fisiese lees-aksies	0	0	2	0
Vooruitlees-aksies	6820	0	633	0
Aantal deursoeke	1	1	1	1

Die indeks veroorsaak 'n groot vermindering in die aantal logiese lees-aksies wat moet plaasvind, en 2 fisiese lees-aksies het plaasgevind. Die *internet*-tabel word vir beide die normale tabeldeursoek en indekssoektog een keer deursoek.

Die aantal logiese lees-aksies (6797) is meer as wat die berekende waarde (6516) in paragraaf 5.3 vir die *internet*-tabel toon. Die redes hiervoor is dat die data-rye nie noodwendig direk na mekaar op die databladsye gestoor word nie en dat 'n benaderde gemiddelde waarde vir die data-ry se lengte gebruik is.

6.3.1.1.3 Die resultaat van die "DBCC SHOW_STATISTICS"-bevel (Navraag 1)

Die bevel toon die digtheid vir die indeks op die *userid*-kolom van die *internet*-tabel aan. Die waardes vir die digtheid en die kardinaliteit (wat bereken word as 1 / *digtheid*) word in tabel 6.4 getoon.

Tabel 6.4 – Die digtheid en kardinaliteit vir die indeks(e) in Navraag 1.

Kolom	Digtheid	Kardinaliteit	Gemiddelde Sleutellengte
userid	0.001105	905	5.905

Die kardinaliteit stem ooreen met die waarde wat verkry word vir die "SELECT DISTINCT userid FROM internet"-navraag. Die navraag "SELECT CAST(SUM(LEN(userid)) AS FLOAT) / COUNT(userid) FROM internet" lewer die waarde 5.905, wat ooreenstem met die gemiddelde sleutellengte en die navraag kan gebruik word aangesien die *userid*-kolom nie "NULL"-waardes bevat nie. Die digtheid van 0.001105 dui op 'n hoë selektiwiteit (sien paragraaf 3.2.7.3.2).

6.3.1.1.4 Die uitvoerplan van SQL Server (Navraag 1)

Die eerste uitvoerplan wat bespreek word, is die uitvoerplan wat geen indekse gebruik nie. Tabel 6.5 toon die waardes van die uitvoerplan.

Tabel 6.5 – Die uitvoerplan se waardes vir Navraag 1 vir SQL Server (Geen indekse).

	Fases van uitvoerplan	
Waardes ondersoek	Internet Tabeldeursoek	Seleksie
Fisiese bewerking	Tabeldeursoek	Seleksie
Logiese bewerking	Tabeldeursoek	Seleksie
Verwagte rye	37,006	37,006
Rygrootte	237	N.v.t.
Lees/skryf-aksie-koste	5.07	N.v.t.
SVE-koste	0.372	N.v.t.
Uitvoerings	1	N.v.t.
Koste van fase	5.444476 (100%)	0 (0%)
Subboomkoste	5.44	5.44

Om die navraag te kan hanteer, word die *internet*-tabel vir die kwalifiserende rye deursoek. Twee fases word toegepas, waarvan die eerste 'n tabeldeursoek van die *internet*-tabel is en die tweede 'n seleksie van kolomme is waar geen verdere lees-aksies nodig is nie. Dit is duidelik

dat die lees/skryf-koste van die *internet*-tabel die meeste tot die totale koste bydra. Die totale koste van die navraag is 5.44 (sien punt 1.4 in paragraaf 6.2 vir 'n verduideliking oor die koste).

Tabel 6.6 toon die uitvoerplan se waardes nadat die indeks bygevoeg is. 'n Gebondelde indekssoektog word gebruik om die rekords te verkry. Die lees/skryf-aksie-koste het aansienlik verbeter met die byvoeging van die indeks (van 5.44 na 0.541). Die totale koste na seleksie is 0.541 wat 'n 90% ((0.541 – 5.44) / 5.44 * (-100)) verbetering aantoon. Die verbetering stem ooreen met die vooruitskatting van die indeksoptimeringsghoeroe. Die indeks is dus 'n goeie keuse.

Tabel 6.6 – Die uitvoerplan se waardes vir Navraag 1 vir SQL Server (Met indeks(e)).

	Fases van uitvoerplan	
Waardes ondersoek	**Internet** **Gebondelde Indekssoektog**	**Seleksie**
Fisiese bewerking	Gebondelde Indekssoektog	Seleksie
Logiese bewerking	Gebondelde Indekssoektog	Seleksie
Verwagte rye	33,586	33,586
Rygrootte	230	N.v.t.
Lees/skryf-aksie-koste	0.504	N.v.t.
SVE-koste	0.0370	N.v.t.
Uitvoerings	1	N.v.t.
Koste van fase	0.541870 (100%)	0 (0%)
Subboomkoste	0.541	0.541

6.3.1.2 MySQL-interpretasie (Navraag 1)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 2) gegee is.

6.3.1.2.1 Moontlike indekse wat toegepas kan word

'n B-boomindeks, *internet1*, word in stygende volgorde op die *userid*-kolom van die *internet*-tabel geskep volgens die bespreking wat in paragraaf 6.3.1.1.1 gegee word. Die moontlikheid bestaan ook dat 'n gedeeltelike indeks (sien paragraaf 3.3.4) op die *userid*-kolom geskep mag word. Dit vereis egter dat die lengtes van die waardes in kolom *userid* redelik bekend is en die waardes redelik moet verskil volgens die eerste aantal karakters wat vir die indeks gebruik kan word. 'n Volle indeks word in hierdie geval gebruik, maar 'n gedeeltelike indeks van ongeveer 10 karakters kon ook gebruik word, aangesien die verspreiding van die *userid*-kolom bekend is aan die skrywer. Normaalweg is die verspreiding wat die kolomwaardes gaan besit, nie bekend nie.

6.3.1.2.2 Die resultaat van die navraag ten opsigte van uitvoertyd en die aantal rekords

Die waardes vir die gemiddelde uitvoertyd en die aantal rekords word in tabel 6.7 getoon.

Tabel 6.7 – Die resultaat vir Navraag 1 ten opsigte van uitvoertyd en die aantal rekords

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	15 (16)	10 (15)	10 (14)	10 (13)
Normaliseer	1.000 (1.000)	0.667 (0.938)	0.667 (0.875)	0.667 (0.813)
Aantal rekords	33586	33586	33586	33586

Die uitvoertyd vir die MyISAM-stoorenjin word normaal geskryf en die uitvoertyd vir die InnoDB-stoorenjin word in hakies gegee. Dit blyk uit tabel 6.7 dat kasberging nie 'n groot verskil in uitvoertyd lewer wanneer 'n indeks vir hierdie navraag gebruik word nie. Kasberging lewer 'n beter resultaat ten opsigte van die genormaliseerde uitvoertyd vir die MyISAM-tabel met geen indekse (van 1.000 na 0.667) teenoor die MyISAM-tabel met die indeks wat bygevoeg is (van 0.667 na 0.667). Die navraag op die InnoDB-stoorenjin neem in al die gevalle langer om uit te voer as die navraag op die MyISAM-stoorenjin.

6.3.1.2.3 Die "EXPLAIN"-bevel se resultate met en sonder indekse

Die resultaat van die "EXPLAIN"-bevel vir die navraag met geen indekse word in tabel 6.8 vertoon.

Tabel 6.8 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 1 (Geen indekse)

id	Seleksie-tipe	Tabel	Verbindings-tipe	Moontlike sleutels	Sleutel gebruik	Sleutel-lengte	Verwysing	Verwagte aantal rye	Ekstra
1	SIMPLE	internet	ALL	NULL	NULL	NULL	NULL	338859 (329224)	Using where

Die *seleksie-tipe* is "SIMPLE", wat aandui dat geen subnavrae of "unions" gebruik word nie. Die tabel wat gebruik word, is die *internet*-tabel. Die *verbindingstipe* is "ALL", wat aandui dat 'n tabeldeursoek plaasvind. Die *verwagte aantal rye* wat MySQL verwag om te ontleed om die navraag te bevredig is 338859. Die *verwagte aantal rye* vir die InnoDB-stoorenjin is 329224. Die verskil tussen die *verwagte aantal rye* van MyISAM en InnoDB dui op die vermoë van MyISAM om die tabel se aantal rye te stoor (sien paragraaf 3.3.3) waar InnoDB in hierdie geval 'n skatting moet maak. Die waarde "Using where" in die *ekstra*-kolom dui aan dat 'n "WHERE"-gedeelte gebruik word.

Die resultaat van die "EXPLAIN"-bevel waar indekse geskep is, word in tabel 6.9 vertoon.

Tabel 6.9 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 1 (Met indeks(e)).

Id	Seleksie-tipe	Tabel	Verbindings-tipe	Moontlike sleutels	Sleutel gebruik	Sleutel-lengte	Verwysing	Verwagte aantal rye	Ekstra
1	SIMPLE	internet	ref	internet1	internet1	21	const	32725 (31592)	Using where

Die *seleksie-tipe* is "SIMPLE" wat aandui dat geen subnavrae of "unions" gebruik word nie. Die *verbindingstipe* is "ref" wat aandui dat alle rye met ekwivalente indekswaardes gelees word. Die moontlike sleutel wat gebruik kan word, is *internet1*. Die *internet1* sleutel word wel volgens die *sleutel gebruik*-kolom vir die navraag gekies. Die lengte van die sleutel is 21 (kolom *userid* se lengte is 20 grepe en die ekstra greep dui daarop dat die lengte van die karakterkolom gestoor word). Die waarde in die *verwysing*-kolom is "const" wat aandui dat 'n konstante (X) gebruik word. Die *verwagte aantal rye* wat MySQL moet deursoek, is 32725 (31592 vir InnoDB). Die waarde "Using where" in die *ekstra*-kolom dui aan dat 'n "WHERE"-gedeelte gebruik word.

MySQL se SVE-koste is volgens die Windows Task Manager-toepassing meer as SQL Server se SVE-koste. MySQL gebruik vir hierdie navraag gemiddeld ongeveer 20% en SQL Server gebruik gemiddeld 10% van die SVE se kapasiteit. Die indeks het 'n verbetering ten opsigte van uitvoertyd gelewer (volgens die normaliseringswaarde 0.667 (0.813) in tabel 6.7) en MySQL toon 'n vermindering in die aantal verwagte rye wat vir die resultaat deursoek moet word (32725 (31592) teenoor 338859 (329224) met geen indekse volgens tabel 6.8 en tabel 6.9).

6.3.2 Navraag 2

Die navraag is soortgelyk aan Navraag 1 en toon die effek van 'n sorteerargument aan. Die *datetime*-kolom word dalend gesorteer aangesien die rekords alreeds in stygende volgorde is en die impak van die sorteer nie groot met die stygende opsie sou wees nie.

```
SELECT *
FROM internet
WHERE userid = 'X'
ORDER BY datetime desc;
```

6.3.2.1 SQL Server-interpretasie (Navraag 2)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 1) gegee is.

6.3.2.1.1 Moontlike indekse wat toegepas kan word

Die indeksoptimeringsghoeroe stel 'n indeks op die *userid*-kolom van die *internet*-tabel voor. Die ghoeroe beweer dat 'n 60%-verbetering in produktiwiteit gelewer sal word. Die enigste ander twee indekse wat moontlik is, is 'n indeks op die *datetime*-kolom en 'n indeks op *<userid, datetime>*-kolomme. Die ontbondelde indekse word ook ondersoek en die uitvoertye (sonder kasgeheue), SVE-tye (sonder kasgeheue) en koste word in tabel 6.10 getoon. Dit blyk dat die gebondelde indeks *<userid, datetime>* op grond van die koste die beste keuse in hierdie geval is. Die aanname word egter gemaak dat daar alreeds 'n gebondelde indeks op 'n ander kolom as die kolomme wat in tabel 6.10 voorgestel is. Die beste ontbondelde indeks is die indeks *<userid, datetime>* volgens die SVE-tyd. Dit blyk dat die ontbondelde indeks relatief min hulpbronne in beslag neem ten spyte van die hoë koste. Weens die hoë koste moet die navraag egter geforseer word om die indeks te gebruik.

Tabel 6.10 – Moontlike indekse vir Navraag 2 (SQL Server)

Indeks(e)	Koste	Uitvoertyd (Sekondes)	SVE-tyd (Sekondes)
Gebondelde indeks *<userid, datetime>*	0.54	17	0.270
Gebondelde indeks *<userid>*	2.87	16	0.550
Geen indekse (slegs tabeldeursoek)	7.89	22	1.005
Gebondelde indeks *<userid>* en ontbondelde indeks *<datetime>*	10.4	17	2.030
Ontbondelde indeks *<userid, datetime>*	206	8	0.791
Ontbondelde indeks *<userid>*	208	17	1.479
Ontbondelde indeks *<userid>* en ontbondelde indeks *<datetime>*	212	18	2.023

6.3.2.1.2 Die resultaat van die navraag ten opsigte van uitvoertyd, aantal rekords en die "STATISTICS IO"-opsie (Navraag 2)

Die waardes vir die gemiddelde uitvoertyd, die aantal rekords en die inligting verskaf deur die "STATISTICS IO"-opsie word in tabel 6.11 getoon.

Dit blyk dat die byvoeg en verwyder van die vorige navrae se indekse wel 'n invloed op die resultate lewer aangesien die logiese, fisiese en vooruitlees-aksies vir die navraag sonder indekse verskil van tabel 6.3 se waardes. Die waardes is steeds naby mekaar en die verskil kan geïgnoreer word. Die waarde vir die uitvoertyd sonder indekse en sonder die kasgeheue (22 sekondes) bewys dat die uitvoertyd nie 'n konstante faktor is nie aangesien tabel 6.3 vir Navraag 1 toon dat die navraag 26 sekondes neem, wat nie klop met die bykomende koste nie (van 5.44 na 7.89 (sien tabel 6.13)).

Tabel 6.11 – Die resultaat ten opsigte van uitvoertyd, rekords en "STATISTICS IO" (Navraag 2)

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	22	21	8	10
Normalisering	1.000	0.955	0.364	0.455
SVE-tyd (sekondes)	1.005	0.991	0.791	0.564
Normalisering	1.000	0.986	0.787	0.561
Aantal rekords	33586	33586	33586	33586
Logiese lees-aksies	6726	6726	33723	33722
Fisiese lees-aksies	0	0	3	0
Vooruitlees-aksies	6752	0	761	0
Aantal deursoeke	1	1	1	1

Dit wil voorkom of die gebruik van kasberging nie werklik 'n groot verbetering in uitvoertyd lewer nie (aangedui deur die waardes 1.000 en 0.955 sonder indekse, en die waardes 0.364 en 0.455 met indekse). Kasberging lewer wel 'n verbetering ten opsigte van die SVE-tyd vir die navraag met die indeks (van 0.787 na 0.561). Die indeks bied 'n verbetering vir die uitvoertyd (volgens die normaliseringswaardes 0.364 en 0.455) en vir die SVE-tyd (volgens die normaliseringswaardes 0.787 en 0.561), en is 'n goeie keuse. Die tabel en indeks word net een keer deursoek.

6.3.2.1.3 Die resultaat van die "DBCC SHOW_STATISTICS"-bevel (Navraag 2)

Die bevel toon die digtheid vir die indeks op die *userid*-kolom van die *internet*-tabel aan. Die waardes vir die digtheid en die kardinaliteit (wat bereken word as 1 / *digtheid*) word in tabel 6.12 getoon.

Tabel 6.12 – Die digtheid en kardinaliteit vir die indeks(e) in Navraag 2.

Kolom	Digtheid	Kardinaliteit	Gemiddelde Sleutellengte
userid	0.001105	905	5.905
<userid, datetime>	0.000007	142810	13.905

Die kardinaliteit stem ooreen met die waarde wat verkry word vir die "SELECT DISTINCT userid, datetime FROM internet"-navraag. Die waardes vir die *userid*-kolom stem ooreen met paragraaf 6.3.1.1.3 se tabel 6.4 se waardes. Die sleutellengte van die *<userid, datetime>*-kolom word verkry deur die gemiddelde lengte van *userid*-kolom plus 8 grepe vir die *datetime*-kolom (sien paragraaf 3.2.5.2 vir beskrywing van datatipes).

6.3.2.1.4 Die uitvoerplan van SQL Server (Navraag 2)

Die eerste uitvoerplan wat bespreek word, is die uitvoerplan wat geen indekse gebruik nie. Tabel 6.13 toon die waardes van die uitvoerplan. Die *internet*-tabel word deursoek vir die kwalifiserende rye. Drie fases word toegepas waarvan die eerste fase 'n tabeldeursoek van die *internet*-tabel is en die tweede fase 'n sorteringsfase volgens die *datetime*-kolom. Die tabeldeursoek neem die meeste koste (68%) van die totale koste in beslag. Die sortering neem 32% van die totale koste in beslag. Die totale koste van die navraag is 7.89.

Tabel 6.13 – Die uitvoerplan se waardes vir Navraag 2 vir SQL Server (Geen indekse).

	Fases van uitvoerplan		
Waardes ondersoek	**Internet Tabeldeursoek**	**Sortering**	**Seleksie**
Fisiese bewerking	Tabeldeursoek	Sortering	Seleksie
Logiese bewerking	Tabeldeursoek	Sortering	Seleksie
Verwagte rye	33,586	33,586	33,586
Rygrootte	237	230	N.v.t.
Lees/skryf-aksie-koste	5.02	0.0112	N.v.t.
SVE-koste	0.372	2.32	N.v.t.
Uitvoerings	1	1	N.v.t.
Koste van fase	5.391883 (68%)	2.495252 (32%)	0 (0%)
Subboomkoste	5.39	7.89	7.89

Tabel 6.14 toon die uitvoerplan se waardes nadat die ontbondelde indeks *<userid, datetime>* bygevoeg is.

Tabel 6.14 – Die uitvoerplan se waardes vir Navraag 2 vir SQL Server (Met indeks(e)).

	Fases van uitvoerplan		
Waardes ondersoek	**Internet Indekssoektog**	**"Bookmark Lookup"**	**Seleksie**
Fisiese bewerking	Indekssoektog	"Bookmark Lookup"	Seleksie
Logiese bewerking	Indekssoektog	"Bookmark Lookup"	Seleksie
Verwagte rye	33,586	33,586	33,586
Rygrootte	30	237	N.v.t.
Lees/skryf-aksie-koste	0.104	206	N.v.t.
SVE-koste	0.0371	0.0369	N.v.t.
Uitvoerings	1	1	N.v.t.
Koste van fase	0.141230 (0%)	205.930700 (100%)	0.003360 (0%)
Subboomkoste	0.141	206	206

'n Indekssoektog word gebruik om die rekords te verkry. Die ontbondelde indeks bevat in die blaarnodusse verwysings na die databladsye en daarom besit die tweede fase ("bookmark lookup") die grootste koste (100%).

6.3.2.2 MySQL-interpretasie (Navraag 2)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 2) gegee is.

6.3.2.2.1 Moontlike indekse wat toegepas kan word

Die indekse *<userid, datetime>* (internet2) en *<userid>* (internet1) word geskep. MySQL besluit op die *<userid, datetime>*-indeks. Die *<userid >*-indeks is geforseer en die resultaat (van Tabel 6.14) toon aan dat MySQL die "Using filesort"-argument in die *ekstra*-kolom toon. Dit beteken dat MySQL 'n ekstra rondte moet doen om uit te vind hoe om die rye in gesorteerde volgorde te verkry. Die *<userid, datetime>*-indeks het nie die "Using filesort"-argument nie, wat aandui dat sortering in die indeks plaasvind. Die indeks *<userid, datetime>* word vir die navraag gebruik.

6.3.2.2.2 Die resultaat van die navraag ten opsigte van uitvoertyd en die aantal rekords

Die waardes vir die gemiddelde uitvoertyd en die aantal rekords word in tabel 6.15 getoon. Die uitvoertyd vir die MyISAM-stoorenjin word normaal geskryf en die uitvoertyd vir die InnoDB-stoorenjin word in hakies gegee.

Tabel 6.15 – Die resultaat vir Navraag 2 ten opsigte van uitvoertyd en die aantal rekords

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	11 (10)	11 (10)	11 (13)	11 (13)
Normaliseer	1.000 (1.000)	1.000 (1.000)	1.000 (1.300)	1.000 (1.300)
Aantal rekords	33586	33586	33586	33586

Die indeks lewer nie 'n verbetering in uitvoertye nie en dit lyk of die sorteringsproses die navraag vinniger as Navraag 1 sonder indekse en kasgeheue laat uitvoer (sien paragraaf 6.3.1.2.2). Die indeks word steeds voorgestel aangesien dit vir verskeie kliënte wat Navraag 2 gelyktydig loop 'n verbetering tot gevolg sal hê. Kasberging het nie 'n verbetering op die resultate aangebring nie. Die indeks vir InnoDB was stadiger as die navraag sonder die indeks.

6.3.2.2.3 Die "EXPLAIN"-bevel se resultate met en sonder indekse

Hier volg nou die resultaat van die "EXPLAIN"-bevel vir die navraag op die tabel vir die geval met geen indekse en word in tabel 6.16 vertoon.

Tabel 6.16 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 2 (Geen indekse)

id	Seleksie-tipe	Tabel	Verbindings-tipe	Moontlike sleutels	Sleutel gebruik	Sleutel-lengte	Verwysing	Verwagte aantal rye	Ekstra
1	SIMPLE	internet	ALL	NULL	NULL	NULL	NULL	338859 (343108)	Using where Using filesort

Die *seleksie-tipe* is "SIMPLE" wat aandui dat geen subnavrae of "unions" gebruik word nie. Die tabel wat gebruik word, is die *internet*-tabel. Die *verbindingstipe* is "ALL" wat aandui dat 'n tabeldeursoek gedoen word. Die *verwagte aantal rye* wat MySQL moet ontleed om die navraag te bevredig, is 338859. Die *verwagte aantal rye* vir die InnoDB-stoorenjin is 343108. Die verskil tussen die *verwagte aantal rye* van MyISAM en InnoDB dui op die vermoë van MyISAM om die tabel se aantal rye te stoor (sien paragraaf 3.3.3) waar InnoDB in hierdie geval 'n raaiskoot moet waag. Die waarde "Using where" in die *ekstra*-kolom dui aan dat 'n "WHERE"-gedeelte gebruik word en dat MySQL 'n ekstra rondte deur die data moet gaan om die data te sorteer ("Using filesort").

Die resultaat van die "EXPLAIN"-bevel waar indekse geskep is, word in tabel 6.17 vertoon.

Tabel 6.17 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 2 (Met indeks(e)).

Id	Seleksie-Tipe	Tabel	Verbindings-tipe	Moontlike sleutels	Sleutel gebruik	Sleutel-lengte	Verwysing	Verwagte aantal rye	Ekstra
1	SIMPLE	internet	ref	internet1, internet2	internet2	21	const	32725 (31592)	Using where

Die *seleksie-tipe* is "SIMPLE", wat aandui dat geen subnavrae of "unions" gebruik word nie. Die *verbindingstipe* is "ref", wat aandui dat alle rye met ekwivalente indekswaardes gelees word. Die moontlike sleutel wat gebruik kan word, is *internet1* en *internet2*. Die *internet2*-sleutel word volgens die sleutelkolom vir die navraag gekies. Die lengte van die sleutel is 21 (kolom *userid* se lengte is 20). Die waarde in die *verwysing*-kolom is "const", wat aandui dat 'n konstante (X) gebruik word. Die *verwagte aantal rye* wat MySQL moet deursoek, is 32725 (31592 vir InnoDB). Die waarde "Using where" in die *ekstra*-kolom dui aan dat 'n "WHERE"-gedeelte gebruik word. MySQL toon 'n vermindering in die aantal verwagte rye wat vir die resultaat deursoek moet word: (32725 (31592) teenoor 338859 (343108) met geen indekse).

6.3.3 Navraag 3

Die navraag beskryf die uitwerking van 'n subnavraag op die elemente (byvoorbeeld uitvoertyd) wat gemeet word. Die metode waarmee die subnavraag hanteer word, word vir SQL Server in die uitvoerplan en vir MySQL in die resultaat van die "EXPLAIN"-bevel getoon.

```
SELECT *
FROM internet
WHERE userid IN
(SELECT netwerkid FROM persoon WHERE voorletters = 'X');
```

Die navraag kan ook geskryf word as:

```
SELECT *
FROM internet i JOIN persoon p ON i.userid = p.netwerkid
WHERE voorletters = 'X';
```

Die doel is egter om 'n subnavraag se uitwerking te meet, en nie 'n verbinding nie.

6.3.3.1 SQL Server-interpretasie (Navraag 3)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 1) gegee is.

6.3.3.1.1 Moontlike indekse wat toegepas kan word

Die indekse wat die indeksoptimeringsghoeroe voorstel, is 'n gebondelde indeks op die *userid*-kolom van die *internet*-tabel (stygend) en 'n nie-gebondelde indeks op die twee kolomme <*voorletters, netwerkid*> van die *persoon*-tabel (stygend). Die rede vir die tweekolomindeks is dat beide kolomme vir die navraag direk uit die indeks verkry kan word. Die subnavraag op die *persoon*-tabel se resultaat kan dan net uit die tweekolomindeks verkry word sonder dat data vanaf die skyf gelees moet word. Die ghoeroe stel dat die interne koste met 25% sal verbeter.

Tabel 6.18 toon 'n paar kostes, uitvoertye en SVE-tye vir verskeie indekse wat ondersoek is.

Die ontbondelde indeks op die <*netwerkid, voorletters*>-kolomme se koste (8.06) en SVE-tyd (0.966 sekondes) dui daarop dat die volgorde eerder die *voorletters*-kolom en dan die *netwerkid*-kolom moet wees, aangesien die koste (7.73) en SVE-tyd (0.878) beter is. Die ander kombinasies wat moontlik met hierdie volgorde (*netwerkid* eerste) gemaak kan word, word geïgnoreer.

Tabel 6.18 – Interne koste en uitvoertyd van SQL Server vir verskeie indekse sonder kasberging (Navraag 3)

Indeks(e)	Koste	Uitvoertyd (sekondes)	SVE-tyd (Sekondes)
Gebondelde indeks *<userid>* op *internet*, ontbondelde indeks *<voorletters,netwerkid>* op *persoon*	6.30	8	0.731
Gebondelde indeks *<userid>*	6.97	8	0.798
Ontbondelde indeks *<voorletters,netwerkid>* op *persoon*	7.73	7	0.878
Gebondelde indeks *<voorletters, netwerkid>* op *persoon*	7.80	8	0.878
Ontbondelde indeks *<netwerkid, voorletters>* op *persoon*	8.06	9	0.966
Geen indekse (slegs tabeldeursoeke)	8.39	7	0.958
Ontbondelde indeks *<userid>* op *internet*, ontbondelde indeks *<voorletters,netwerkid>* op *persoon*	461	4	0.419

Die gebondelde indeks op die *userid*-kolom van die *internet*-tabel en die ontbondelde indeks *<voorletters, netwerkid>* lewer die beste koste van 6.30 en die tweede beste SVE-tyd (0.731). Die ontbondelde indekse *<userid>* en *<voorletters, netwerkid>* lewer die swakste koste (461) en die beste SVE-tyd (0.419). Die koste het egter voorkeur in hierdie geval omdat die werkverrigting aansienlik sal verswak indien 'n groot hoeveelheid navrae op die tabel gerig word, aangesien die koste 'n weerspieëling van die hoeveelheid logiese lees-aksies is. Die beste keuse sal dus die gebondelde indeks *<userid>* en die ontbondelde indeks *<voorletters, netwerkid>* wees. As daar alreeds 'n ander gebondelde indeks op die *internet*-tabel bestaan, sal die ontbondelde indeks alleen ook 'n goeie keuse wees. Die gebondelde indeks *<voorletters, netwerkid>* en die ontbondelde indeks *<voorletters, netwerkid>* lewer ongeveer dieselfde resultate. In so 'n geval waar die gebondelde en ontbondelde indekse ekwivalent is, word die ontbondelde indeks gekies aangesien wysigbewerkings makliker op 'n ontbondelde indeks toegepas kan word.

Die gebondelde indeks op die *userid*-kolom is 'n logiese indeks om te skep as die *internet*-tabel beskou word. Die gebondelde indeks *<userid>* en die ontbondelde indeks *<voorletters, netwerkid>* word gebruik.

6.3.3.1.2 Die resultaat van die navraag ten opsigte van uitvoertyd, aantal rekords en die "STATISTICS IO"-opsie (Navraag 3)

Die waardes vir die gemiddelde uitvoertyd, die aantal kwalifiserende rekords en die inligting verskaf deur die "STATISTICS IO"-opsie word in tabel 6.19 getoon.

Tabel 6.19 – Die resultaat ten opsigte van uitvoertyd, rekords en "STATISTICS IO" (Navraag 3)

	Tabel	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	Beide	7	6	8	7
Normalisering	Beide	1.000	0.857	1.286	1.000
SVE-tyd (sekondes)	Beide	0.958	0.855	0.731	0.646
Normalisering	Beide	1.000	0.892	0.763	0.674
Aantal rekords	Beide	8926	8926	8926	8926
Logiese lees-aksies	internet	6726	6726	6642	6642
Fisiese lees-aksies	internet	0	0	2	0
Vooruitlees-aksies	internet	6752	0	6821	0
Aantal deursoeke	internet	1	1	1	1
Logiese lees-aksies	persoon	680	680	9	9
Fisiese lees-aksies	persoon	0	0	2	0
Vooruitlees-aksies	persoon	688	0	8	0
Aantal deursoeke	persoon	1	1	1	1

Die kasberging-aksie ("caching action") lewer 'n klein verbetering in uitvoertyd (vanaf 1.000 na 0.857 sonder indekse, en vanaf 1.286 na 1.000 met indekse). Die indeks lewer 'n verbetering in beide SVE-tyd (volgens die normaliseringswaardes 0.763 en 0.674) en logiese lees-aksies (6726 na 6642 vir die *internet*-tabel en vanaf 680 na 9 vir die *persoon*-tabel). Die verbetering sal beter vertoon indien meer navrae gebruik word aangesien 'n vermeerdering in navrae die databladsye in die geheue sal laat wissel en meer fisiese lees-aksies tot gevolg sal hê soos wat databladsye benodig word. Die indeks het in hierdie geval nie 'n verbetering in die uitvoertyd gelewer nie (1.286 sonder die kasheue en 1.000 met die kasgeheue), maar aangesien die verskil klein is en die uitvoertyd as onakkuraat beskou word, is die resultaat onbelangrik. Die skep van die indeks op die *internet*-tabel het nie so 'n groot verskil in die aantal logiese lees-aksies soos vir die *persoon*-tabel bewerkstellig nie. Dit dui daarop dat die indeks dalk uitgelaat kan word en die effek op die resultaat nie so groot behoort te wees nie (aangedui deur die 7.73 koste en die 0.878 sekondes SVE-tyd van tabel 6.18). Die verskil in die logiese lees-aksies van die *persoon*-tabel met en sonder indekse dui aan dat die indeks 'n goeie keuse is.

6.3.3.1.3 Die resultaat van die "DBCC SHOW_STATISTICS"-bevel (Navraag 3)

Die bevel toon die digthede, kardinaliteite en sleutellengtes vir die twee indekse wat geskep is. Tabel 6.20 vertoon die resultaat van die bevel. Die waardes vir die indeks op die *userid*-kolom van die *internet*-tabel stem ooreen met die waardes wat vir Navraag 1 verkry is (sien paragraaf 6.3.1.1.3). Die kardinaliteit 3396 van die *voorletters*-kolom is die aantal unieke voorletters in die tabel. Die sleutellengtes word met dieselfde selekteerbevel as die bevel in paragraaf 6.3.1.1.3 bereken. Die verskil is dat die lengtes van die *voorletters* en *netwerkid* saamgetel word.

Tabel 6.20 – Die digtheid en kardinaliteit vir die indeks(e) in Navraag 3.

Kolom	Digtheid	Kardinaliteit	Gemiddelde Sleutellengte
userid	0.001105	905	5.905
voorletters	0.000294	3396	1.788
voorletters, netwerkid	0.000014	69980	9.724

6.3.3.1.4 Die uitvoerplan van SQL Server (Navraag 3)

Die uitvoerplan van die navraag met geen indekse op die tabelle word eerste bespreek. Tabel 6.21 toon die waardes van die uitvoerplan.

Tabel 6.21 – Die uitvoerplan se waardes vir Navraag 3 vir SQL Server (Geen indekse).

	Fases van uitvoerplan			
Waardes ondersoek	Persoon Tabeldeursoek	Internet Tabeldeursoek	"Hash Match / Right semi-join"	Seleksie
Fisies	Tabeldeursoek	Tabeldeursoek	"Hash Match"	Seleksie
Logies	Tabeldeursoek	Tabeldeursoek	"Right semi-join"	Seleksie
Verwagte rye	1,530	338,859	84,871	84,871
Rygrootte	85	237	230	N.v.t.
Lees/skryf-aksie-koste	0.540	5.02	0	N.v.t.
SVE-koste	0.0770	0.372	2.34	N.v.t.
Uitvoerings	1	1	1	N.v.t.
Koste van fase	0.617598 (7%)	5.391883 (64%)	2.368701 (28%)	0.008488 (0%)
Subboomkoste	0.617	5.39	8.38	8.39

Die uitvoerplan se persentasies is in totaal 99%. Dit blyk dat SQL Server nie die persentasie vir die seleksiefase korrek weergee nie. Afronding kan moontlik 'n verklaring vir die persentasie wees. Die twee tabelle (*persoon* en *internet*) gebruik tabeldeursoeke om die onderskeie data uit elke tabel te verkry. Die verkrygde data van die *persoon*-tabel word gebruik om 'n "hash"-tabel te bou ("Hash Match"). Elke ry van die verkrygde data in die *internet*-tabel word gebruik om 'n ooreenstemmende ry in die "hash"-tabel te kry ("Right semi-join"). Die twee grootste bydraes tot die navraag se koste is die deursoek van die *internet*-tabel (64%) en die verbinding van die twee tabeldeursoekresultate (28%). Die totale koste is dus 8.39 volgens SQL Server se kostemeting. SQL Server se verwagte aantal rye is aansienlik meer as die werklike aantal rye.

Tabel 6.22 toon die waardes van die uitvoerplan nadat die indekse geskep is. 'n Indekssoektog word toegepas op die *persoon*-tabel. Die resultaat word met 'n groeperingsfunksie verklein en gesorteer volgens die aantal unieke *netwerkid*-waardes ("Stream aggregate / Aggregate") om 'n nuwe tabel te vorm. 'n Gebondelde indekssoektog word op die *internet*-tabel toegepas en die

resultaat word verbind met die verkleinde reeks *netwerkid*-waardes se tabel (saamvoegverbinding/binneverbinding).

Tabel 6.22 – Die uitvoerplan se waardes vir Navraag 3 vir SQL Server (Met indeks(e)).

	Fases van uitvoerplan				
Waardes ondersoek	**Persoon Indekssoektog**	**"Stream Aggregate / Aggregate"**	**Internet Gebondelde Indeksdeursoek**	**Saamvoeg-verbinding / binneverbinding**	**Seleksie**
Fisies	Indekssoektog	"Stream Aggregate"	Gebondelde Indeksdeursoek	Saamvoegverbinding	Seleksie
Logies	Indekssoektog	"Aggregate"	Gebondelde Indeksdeursoek	Binneverbinding	Seleksie
Verwagte rye	1,568	1,568	338,859	73,990	74,034
Rygrootte	27	18	259	240	N.v.t.
Lees/skryf-aksie-koste	0.0100	0	5.07	0	N.v.t.
SVE-koste	0.00180	0.0116	0.372	0.822	N.v.t.
Uitvoerings	1	1	1	1	N.v.t.
Koste van fase	0.011840 (0%)	0.011682 (0%)	5.445217 (86%)	0.822058 (13%)	0.014802 (0%)
Subboomkoste	0.0118	0.0235	5.45	6.29	6.31

Dit lyk of die gebondelde indeks op die *internet*-tabel se lees/skryf-aksie-kostes ongeveer dieselfde as 'n tabeldeursoek se koste is. 'n Moontlike verduideliking is dat al die kolomme van die *internet*-tabel vir die seleksie benodig word. Die groot bydraes tot die koste van die navraag is die indekssoektog van die *internet*-tabel (86%) en die verbindingsfase (13%). Die indekse verskaf ongeveer 'n 25%-verhoging ((6.31 – 8.39) / 8.39 * (-100)) in produktiwiteit ten opsigte van die kostemetings in SQL Server, en die verbetering klop met die ghoeroe se voorspelling van 25%. Die indeks is dus 'n goeie keuse vir hierdie spesifieke navraag.

6.3.3.2 MySQL-interpretasie (Navraag 3)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 2) gegee is.

6.3.3.2.1 Moontlike indekse wat toegepas kan word

'n Paar moontlike indekse bestaan, naamlik 'n indeks op die *internet*-tabel se *userid*-kolom (*internet1)*, 'n indeks op die *voorletters*-kolom (*persoon1)*, 'n indeks op die *netwerkid*-kolom (*persoon2)*, 'n indeks op die <*netwerkid, voorletters*>-kolomme van die *persoon*-tabel (*persoon3)* en 'n indeks op die <*voorletters, netwerkid*>-kolomme (*persoon4)*. Aangesien MySQL net een indeks per tabel kies (sien paragraaf 3.3.4), moet die keuse beperk word tot een indeks per tabel.

Dit blyk dat MySQL nie indekse op die *internet*-tabel vir hierdie navraag gebruik nie (selfs al word die indeksgebruik geforseer). Die "EXPLAIN"-resultaat toon aan dat die navraag die normale tabeldeursoek gebruik ongeag of 'n indeks op die *internet*-tabel bestaan. MySQL kies die *persoon1*-indeks bo die ander indekse. As die *persoon1*-indeks verwyder word, word die *persoon3*-indeks gekies. Tabel 6.23 toon die uitvoertye van die onderskeie indekse. MyISAM word normaal vertoon en InnoDB se waardes word in hakies vertoon.

Die indeks *persoon4* op *<voorletters, netwerkid>* is 'n goeie keuse, aangesien al die data vanaf die indeks vir die subnavraag verkry kan word. Die indeks *persoon3* is volgens MySQL ook 'n goeie keuse, maar die uitvoertyd toon dat die *persoon4* indeks vinniger uitvoer.

Tabel 6.23 – Moontlike indekse vir die *persoon*-tabel vir Navraag 3 sonder kasberging (MySQL)

Indeks	Uitvoertyd (Sekondes)	Verwagte rye vir *persoon*-tabel
<netwerkid> (persoon1)	26 (15)	1 (1)
<netwerkid, voorletters> (persoon3)	23 (14)	1 (1)
<voorletters, netwerkid> (persoon4)	17 (14)	1 (1)
<voorletters> (persoon2)	> 900	1362 (3874)
Geen indekse (slegs tabeldeursoek)	> 900	69980 (70372)

6.3.3.2.2 Die resultaat van die navraag ten opsigte van uitvoertyd en die aantal rekords

Die resultaat ten opsigte van uitvoertyd, normalisering en aantal rekords word vir Navraag 3 in tabel 6.24 vertoon.

Tabel 6.24 – Die resultaat vir Navraag 3 ten opsigte van uitvoertyd en die aantal rekords

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	> 900 (> 900)	> 900 (> 900)	17 (14)	17 (13)
Normaliseer	1.000 (1.000)	1.000 (1.000)	0.019 (0.016)	0.019 (0.014)
Aantal rekords	8926	8926	8926	8926

Die navraag neem sonder indekse langer as 15 minute en is gestaak vir beide stoorenjins. Die gebruik van kasberging het ook nie 'n merkbare invloed op die resultaat nie. Verskeie buffergroottes, soos die verbindingsbuffer ("join_buffer_size"), is verstel om die navraag se spoed te vermeerder, maar dit blyk dat 'n indeks benodig word om die navraag te hanteer. Die aantal rekords wat aan die navraag voldoen, is 8926. Die SVE van die bediener is gemiddeld 98% (volgens "Windows Task Manager) besig vir die volle 15 minute. Die uitvoertyd vir die navraag verbeter met die skep van 'n indeks (van 15 minute (normaliseringswaarde: 1.000) na

17 sekondes (normaliseringswaarde: 0.019) vir MyISAM-stoorenjin en na 13 sekondes (normaliseringswaarde: 0.014) vir die InnoDB-stoorenjin). Dit blyk dat die InnoDB-stoorenjin hierdie navraag beter hanteer as die MyISAM-stoorenjin (normaliseringswaarde van 0.016 teenoor 0.019 sonder kasberging en 0.014 teenoor 0.019 met kasberging).

6.3.3.2.3 Die "EXPLAIN"-bevel se resultate met en sonder indekse

Die navraag sonder indekse se "EXPLAIN"-bevelresultaat word in tabel 6.25 vertoon. Die *internet*-tabel word eerste deursoek en die verwagte aantal rye is 338859, wat die totale aantal rye in die tabel voorstel. Die navraaggedeelte wat op die *internet*-tabel konsentreer (voorgestel deur die eerste ry van die resultaat in tabel 6.25) se seleksie-tipe is "PRIMARY", wat aandui dat die eerste ry in tabel 6.25 die buitenste deel van die navraag beskryf. Al die rekords in die *internet*-tabel word deursoek en die *ekstra*-kolom dui aan dat 'n "WHERE"-gedeelte gebruik word. Sortering word vir die resultaat van die "WHERE"-gedeelte op die *internet*-tabel toegepas (aangedui deur die "Using filesort"-gedeelte in die *ekstra*-kolom).

Tabel 6.25 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 3 (Geen indekse)

id	Seleksie-tipe	Tabel	Verbindings-tipe	Moontlike sleutels	Sleutel gebruik	Sleutel-lengte	Verwysing	Verwagte aantal rye	Ekstra
1	PRIMARY	internet	ALL	NULL	NULL	NULL	NULL	338859 (342106)	Using where Using filesort
2	DEPENDENT SUBQUERY	persoon	ALL	NULL	NULL	NULL	NULL	69980 (70372)	Using where

Die laaste ry van die "EXPLAIN"-resultaat in tabel 6.25 toon dat die navraaggedeelte wat op die *persoon*-tabel konsentreer 'n subnavraag is wat afhanklik is van die buitenste gedeelte ("PRIMARY") van die navraag (aangedui deur die "DEPENDENT SUBQUERY"). Die verbindingstipes ("ALL") dui volledige tabeldeursoeke aan. Die aantal verwagte rye is onderskeidelik 338859 (342106 vir InnoDB) vir die *internet*-tabel en 69980 (70372 vir InnoDB) vir die *persoon*-tabel. Die res van die kolomme in die "EXPLAIN"-resultaat het die waarde "NULL", wat aandui dat geen indekse beskikbaar is nie.

Tabel 6.26 toon die "EXPLAIN"-resultaat vir Navraag 3 met die indeks bygevoeg.

Die verskil vir die bevel se resultaat teenoor die resultaat sonder indekse word getoon in die *verbindingstipe*-kolom in tabel 6.26 vir die "DEPENDENT SUBQUERY" ry. Die *verbindingstipe*-kolom bevat die waarde "ref", wat aandui dat 'n nie-unieke en nie-primêre indeks gebruik word. As meer as een indeks geskep is, sal die *moontlike sleutels*-kolom die moontlike indeks se name bevat. In hierdie geval is net die een geskepte indeks se naam hier aangesien die indeks

Tabel 6.26 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 3 (Met indeks(e))

id	Seleksie-tipe	Tabel	Verbindings-tipe	Moontlike sleutels	Sleutel gebruik	Sleutel-lengte	Verwysing	Verwagte aantal rye	Ekstra
1	PRIMARY	internet	ALL	NULL	NULL	NULL	NULL	338859 (321570)	Using where Using filesort
2	DEPENDENT SUBQUERY	persoon	ref	persoon4	persoon4	34	const, func	1 (1)	Using where Using index

geforseer word. Die *sleutel gebruik*-kolom bevat die geskepte indeks se naam, wat aandui dat die indeks wel gebruik is. Die sleutellengte is 34. Die *verwysing*-kolom bevat die waardes "const, func" wat aandui dat die konstante "X" gebruik word en dat funksies gebruik word om die resulterende netwerkid-waardes saam te groepeer vir die "IN"-argument. Die *verwagte aantal rye*-kolom bevat die waarde 1, wat aansienlik beter as 69980 (soos in tabel 6.25) is. Die inligting vir die *ekstra*-kolom het die waarde "Using index" addisioneel by. Die InnoDB-stoorenjin se verwagte aantal rye is onderskeidelik 321570 en 1.

6.3.4 Navraag 4

Die uitwerking van die "union"-argument word met hierdie navraag getoon. Die doel is om aan te toon dat MySQL wel met "union" twee indekse van dieselfde tabel kan gebruik.

```
SELECT *
FROM persoon
WHERE van LIKE 'A%' AND titel = 'B'
UNION
SELECT *
FROM persoon
WHERE voorname LIKE 'C%' AND aktief = 'D'
```

6.3.4.1 SQL Server-interpretasie (Navraag 4)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 1) gegee is.

6.3.4.1.1 Moontlike indekse wat toegepas kan word

Die ghoeroe stel 'n gebondelde indeks met al die kolomme van die tabel voor wat volgens die ghoeroe 'n 47%-verbetering in produktiwiteit sal lewer. Die indekse wat vir die navraag oorweeg word, word in tabel 6.27 gelys.

Tabel 6.27 – Moontlike indekse vir Navraag 4 in SQL Server (gesorteer volgens koste)

Indeks(e)	Koste	Uitvoertyd (sekondes)	SVE-tyd (sekondes)
Gebondelde indeks op al die kolomme van die *persoon*-tabel	0.777	1.006	0.297
Gebondelde indeks <*voorname, van*>	0.799	1.501	0.300
Gebondelde indeks <*voorname, aktief*>	0.806	1.985	0.307
Gebondelde indeks <*voorname*>	0.808	1.848	0.307
Gebondelde indeks <*van, voorname*>	0.809	1.645	0.314
Gebondelde indeks <*van, titel*>	0.823	1.441	0.291
Gebondelde indeks <*van*>	0.827	1.489	0.277
Geen indekse (slegs tabeldeursoeke)	1.47	1.535	0.377
Gebondelde indeks <*voorname, aktief*>, ontbondelde indeks <*van,titel*>	6.47	2.827	0.902
Gebondelde indeks <*van, titel*>, ontbondelde indeks <*voorname,aktief*>	8.3	4	2.000
Ontbondelde indeks <*voorname, aktief*>, ontbondelde indeks <*van, titel*>	14.7	3	1.439
Gebondelde indeks <*voorname*>, ontbondelde indeks <*van*>	16.9	4	1.656
Ontbondelde indeks <*voorname*>, ontbondelde indeks <*van*>	29.7	5	1.622
Ontbondelde indeks <*voorname, van*>	30.1	2.367	0.955
Ontbondelde indeks <*van,voorname*>	30.1	4	1.639

SQL Server kies nie een van die ontbondelde indekse nie aangesien hul koste meer as normale tabeldeursoeke (1.47) kos. Die navraag met die gebondelde indeks op al die kolomme bied wel die minste koste (0.776), maar is onprakties in dié opsig dat die oorhoofse koste van bywerkings onaanvaarbaar sal wees as bywerkings gereeld op die indeks toegepas word. Die kostes van die gebondelde indekse is meestal minder as die kostes van die ontbondelde indekse (slegs die ontbondelde indekse op <*voorname, aktief*> en <*van, titel*> lewer 'n koste wat beter as 'n gebondelde-indeks-kombinasie is). As 'n gebondelde indeks (wat nie een van die kolomme onder bespreking bevat nie) alreeds op die *persoon*-tabel bestaan, is dit beter om in hierdie geval geen indeks te skep nie. As geen gebondelde indeks alreeds bestaan nie, is die beste keuses die <*voorname, van*>-indeks, die <*voorname, aktief*>-indeks of 'n indeks net op die *voorname*-kolom.

Die indeks op die *van*-kolom lewer wel die beste SVE-tyd, maar in hierdie geval is die tye so naby aan mekaar dat daar nie behoorlik onderskeid getref kan word nie, en word die besluit eerder volgens die koste bepaal.

As gevolg van die klein aantal rekords wat verkry word (2898 volgens tabel 6.28 in die volgende paragraaf), is indekse onnodig aangesien die gewone tabeldeursoek 'n goeie resultaat lewer ten opsigte van koste (sien paragraaf 6.3.4.1.4) en uitvoertyd (sien tabel 6.28). As die tabel egter vergroot, sal indekse wel 'n rol speel. Die gebondelde <*voorname, van*>-indeks word vir illustrasiedoeleindes toegepas aangesien die indeks beide seleksiedele van die navraag bevoordeel.

6.3.4.1.2 Die resultaat van die navraag ten opsigte van uitvoertyd, aantal rekords en die "STATISTICS IO"-opsie (Navraag 4)

Die waardes vir die gemiddelde uitvoertyd, die aantal rekords, en die inligting verskaf deur die "STATISTICS IO"-opsie word in tabel 6.28 getoon. Die kasgeheue veroorsaak 'n verbetering in die uitvoertyd (vanaf 1.000 na 0.618 sonder indekse, en vanaf 0.978 na 0.528 met indekse) en SVE-tyd (vanaf 1.000 na 0.682 sonder indekse en vanaf 0.796 na 0.485 met indekse). Die indeks verminder die logiese lees-aksies van 1360 na 707, wat aandui dat die indeks 'n goeie keuse is. Die feit dat min rekords verkry is veroorsaak dat die verskil in uitvoertyd en koste tussen 'n tabeldeursoek en die indeks nie so groot is nie, maar die lees-aksies dui wel die verskil aan. Die *persoon*-tabel word hier twee keer gelees.

Tabel 6.28 – Die resultaat vir Navraag 4 ten opsigte van uitvoertyd, aantal rekords en "STATISTICS IO".

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	1.535	0.949	1.501	0.811
Normalisering	1.000	0.618	0.978	0.528
SVE-tyd (sekondes)	0.377	0.257	0.300	0.183
Normalisering	1.000	0.682	0.796	0.485
Aantal rekords	2898	2898	2898	2898
Logiese lees-aksies	1360	1360	707	707
Fisiese lees-aksies	0	0	2	0
Vooruitlees-aksies	688	0	683	0
Aantal deursoeke	2	2	2	2

6.3.4.1.3 Die resultaat van die "DBCC SHOW_STATISTICS"-bevel (Navraag 4)

Die bevel toon die digthede, kardinaliteite en sleutellengtes vir die indeks wat geskep is. Tabel 6.29 vertoon die resultaat van die bevel.

Tabel 6.29 – Die digtheid en kardinaliteit vir die indeks(e) in Navraag 4.

Kolom	Digtheid	Kardinaliteit	Gemiddelde Sleutellengte
voorname	0.000019	51553	12.862
voorname, van	0.000015	68361	19.987

Die kardinaliteit van die *voorname*-kolom is 51553 en die kardinaliteit van beide die *voorname*- en *van*-kolomme is 68361. Die sleutellengtes is onderskeidelik 12.862 en 19.987, wat ongeveer aandui hoeveel grepe die kolom(me) in die indeks beslaan (sonder opskrifinligting ("header")).

6.3.4.1.4 Die uitvoerplan van SQL Server (Navraag 4)

Die uitvoerplan se waardes vir die navraag met geen indekse word in tabel 6.30 vertoon.

Die *persoon*-tabel ondergaan twee normale tabeldeursoekherhalings om die rekords te verkry. Die rekords word daarna bymekaargevoeg om een tabel te gee. Die duplikate word verwyder met 'n "distinct sort" fase (sien paragraaf 3.2.8.2) wat die rekords sorteer in volgorde van al die kolomme. Die waardes van hierdie fases word in tabel 6.30 vertoon. Die verskillende herhalings vir die tabel word voorgestel deur *persoon 1* en *persoon 2*.

Tabel 6.30 – Die uitvoerplan se waardes vir Navraag 4 vir SQL Server (Geen indekse)

	Fases van uitvoerplan				
Waardes ondersoek	**persoon 1 Tabeldeursoek**	**persoon 2 Tabeldeursoek**	**Samevoeging**	**Sortering / Unieke sortering**	**Seleksie**
Fisies	Tabeldeursoek	Tabeldeursoek	Samevoeging	Sortering	Seleksie
Logies	Tabeldeursoek	Tabeldeursoek	Samevoeging	Unieke Sortering	Seleksie
Verwagte rye	1,148	1,299	2,447	2,447	2,447
Rygrootte	85	110	104	104	N.v.t.
Lees/skryf-aksie-koste	0.540	0.540	0	0.0112	N.v.t.
SVE-koste	0.0770	0.0770	0.000245	0.0430	N.v.t.
Uitvoerings	1	1	1	1	N.v.t.
Koste van fase	0.617598 (42%)	0.617598 (42%)	0.179394 (12%)	0.054346 (4%)	0 (0%)
Subboomkoste	0.617	0.617	1.41	1.47	1.47

Dit blyk dat die tabeldeursoeke die grootste bydrae tot die koste behels. Indekse behoort die koste te verklein. Dit is interessant om daarop te let dat die sortering nie 'n groot koste beloop nie. Dit het waarskynlik te make met die beperkte aantal rekords. Dit blyk dat daar wel duplikate in die navraag verwyder is. Die boonste seleksiegedeelte van die navraag lewer 1411 rekords en die onderste navraag lewer 1511 rekords. Gesamentlik moet die navraag met duplikate dus 2922 rekords lewer. Die unieke sorteringsfase verwyder dus 24 rekords. Die koste van die seleksie is nie werklik nul nie, maar die waarde is weglaatbaar klein.

Tabel 6.31 vertoon die uitvoerplan se waardes nadat die indeks bygevoeg is.

'n Gebondelde indeksdeursoek word op die boonste seleksiegedeelte van die navraag toegepas en 'n gebondelde indekssoektog word op die onderste seleksiegedeelte van die navraag toegepas. Dit blyk duidelik dat die indekssoektog 'n kleiner koste dra as die indeksdeursoekproses. Die twee resultate word saamgevoeg in een tabel; daarna word dit gesorteer en word die duplikate verwyder.

Tabel 6.31 – Die uitvoerplan se waardes vir Navraag 4 vir SQL Server (Met indeks(e))

	Fases van uitvoerplan				
Waardes ondersoek	**Persoon 1 Indeksdeursoek**	**Persoon 2 Gebondelde Indekssoektog**	**Samevoeging**	**Sortering / Unieke sortering**	**Seleksie**
Fisies	Gebondelde Indeksdeursoek	Gebondelde Indekssoektog	Samevoeging	Sortering	Seleksie
Logies	Gebondelde Indeksdeursoek	Gebondelde Indekssoektog	Samevoeging	Unieke Sortering	Seleksie
Verwagte rye	1,163	1,640	2,804	2,804	2,804
Rygrootte	135	160	104	104	N.v.t.
Lees/skryf-aksie-koste	0.540	0.0241	0	0.0112	N.v.t.
SVE-koste	0.0770	0.00287	0.000280	0.0502	N.v.t.
Uitvoerings	1	1	1	1	N.v.t.
Koste van fase	0.617598 (77%)	0.026980 (3%)	0.093086 (12%)	0.061463 (8%)	0 (0%)
Subboomkoste	0.617	0.0269	0.737	0.799	0.799

Die indeks se uitvoerplan toon 'n verbetering in die koste van ongeveer 46% ((0.799 – 1.47) / 1.47 * (-100)), wat 'n redelike groot verbetering is (die ghoeroe voorspel 47%).

6.3.4.2 MySQL-interpretasie (Navraag 4)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 2) gegee is.

6.3.4.2.1 Moontlike indekse wat toegepas kan word

Die belangrike punt word gemaak dat MySQL net een indeks per tabel in 'n navraag gebruik behalwe as 'n "UNION"-bewerking plaasvind. Agt indekse is gelyktydig op die MySQL-databasis getoets, naamlik *<voorname>*, *<van>*, *<voorname; aktief>*, *<van; titel>*, *<voorname, van>*, *<van, voorname>*, *<voorname, aktief, van, titel>* en 'n indeks op al die kolomme van die tabel. Die indekse se name is *persoon1* tot by *persoon8* onderskeidelik. Geen indekse kon as uniek (en dus gebondel) geskep word nie, aangesien duplikate wel in die kolomme voorkom. Die indekse, uitvoertye en verwagte aantal rye word in tabel 6.32 getoon.

Die indeks wat vir die boonste seleksiegedeelte van die "UNION" gekies is, word eerste genoem en daarna die indeks vir die onderste seleksiegedeelte. MySQL kies net 'n indeks indien die indeks se beginkolom in die seleksiegedeelte se "WHERE"-gedeelte (of ander soortgelyke gedeeltes) voorkom. Dus is net *persoon2, persoon4, persoon6* en *persoon8* beskikbaar vir die eerste seleksiegedeelte, en die res is beskikbaar vir die tweede seleksiegedeelte. Die indekse *persoon7* en *persoon8* is onprakties as wysigings gereeld op die tabelle plaasvind, en word net ter illustrasie in tabel 6.32 bygevoeg. Die eerste indeks van elke ry in tabel 6.32 word vir die

Tabel 6.32 – MySQL se keuses van die agt beskikbare indekse vir Navraag 4

Indeks(e)	Uitvoertyd (sekondes)	Verwagte rye (Boonste seleksie)	Verwagte rye (Onderste seleksie)
<van; titel> (*persoon4*), *<voorname; aktief>* (*persoon3*)	1.028 (0.601)	3000 (7152)	2496 (6502)
Al die kolomme van *persoon*-tabel (*persoon8*), *<voorname, aktief, van, titel>* (*persoon7*)	1.148 (0.313)	3346 (6622)	1631 (3776)
<van> (*persoon2*), *<voorname>* (*persoon1*)	0.933 (0.810)	4727 (9234)	2358 (5184)
<van> (*persoon2*), geen indeks op die tweede *persoon* -tabel nie	0.774 (0.777)	4727 (9234)	69980 (70248)
<van, voorname> (*persoon6*), *<voorname, van>* (*persoon5*)	0.713 (1.009)	5233 (7360)	1631 (3880)
Geen indekse (slegs tabeldeursoeke)	1.030 (1.000)	69980 (69937)	69980 (69937)
Geen indeks op die eerste *persoon*-tabel nie, *<voorname>* (*persoon1*)	1.192 (1.015)	69980 (70605)	2358 (5184)

eerste seleksiegedeelte van Navraag 4 gebruik, en die tweede indeks van dieselfde ry word vir die tweede seleksiegedeelte gebruik.

Die uitvoertye is naby aan mekaar en wissel vir elke herhaling van die navraag; daarom word meer na die verwagte aantal rye vir die keuse van die indeks(e) gekyk. MySQL kies uit die agt indekse die indeks op al die kolomme (*persoon8*) vir die boonste seleksiegedeelte (MyISAM se verwagte rekords is 3346 en InnoDB se verwagte rekords is 6622). Die tweede grootste indeks (*persoon7*) word vir die onderste seleksiegedeelte van die navraag (1631 vir MyISAM en 3776 vir InnoDB) gekies. Dit wil dus voorkom of hierdie indekse volgens MySQL die beste indekse is; die indekse is egter net ter illustrasie. Die MyISAM-getal is meer akkuraat aangesien die presiese totaal van die rekords bekend is, terwyl InnoDB hier 'n skatting moet maak.

Die indeks op die *<voorname, van>*- en *<van,voorname>*-kolomme word nie gebruik nie aangesien die indekse se doel is om een indeks te kry wat deur beide seleksiegedeeltes van die navraag gebruik word. Die bepaling vir MySQL is dat daar na die eerste kolom van die indeks in die navraag verwys moet word voordat die indeks gebruik word. Nie een van die indekse word deur beide seleksiegedeeltes verwys nie, en die indekse is dus nie goeie keuses nie.

Die indekse op *<voorname, aktief>* en *<van,titel>* word ook weggelaat aangesien die kolomme *aktief* en *titel* nie 'n groot verskeidenheid lewer in die moontlike waardes wat die kolomme kan aanneem nie en aangesien vergelykbewerkings minder koste dra as "LIKE"-bewerkings. Die indekse op onderskeidelik die *voorname*- en *van*-kolomme blyk dus die beste opsie vir hierdie navraag in MySQL te wees, en die ander indekse word verwyder.

6.3.4.2.2 Die resultaat van die navraag ten opsigte van uitvoertyd en die aantal rekords

Tabel 6.33 toon die resultate vir Navraag 4.

Tabel 6.33 – Die resultaat vir Navraag 4 ten opsigte van uitvoertyd en die aantal rekords

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	1.030 (1.000)	0.688 (0.345)	0.933 (0.810)	0.995 (1.032)
Normaliseer	1.000 (1.000)	0.668 (0.345)	0.906 (0.810)	0.966 (1.032)
Aantal rekords	2898	2898	2898	2898

Kasberging lewer 'n beter uitvoertyd (0.688 vir MyISAM en 0.345 vir InnoDB) sonder indekse, en 'n verswakking met indekse (van 0.966 vir MyISAM en 1.032 vir InnoDB). Die kort uitvoertye dui aan dat die indekse eintlik onnodig is vir hierdie navraag en dat kasgeheue voldoende is.

6.3.4.2.3 Die "EXPLAIN"-bevel se resultate met en sonder indekse

Die "EXPLAIN"-bevel word eerste ondersoek vir 'n navraag sonder indekse. Tabel 6.34 toon die resultaat van die bevel.

Tabel 6.34 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 4 (Geen indekse).

Id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	PRIMARY	persoon	ALL	NULL	NULL	NULL	NULL	69980 (69937)	Using where
2	UNION	persoon	ALL	NULL	NULL	NULL	NULL	69980 (69937)	Using where
NULL	UNION RESULT	<union1,2>	ALL	NULL	NULL	NULL	NULL	NULL	

Drie rye word as resultaat in tabel 6.34 vertoon. Die laaste ry het nie 'n id nie, aangesien dit die "union" se resultaat is. Die *seleksie-tipe* vir die eerste ry is "PRIMARY" wat aandui dat die ry die boonste gedeelte van die "union"-bewerking voorstel. Die *seleksie-tipe* van die tweede seleksiegedeelte van die navraag is "UNION", wat aandui dat die navraag in 'n "union"-struktuur gebruik is. Die *tabel*-kolom se waarde van die derde ry is <union1,2>, wat die saamgevoegde tabel van die eerste en tweede gedeeltes van die "union" voorstel. Die *tipe* van al die rye is ALL, wat beteken dat tabeldeursoeke vir die eerste twee rye van die "EXPLAIN"-resultaat en 'n volledige deursoek van die "union"-resultaat toegepas is. Die aantal verwagte rye vir beide tabeldeursoeke is 69980 (69937 vir InnoDB). Die "union"-resultaat-ry het "NULL" in die *verwagte aantal rye*-kolom, wat aandui dat MySQL nie inligting besit om 'n skatting te maak nie. Die *ekstra*-kolom vir die eerste en tweede ry van die bevel het die waardes "Using where", wat aandui dat 'n "WHERE"-gedeelte gebruik is.

Tabel 6.35 toon die "EXPLAIN"-bevel se resultaat nadat die indekse bygevoeg is.

Tabel 6.35 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 4 (Met indeks(e)).

Id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	PRIMARY	persoon	range	persoon2	persoon2	32	NULL	4727 (9234)	Using where
2	UNION	persoon	range	persoon1	persoon1	47	NULL	2358 (5184)	Using where
NULL	UNION RESULT	<union1,2>	ALL	NULL	NULL	NULL	NULL	NULL	

Die waarde "range" in die *tipe*-kolom dui daarop dat 'n indeks gebruik word om 'n reeks waardes ("LIKE") te kry. Die verwagte rye is onderskeidelik 4727 (9234 vir InnoDB) en 2358 (5184 vir InnoDB). Die indeks het die verwagte aantal rye verminder van 69980 (69937 vir InnoDB) na onderskeidelik 4727 (9234 vir InnoDB) en 2358 (5184 vir InnoDB), wat aandui dat die indeks 'n goeie verbetering is aangesien daar nie baie bywerkings plaasgevind het wat die statistiese inligting van die databasis kon affekteer nie. Dit is normaalweg beter om eers die statistiese inligting van die databasis by te werk voor die keuse oor indekse gemaak word.

6.3.5 Navraag 5

Die navraag illustreer 'n gekorreleerde subnavraag ("correlated subquery"). Die subnavraag het 'n verwysing na 'n tabel wat buite die subnavraag bestaan (MySQL AB, 2005:748). Die navraag lyk soos volg:

```
SELECT *
FROM persoon p1
WHERE voorletters IN
        (SELECT p2.voorletters FROM persoon p2 WHERE p2.netwerkid = p1.netwerkid)
```

Die navraag kan geskryf word as 'n verbinding (Navraag 5.1):

```
SELECT *
FROM persoon p1
JOIN persoon p2 ON p1.netwerkid = p2.netwerkid AND p1.voorletters = p2.voorletters
```

Die doel is egter om die hantering van 'n gekorreleerde subnavraag te ondersoek. Die gebruik van dieselfde tabel in hierdie navraag se enigste uitwerking is dat slegs een gebondelde indeks geskep kan word. Verder word die navraag soos enige ander gekorreleerde subnavraag hanteer. Die navraag is ekwivalent aan 'n normale "SELECT * FROM persoon" ten opsigte van

die rekords wat verkry word. Normaalweg is tabelle buite die subnavraag nie dieselfde tabel as die tabelle in die subnavraag nie, maar die gevalle word dieselfde hanteer. Nog 'n voorbeeld van so navraag is:

```
SELECT *
FROM tabel1 t1
WHERE kolom2 IN
        (SELECT kolom2 FROM tabel2 t2 WHERE t2.verbindkolom = t1.kolom1)
```

Die laaste navraag lewer egter net 2 rekords en word geïgnoreer.

6.3.5.1 SQL Server-interpretasie (Navraag 5)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 1) gegee is.

6.3.5.1.1 Moontlike indekse wat toegepas kan word

Die ghoeroe stel in hierdie geval geen indeks voor nie. 'n Goeie indeks in hierdie geval is 'n indeks op beide kolomme (*netwerkid* en *voorletters*) aangesien dit 'n oordekkingsindeks (sien paragraaf 3.2.7.3.2) lewer. Die enigste aspek wat vir die oordekkingsindeks bepaal moet word, is of die indeks gebondel of ontbondel moet wees en ook die volgorde van die kolomme. 'n Gebondelde indeks op die *voorletters*-kolom is ook ondersoek. Die ontbondelde oordekkingsindekse word nie vir die *persoon*-tabel *p1* geforseer nie aangesien die koste en SVE-tyd te hoog sal wees (soos gesien vir vorige navrae met ontbondelde indekse). Tabel 6.36 toon die moontlikhede vir die navraag (die uitvoertye en SVE-tyd word sonder kasberging getoon).

Die gevalle waar een indeks 'n gebondelde indeks is op een kolom (*netwerkid* of *voorletters*) en die ander indeks 'n ontbondelde indeks op die ander kolom is onprakties in dié opsig dat as daar nie alreeds 'n gebondelde indeks op die *persoon*-tabel bestaan nie, sal die gebondelde indeks eerder in hierdie geval op beide kolomme geplaas word om 'n oordekkingsindeks te vorm. Die voordeel van die gebondelde indekse is dat die indeks vir beide die subnavraag en die seleksiegedeelte buite die subnavraag gebruik kan word. Die volgorde van indekse kan nie uit die resultate van tabel 6.36 bepaal word nie aangesien die waardes van die SVE-tye, koste en logiese lees-aksies so naby aan mekaar vir die oordekkingsindekse is. Die volgorde word gekies as *<netwerkid, voorletters>*. As geen gebondelde indekse op die tabel bestaan nie, kan 'n gebondelde indeks *<netwerkid, voorletters>* geskep word. As 'n gebondelde indeks reeds op

die *netwerkid*- of *voorletters*-kolom bestaan is dit moontlik om net die ander kolom by die indeks te voeg.

Tabel 6.36 – Moontlike indekse vir Navraag 5 (SQL Server)

Indeks(e)	Koste	Uitvoertyd (sekondes)	SVE-tyd (sekondes)
Gebondelde indeks <*netwerkid, voorletters*>	2.09	3	1.125
Gebondelde indeks <*voorletters, netwerkid*>	2.09	4	1.145
Gebondelde indeks <*netwerkid*>, ontbondelde indeks <*netwerkid, voorletters*>	3.09	5	1.472
Gebondelde indeks <*voorletters*>, ontbondelde indeks <*netwerkid, voorletters*>	3.15	5	1.553
Ontbondelde indeks <*netwerkid, voorletters*>	3.17	6	1.519
Ontbondelde indeks <*voorletters, netwerkid*>	3.17	7	1.492
Gebondelde indeks <*van*>, ontbondelde indeks <*netwerkid, voorletters*>	3.19	5	1.476
Gebondelde indeks <*netwerkid*>	3.46	6	1.422
Geen indekse (slegs tabeldeursoeke)	3.46	6	1.425
Gebondelde indeks <*voorletters*>	3.50	6	1.447

Die aanname word gemaak dat 'n gebondelde indeks reeds op 'n ander kolom van die *persoon*-tabel bestaan (byvoorbeeld die *van*-kolom). Tabel 6.36 toon egter dat die ontbondelde indeks <*netwerkid, voorletters*> alleen 'n beter resultaat ten opsigte van koste en lees-aksies lewer. Die gebondelde indeks bestaan egter volgens die aanname, en die ontbondelde indeks <*netwerkid, voorletters*> word gebruik.

6.3.5.1.2 Die resultaat van die navraag ten opsigte van uitvoertyd, aantal rekords en die "STATISTICS IO"-opsie (Navraag 5)

Die waardes vir die gemiddelde uitvoertyd, die aantal rekords en die inligting verskaf deur die "STATISTICS IO"-opsie word in tabel 6.37 getoon.

Dit blyk dat kasberging nie 'n groot verbetering vir hierdie navraag bied vir die uitvoertye (volgens normaliseringswaardes 1.000 sonder indekse, en 1.000 met indekse) of die SVE-tye (volgens normaliseringswaardes 0.935 sonder indekse, en 0.975 met indekse) nie. Die enigste aanduiding dat die indeks 'n verbetering lewer, is in die aantal logiese lees-aksies. Dit blyk dus dat die indeks dalk in hierdie geval onnodig is.

Tabel 6.37 – Die resultaat vir Navraag 5 ten opsigte van uitvoertyd, aantal rekords en "STATISTICS IO"

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	6	6	5	6
Normalisering	1.000	1.000	0.833	1.000
SVE-tyd (sekondes)	1.425	1.332	1.476	1.389
Normalisering	1.000	0.935	1.036	0.975
Aantal rekords	69980	69980	69980	69980
Logiese lees-aksies	1360	1360	1012	1012
Fisiese lees-aksies	0	0	4	0
Vooruitlees-aksies	688	0	1011	0
Aantal deursoeke	2	2	2	2

6.3.5.1.3 Die resultaat van die "DBCC SHOW_STATISTICS"-bevel (Navraag 5)

Die bevel toon die digthede, kardinaliteite en sleutellengtes vir die indekse wat geskep is. Tabel 6.38 vertoon die resultaat van die bevel.

Tabel 6.38 – Die digtheid en kardinaliteit vir die indeks(e) in Navraag 5

Kolom	Digtheid	Kardinaliteit	Gemiddelde Sleutellengte
van	0.000048	20971	7.125
netwerkid	0.000014	69979	7.936
netwerkid, voorletters	0.000014	69980	9.724

Die kardinaliteit van die *van*-kolom is 20971. Die kardinaliteit van die *netwerkid*-kolom en *<netwerkid, voorletters>*-kolomme is onderskeidelik 69979 en 69980. Die kardinaliteit van die *netwerkid*-kolom toon aan dat daar twee rekords met dieselfde netwerkid bestaan (69979 teen 69980). Die sleutellengtes word volgens die metodes in die vorige navrae uitgewerk.

6.3.5.1.4 Die uitvoerplan van SQL Server (Navraag 5)

Die uitvoerplan se waardes vir die navraag sonder indekse word in tabel 6.39 vertoon.

Die *p2*-tabel word eerste deursoek, gevolg deur die *p1*-tabeldeursoek. Die resultaat van die *p1*-tabeldeursoek word gebruik om 'n "hash"-tabel te bou ("Hash Match"). Elke ry van die verkrygde data in die *p2*-tabel word gebruik om 'n ooreenstemmende ry in die "hash"-tabel te kry ("Right semi-join"). Die "Hash match / Right Semi Join"-fase is die duurste fase, met 64%. Die twee tabeldeursoekfases het elkeen 18% van die totale koste in beslag geneem.

Tabel 6.39 – Die uitvoerplan se waardes vir Navraag 5 vir SQL Server (Geen indekse)

	Fases van uitvoerplan			
Waardes ondersoek	**p2 Tabeldeursoek**	**p1 Tabeldeursoek**	**"Hash Match / Right Semi Join"**	**Seleksie**
Fisies	Tabeldeursoek	Tabeldeursoek	"Hash Match"	Seleksie
Logies	Tabeldeursoek	Tabeldeursoek	"Right Semi Join"	Seleksie
Verwagte rye	69,980	69,980	69,980	69,980
Rygrootte	110	85	79	N.v.t.
Lees/skryf-aksie-koste	0.540	0.540	0	N.v.t.
SVE-koste	0.0770	0.0770	2.21	N.v.t.
Uitvoerings	1	1	1	N.v.t.
Koste van fase	0.617598 (18%)	0.617598 (18%)	2.212930 (64%)	0.006998 (0%)
Subboomkoste	0.617	0.617	3.45	3.46

Tabel 6.40 vertoon die uitvoerplan se waardes nadat die indeks bygevoeg is.

Tabel 6.40 – Die uitvoerplan se waardes vir Navraag 5 vir SQL Server (Met indeks(e))

	Fases van uitvoerplan			
Waardes ondersoek	**p2 Indeksdeursoek**	**p1 Gebondelde Indeksdeursoek**	**"Hash Match / Right Semi Join"**	**Seleksie**
Fisies	Indeksdeursoek	Gebondelde Indeksdeursoek	"Hash Match"	Seleksie
Logies	Indeksdeursoek	Gebondelde Indeksdeursoek	"Right Semi Join"	Seleksie
Verwagte rye	69,980	69,980	69,980	69,980
Rygrootte	78	112	79	N.v.t.
Lees/skryf-aksie-koste	0.262	0.558	0	N.v.t.
SVE-koste	0.0770	0.0770	2.21	N.v.t.
Uitvoerings	1	1	1	N.v.t.
Koste van fase	0.339079 (11%)	0.635376 (20%)	2.212930 (69%)	0.006998 (0%)
Subboomkoste	0.339	0.635	3.19	3.19

Die resultate van die indeksdeursoek (met ander woorde die indekse <*netwerkid, voorletters*>) van tabel *p2* word in 'n "hash"-tabel geplaas. Die "hash"-tabel word met die resultaat van die gebondelde indeksdeursoek (die gebondelde indeks op die *van*-kolom) van tabel *p1* verbind deur 'n "right semi join"-proses (sien tabel 3.4 in paragraaf 3.2.8.2), en die resultaat word vertoon.

Die indeks se uitvoerplan toon 'n verbetering in die koste van ongeveer 8% ((3.19 – 3.46) / 3.46 * (-100)), wat 'n aanduiding is dat die indeks dalk nie 'n goeie keuse is nie.

6.3.5.2 MySQL-interpretasie (Navraag 5)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 2) gegee is.

6.3.5.2.1 Moontlike indekse wat toegepas kan word

As indekse reeds op sommige van die kolomme (*netwerkid* of *voorletters*) bestaan, moet die navraag se prioriteit bepaal word ten opsigte van ander navrae wat op die twee kolomme toegespits is. As die navraag voldoende hoë prioriteit besit, kan 'n *<netwerkid>*-indeks, 'n *<voorletters>*-indeks, 'n *<netwerkid, voorletters>*-indeks of *<voorletters, netwerkid>*-indeks geskep word. As die navraag 'n lae prioriteit besit, word geen indeks toegepas nie. Aangesien die databasis staties is, word al vier indekse geskep (name van *persoon1* tot *persoon4*) en ondersoek. Nie een van hierdie indekse word deur MySQL vir die seleksienavraag buite die subnavraag gebruik nie. Tabel 6.41 toon die resultate.

Tabel 6.41 – Moontlike indekse vir Navraag 5 (MySQL)

Indeks(e)	Uitvoertyd (sekondes)	Aantal verwagte rye (subnavraag)
<netwerkid> (*persoon1*)	17 (13)	1 (1)
<netwerkid, voorletters> (*persoon3*)	12 (13)	1 (1)
<voorletters, netwerkid> (*persoon4*)	13 (12)	1 (1)
<voorletters> (*persoon2*)	333 (257)	21 (14)
Geen indekse (slegs tabeldeursoeke)	> 900	69980 (69717)

MySQL kies vir MyISAM enige van die twee oordekkingsindekse *persoon3* en *persoon4* (afhangende van watter indeks eerste geskep is) bo die enkelkolomindekse. InnoDB beskou die indekse *persoon1*, *persoon3* en *persoon4* op dieselfde vlak, en kies die indeks wat eerste geskep word. Die indeks *persoon3* word vir beide MyISAM en InnoDB gekies.

6.3.5.2.2 Die resultaat van die navraag ten opsigte van uitvoertyd en die aantal rekords

Tabel 6.42 toon die resultate vir Navraag 5.

Kasberging lewer net vir die InnoDB-stoorenjin met indekse 'n verbetering (van 0.014 na 0.011). Die indeks lewer 'n groot verbetering ten opsigte van die uitvoertyd.

Tabel 6.42 – Die resultaat vir Navraag 5 ten opsigte van uitvoertyd en die aantal rekords

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	> 900 (> 900)	> 900 (> 900)	12 (13)	12 (10)
Normaliseer	1.000 (1.000)	1.000 (1.000)	0.013 (0.014)	0.013 (0.011)
Aantal rekords	69980	69980	69980	69980

6.3.5.2.3 Die "EXPLAIN"-bevel se resultate met en sonder indekse

Die "EXPLAIN"-bevel word eerste vir 'n navraag sonder indekse ondersoek. Tabel 6.43 toon die resultaat van die bevel. Die resultaat dui aan dat 'n "DEPENDENT SUBQUERY"-seleksie-tipe (sien paragraaf 3.3.6.1 se tabel 3.5) gebruik is vir die subnavraag, wat aandui dat die navraag met die *p2*-tabel afhanklik is van die navraag met die *p1*-tabel, wat die primêre navraag is.

Tabel 6.43 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 5 (Geen indekse).

id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	PRIMARY	p1	ALL	NULL	NULL	NULL	NULL	69980 (69717)	Using where
2	DEPENDENT SUBQUERY	p2	ALL	NULL	NULL	NULL	NULL	69980 (69717)	Using where

Tabel 6.44 toon die "EXPLAIN"-bevel se resultaat nadat die indekse bygevoeg is.

Tabel 6.44 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 5 (Met indeks(e)).

id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	PRIMARY	p1	ALL	NULL	NULL	NULL	NULL	69980 (70628)	Using where
2	DEPENDENT SUBQUERY	p2	ref	persoon3	persoon3	34	Internet.p1.netwerkid, func (Internet_innodb.p1.netwerkid, func)	1 (1)	Using where Using index

Die *persoon3*-indeks is gebruik en die sleutellengte is 34. Die waarde vir die tweede ry van die resultaat se *verwysing*-kolom toon aan dat die *netwerkid*-kolom in die verwysing van die indeks gebruik word. Die "func"-waarde toon aan dat 'n funksie gebruik word om die *netwerkid*-waardes saam te groepeer en beskikbaar te stel vir die "IN"-gedeelte.

Die indeks het die aantal verwagte rye verminder van 69980 (69717 vir InnoDB) na 1 (1 vir InnoDB), wat aandui dat die indeks 'n verbetering is.

6.3.6 Navraag 6

Die navraag toon die gebruik van die "GROUP BY"-argument aan. In 'n logiese verband word die navraag gebruik om vir elke *userid*-kolom die totale hoeveelheid internetgebruikgrepe te vertoon en die totale grepe dan van groot na klein te sorteer.

```
SELECT userid, SUM(bytes) bytes, van, voorletters
FROM internet i JOIN persoon p ON i.userid = p.netwerkid
GROUP BY userid, van ,voorletters
ORDER BY SUM(bytes) DESC;
```

6.3.6.1 SQL Server-interpretasie (Navraag 6)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 1) gegee is.

6.3.6.1.1 Moontlike indekse wat toegepas kan word

Die ghoeroe stel voor dat 'n ontbondelde indeks op die *internet*-tabel se *userid*- en *bytes*-kolomme geskep word. Die ghoeroe voorspel 'n 18%-verbetering in die werkverrigting. Die indekse wat ondersoek word, word in tabel 6.45 getoon. Al die uitvoertye en SVE-tye in tabel 6.45 is sonder kasberging vertoon.

As daar nie groot hoeveelhede wysigings op die *internet* en *persoon*-tabelle plaasvind nie, kan die ontbondelde indekse *<userid, bytes>* en *<netwerkid, van, voorletters>* toegepas word. As baie wysigings op die tabelle toegepas word, is die ontbondelde indeks *<userid, bytes>* die beste keuse. Logies gesproke behoort daar nie baie wysigings op die *netwerkid-*, *van-*, *voorletters-* en *userid*-kolomme plaas te vind nie.

Die ontbondelde indekse *<userid, bytes>* en *<netwerkid, van, voorletters>* word toegepas.

Tabel 6.45 – Moontlike indekse vir Navraag 6 (SQL Server)

Indeks(e)	Koste	Uitvoertyd (sekondes)	SVE-tyd (sekondes)
Ontbondelde indeks *<userid, bytes>* op die *internet* -tabel, ontbondelde indeks *<netwerkid, van, voorletters>* op die *persoon*-tabel	31.4	1.879	1.082
Ontbondelde indeks *<userid, bytes>* op die *internet* -tabel, gebondelde indeks *<netwerkid, van, voorletters>* op die *persoon*-tabel	31.7	1.957	1.085
Ontbondelde indeks *<userid, bytes>* op die *internet* -tabel, gebondelde indeks *<netwerkid>* op die *persoon*-tabel	31.7	2.206	1.132
Ontbondelde indeks *<userid, bytes>*	32.0	2.077	1.269
Geen indekse (slegs tabeldeursoeke)	33.3	4	2.834
Gebondelde indeks *<userid, bytes>* op die *internet* -tabel, ontbondelde indeks *<netwerkid, van, voorletters>* op die *persoon*-tabel	35.3	5	1.302
Gebondelde indeks *<userid, bytes>* op die *internet* -tabel, gebondelde indeks *<netwerkid>* op die *persoon*-tabel	35.7	5	1.315
Gebondelde indeks *<userid, bytes>* op die *internet* -tabel, gebondelde indeks *<netwerkid, van, voorletters>* op die *persoon*-tabel	35.7	5	1.339
Gebondelde indeks *<userid, bytes>* op die *internet* -tabel	35.9	5	1.485
Ontbondelde indeks *<userid, bytes>* op die *internet* -tabel, ontbondelde indeks *<netwerkid >* op die *persoon*-tabel	36.6	2.597	1.455
Gebondelde indeks *<userid, bytes>* op die *internet* -tabel, ontbondelde indeks *<netwerkid>* op die *persoon*-tabel	40.6	5	1.252

6.3.5.1.2 Die resultaat van die navraag ten opsigte van uitvoertyd, aantal rekords en die "STATISTICS IO"-opsie (Navraag 6)

Die waardes vir die gemiddelde uitvoertyd, die aantal rekords en die inligting verskaf deur die "STATISTICS IO"-opsie word in tabel 6.46 getoon.

Kasberging lewer 'n beter uitvoertyd (volgens normaliseringswaardes 0.669 sonder indekse, en 0.261 met indekse) maar die SVE-tyd wissel nie aansienlik nie (vanaf 1.000 na 0.940 sonder indekse, en vanaf 0.382 na 0.368 met indekse). Die indekse lewer 'n verbetering ten opsigte van uitvoertyd, SVE-tyd en logiese lees-aksies. Dit blyk dat die indekse goeie keuses is.

6.3.5.1.3 Die resultaat van die "DBCC SHOW_STATISTICS"-bevel (Navraag 6)

Die bevel toon die digthede, kardinaliteite en sleutellengtes vir die indekse wat geskep is. Tabel 6.47 vertoon die resultaat van die bevel.

Tabel 6.46 – Die resultaat vir Navraag 6 ten opsigte van uitvoertyd, aantal rekords en "STATISTICS IO"

	Tabel	Sonder kasgeheue Sonder indeks (sekondes)	Met kasgeheue Sonder indeks (sekondes)	Sonder kasgeheue Met indeks(e) (sekondes)	Met kasgeheue Met indeks(e) (sekondes)
Uitvoertyd (sekondes)	Beide	4	2.675	1.879	1.045
Normalisering	Beide	1.000	0.669	0.470	0.261
SVE-tyd (sekondes)	Beide	2.834	2.665	1.082	1.042
Normalisering	Beide	1.000	0.940	0.382	0.368
Aantal rekords	Beide	543	543	543	543
Logiese lees-aksies	Persoon	704	704	339	339
Fisiese lees-aksies	Persoon	0	0	2	0
Vooruitlees-aksies	Persoon	712	0	338	0
Aantal deursoeke	Persoon	1	1	1	1
Logiese lees-aksies	Internet	6798	6798	1339	1339
Fisiese lees-aksies	Internet	0	0	2	0
Vooruitlees-aksies	Internet	6823	0	1343	0
Aantal deursoeke	Internet	1	1	1	1

Die kardinaliteit stem met vorige navrae se waardes ooreen. Die *bytes*-kolom van tabel *internet* is van tipe "big integer" en beslaan 8 grepe. Dit verklaar die verskil tussen die gemiddelde sleutellengtes van 5.905 en 13.905 in tabel 6.47.

Tabel 6.47 – Die digtheid en kardinaliteit vir die indeks(e) in Navraag 6

Kolom	Digtheid	Kardinaliteit	Gemiddelde Sleutellengte
userid	0.001105	905	5.905
userid, bytes	0.000004	244389	13.905
netwerkid	0.000014	69979	7.936
netwerkid, van	0.000014	69980	15.061
netwerkid, van, voorletters	0.000014	69980	16.849

6.3.5.1.4 Die uitvoerplan van SQL Server (Navraag 6)

Die uitvoerplan se waardes vir die navraag sonder indekse word in tabel 6.48 vertoon. Die seleksiefase se koste is weglaatbaar klein en is saam met die selekteerfase geplaas.

'n Tabeldeursoek word op die *internet*-tabel toegepas. Die resultaat van hierdie deursoek word in 'n "hash"-tabel geplaas en gegroepeer op die "GROUP BY"-gedeelte se kolomme ("Hash match / Aggregate" fase). Die groepering bereken die saamgestelde funksie ("aggregate function") "SUM" se waarde en die waarde word in die "hash"-tabel gestoor saam met die kolomme wat gegroepeer is (*userid, van* en *voorletters*).

Tabel 6.48 – Die uitvoerplan se waardes vir Navraag 6 vir SQL Server (Geen indekse)

	Fases van uitvoerplan						
Waardes ondersoek	**Internet Tabeldeursoek**	**"Hash Match / Aggregate"**	**persoon Tabeldeursoek**	**"Hash Match Team" / binneverbinding**	**"Hash Match Root / Aggregate"**	**Bereken Skalaar**	**Sorteer en seleksie**
Fisies	Tabeldeursoek	"Hash Match"	Tabeldeursoek	"Hash Match Team"	"Hash Match Root"	Bereken Skalaar	Sorteer en seleksie
Logies	Tabeldeursoek	"Aggregate"	Tabeldeursoek	Binneverbinding	"Aggregate"	Bereken Skalaar	Sorteer en seleksie
Verwagte rye	338,859	300	69,980	300	292,423	292,423	292,423
Rygrootte	237	32	85	53	44	36	36
Lees/skryf-aksie-koste	5.07	0	0.558	0	0	0	0.0112
SVE-koste	0.372	2.24	0.0770	0.484	0.0329	0.0292	24.4
Uitvoerings	1	1	1	1	1	1	1
Koste van fase	5.445217 (16%)	2.240857(7%)	0.635376 (2%)	0.484465 (1%)	0.032923 (0%)	0.029241 (0%)	24.383032 (73%)
Subboomkoste	5.45	7.69	0.635	8.81	8.84	8.87	33.3

'n Tabeldeursoek word op die *persoon*-tabel toegepas en die resultaat word met die "hash"-tabel verbind ("Hash Match Team / binneverbindingfase). 'n Fase word bereik waar al die vorige "hash"-funksies gekoördineer word en verder opsommende prosesse plaasvind ("Hash Match Root / Aggregate"). Die fase bring die resultate van die twee tabelle se onderskeie fases saam. In die volgende fase "Bereken Skalaar" ("Compute Scalar") word nuwe waardes bereken van die bestaande waardes in 'n ry. Laastens word die data gesorteer en die seleksie op die data uitgevoer. Die totale koste na seleksie is 33.3.

Tabel 6.49 vertoon die uitvoerplan se waardes nadat die indeks bygevoeg is.

Die enigste verskil tussen die uitvoerplan met die indeks en die uitvoerplan sonder enige indekse is die eerste fases van elke tabel. Die eerste fase op die *internet*-tabel is 'n ontbondelde indeksdeursoek, gevolg deur die tweede fase – 'n "Stream Aggregate / Aggregate"-fase, wat opsommende waardes skep wat soortgelyk is aan die proses wat sommige subnavrae met die "IN"-argument gebruik. Die eerste fase op die *persoon*-tabel is ook 'n ontbondelde indeksdeursoek.

Tabel 6.49 – Die uitvoerplan se waardes vir Navraag 6 vir SQL Server (Met indeks(e))

	Fases van uitvoerplan						
Waardes ondersoek	**internet Indeks-deursoek**	**"Stream Aggregate / Aggregate"**	**persoon Indeks-Deursoek**	**"Hash Match Team" / binneverbinding**	**"Hash Match Root / Aggregate"**	**Bereken Skalaar**	**Sorteer en seleksie**
Fisies	Indeks-deursoek	"Hash Match"	Indeks-deursoek	"Hash Match Team"	"Hash Match Root"	Bereken Skalaar	Sorteer en seleksie
Logies	Indeks-deursoek	"Aggregate"	Indeks-deursoek	binneverbinding	"Aggregate"	Bereken Skalaar	Sorteer en seleksie
Verwagte rye	338,859	906	69,980	891	337,016	337,016	337,016
Rygrootte	30	32	36	53	44	36	36
Lees/skryf-aksie-koste	0.513	0	0.286	0	0	0	0.0112
SVE-koste	0.186	0.953	0.0770	0.154	0.0674	0.0337	28.4
Uitvoerings	1	1	1	1	1	1	1
Koste van fase	1.400032 (4%)	0.953018 (3%)	0.363524 (1%)	0.154716 (0%)	0.067421 (0%)	0.033702 (0%)	28.416384 (91%)
Subboomkoste	1.40	2.35	0.363	2.87	2.94	2.97	31.4

Die indeks se uitvoerplan toon 'n verbetering in die koste van ongeveer 6% ((31.4 – 33.3) / 33.3 * (-100)), wat 'n aanduiding is dat die indeks nie 'n groot verbetering ten opsigte van die koste teweeg bring nie.

6.3.6.2 MySQL-interpretasie (Navraag 6)

MySQL se groeperingsfunksie "GROUP BY" sorteer alreeds elke kolom in die groepering (MySQL AB, 2005:736), en die "ORDER BY"-argument bied nie funksionaliteit om volgens die groeperingsfunksie "SUM" te sorteer nie, maar kan wel volgens 'n alias vir die groeperingsfunksie sorteer (MySQL AB, 2005:693). Om dieselfde resultate te verkry, moet die groeperingsfunksie 'n alias kry en sortering volgens die alias geskied:

```
SELECT userid, SUM(bytes) tbytes, van, voorletters
FROM internet i JOIN persoon p ON i.userid = p.netwerkid
GROUP BY userid, van, voorletters
ORDER BY tbytes DESC;
```

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 2) gegee is.

6.3.6.2.1 Moontlike indekse wat toegepas kan word

Drie indekse word vir die navraag in MySQL ondersoek. Die indeks op die *internet*-tabel is op die *<userid, bytes>*-kolomme (*internet1*), en die indekse op die *persoon*-tabel is *<netwerkid>* (*persoon1*) en *<netwerkid, van, voorletters>* (*persoon2*). Tabel 6.50 toon die moontlike indekse en verskeie kombinasies, die uitvoertye en die aantal verwagte rye.

Tabel 6.50 – Moontlike indekse vir Navraag 6 (MySQL)

Indeks(e)	Uitvoertyd (sekondes)	Aantal verwagte rye (*internet*)	Aantal verwagte rye (*persoon*)
<userid, bytes> op die *internet* -tabel	4 (11)	374 (173)	69980 (69706)
<userid, bytes> op die *internet* -tabel, *<netwerkid >* op die *persoon* -tabel	28 (22)	338859 (344842)	1 (1)
<userid, bytes> op die *internet* -tabel, *<netwerkid, van, voorletters>* op die *persoon* -tabel	20 (20)	338859 (344842)	1 (1)
Geen indekse (slegs tabeldeursoeke)	> 900	338859 (338099)	69980 (70116)

MySQL beskou die indekse *persoon1* en *persoon2* op dieselfde vlak, en kies die indeks wat eerste geskep is. Dit blyk dat die indeks *<userid, bytes>* alleen beter deur MySQL gebruik word as die ander twee kombinasies, en die indeks word gekies.

6.3.6.2.2 Die resultaat van die navraag ten opsigte van uitvoertyd en die aantal rekords

Tabel 6.51 toon die resultate vir Navraag 6.

Tabel 6.51 – Die resultaat vir Navraag 6 ten opsigte van uitvoertyd en die aantal rekords

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	> 900 (> 900)	> 900 (> 900)	4 (11)	3 (11)
Normaliseer	1.000 (1.000)	1.000 (1.000)	0.004 (0.012)	0.003 (0.012)
Aantal rekords	543	543	543	543

Kasberging het nie 'n groot uitwerking op die uitvoertye gelewer nie. Die indeks lewer wel 'n verbetering (volgens normaliseringswaardes 0.004 en 0.003) vir die uitvoertye, en is 'n goeie keuse.

6.3.5.2.3 Die "EXPLAIN"-bevel se resultate met en sonder indekse

Die "EXPLAIN"-bevel word eerste vir 'n navraag sonder indekse ondersoek. Tabel 6.52 toon die resultaat van die bevel.

Tabel 6.52 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 6 (Geen indekse)

id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	SIMPLE	i	ALL	NULL	NULL	NULL	NULL	338859 (338099)	Using temporary, Using filesort
2	SIMPLE	p	ALL	NULL	NULL	NULL	NULL	69980 (70116)	Using where

Die resultaat dui aan dat beide tabeldeursoeke eenvoudige seleksiebewerkings is (geen "UNIONS" of subnavrae nie). Die aantal verwagte rye stem vir MyISAM ooreen met die totale aantal rye in die onderskeie tabelle (InnoDB verkry 338099 vir die *internet*-tabel en 70116 vir die *persoon*-tabel). Die "Using temporary"-gedeelte van die *ekstra*-kolom dui daarop dat "GROUP BY"- en "ORDER BY"-gedeeltes bestaan wat nie dieselfde kolomme en volgordes lys nie. Die resultaat van die navraag word in 'n tydelike tabel gestoor. Die "Using filesort" dui daarop dat MySQL deur die tydelike tabel gaan en die rekords sorteer volgens die "ORDER BY"-gedeelte.

Tabel 6.53 toon die "EXPLAIN"-bevel se resultaat nadat die indeks bygevoeg is.

Tabel 6.53 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 6 (Met indeks(e))

id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	SIMPLE	p	ALL	NULL	NULL	NULL	NULL	69980 (69552)	Using temporary, Using filesort
2	SIMPLE	i	ref	internet1	internet1	21	internet.p.netwerkid (internet_innodb.p.netwerkid)	374 (80)	Using where, Using index

Die *persoon*-tabel word eerste deur MySQL verwerk en daarna die *internet*-tabel. Die *internet*-tabel gebruik die indeks om die kwalifiserende rye te vind. Die resultaat van die twee tabelle word in 'n tydelike tabel gestoor en gesorteer.

Die indeks het die aantal verwagte rye verminder van 338859 (338099 vir InnoDB) na 374 (80 vir InnoDB) vir die *internet*-tabel, wat aandui dat die indeks 'n verbetering is.

6.3.7 Navraag 7

Die navraag toon die uitwerking aan wat 'n verbinding van vyf tabelle het. Die "DISTINCT"-argument word ook hier gebruik.

```
SELECT DISTINCT Datetime, bytes, userid, van, voorname, taal, t1.kolom1, t2.kolom2,
t3.kolom1
FROM internet i
JOIN persoon p ON i.userid = p.netwerkid
JOIN tabel1 t1 ON p.netwerkid = t1.verbindkolom
JOIN tabel3 t3 ON t1.kolom1 = t3.verbindkolom
JOIN tabel2 t2 ON t1.kolom1 = t2.verbindkolom
```

6.3.7.1 SQL Server-interpretasie (Navraag 7)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 1) gegee is.

6.3.7.1.1 Moontlike indekse wat toegepas kan word

As gevolg van tabelle *tabel1, tabel2* en *tabel3* se klein aantal kolomme en rekords sal indekse hierop nie 'n groot verskil in die koste, uitvoertyd en SVE-tyd lewer nie. Ontbondelde oordekkingsindekse (gebondelde indekse is ook 'n moontlikheid) word egter wel vir die drie tabelle ondersoek om die effek op die koste, uitvoertyd en SVE-tyd te toon.

Die ghoeroe stel geen indekse voor nie. 'n Moontlike keuse vir hierdie navraag is oordekkingsindekse (gebondel of ontbondel) op al die betrokke kolomme van elke tabel. Dit is egter normaalweg nie prakties moontlik nie as gevolg van ander navrae wat tot die tabelle gerig word en die keuses van indeks lei. Tabel 6.54 vertoon die moontlike indeks-kombinasies wat ondersoek is.

Dit blyk dat enige ander indeks as 'n oordekkingsindeks nie 'n groot invloed op die navraag uitoefen nie. Die beste indeks in hierdie geval is die twee oordekkingsindekse *<userid, bytes, datetime>* op die *internet*-tabel en *<netwerkid, van, voorname, taal>* op die *persoon*-tabel. Aangesien dit blyk dat die oordekkingsindekse op die *tabel1-, tabel2-* en *tabel3*-tabelle nie 'n verbetering lewer nie, word hulle nie verder gebruik nie. Die twee oordekkingsindekse op die *internet-* en *persoon*-tabelle word gebruik.

Tabel 6.54 – Moontlike indekse vir Navraag 7 (SQL Server)

Indeks(e)	Koste	Uitvoertyd (sekondes)	SVE-tyd (sekondes)
Ontbondelde indeks *<userid, bytes, datetime>* op *internet* -tabel, ontbondelde indeks *<netwerkid, van, voorname, taal>* op *persoon*-tabel	216	30	2.330
Gebondelde indeks *<userid, bytes, datetime>* op *internet* -tabel, gebondelde indeks *<netwerkid, van, voorname, taal>* op *persoon*-tabel	216	30	2.389
Ontbondelde indeks *<userid, bytes, datetime>* op *internet* -tabel, Ontbondelde indeks *<netwerkid, van, voorname, taal>* op *persoon*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel1*-tabel, ontbondelde indeks *<verbindkolom, kolom2>* op *tabel2*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel3*-tabel	216	32	2.464
Gebondelde indeks *<userid, bytes, datetime>* op *internet* -tabel, gebondelde indeks *<netwerkid, van, voorname, taal>* op *persoon*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel1*-tabel, ontbondelde indeks *<verbindkolom, kolom2>* op *tabel2*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel3*-tabel	216	49	2.389
Gebondelde indeks *<userid >* op *internet* -tabel, gebondelde indeks *<netwerkid >* op *persoon*-tabel	220	35	2.587
Gebondelde indeks *<userid>* op *internet* -tabel, gebondelde indeks *<netwerkid>* op *persoon*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel1*-tabel, ontbondelde indeks *<verbindkolom, kolom2>* op *tabel2*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel3*-tabel	220	38	2.579
Geen indekse (slegs tabeldeursoeke)	290	35	2.521
Ontbondelde indeks *<userid>* op *internet* -tabel, Ontbondelde indeks *<netwerkid>* op *persoon*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel1*-tabel, ontbondelde indeks *<verbindkolom, kolom2>* op *tabel2*-tabel, ontbondelde indeks *<verbindkolom, kolom1>* op *tabel3*-tabel	2024	34	5
Ontbondelde indeks *<userid >* op *internet* -tabel, Ontbondelde indeks *<netwerkid >* op *persoon*-tabel	2024	38	5

6.3.5.1.2 Die resultaat van die navraag ten opsigte van uitvoertyd, aantal rekords en die "STATISTICS IO"-opsie (Navraag 7)

Die waardes vir die gemiddelde uitvoertyd, die aantal rekords en die inligting verskaf deur die "STATISTICS IO"-opsie word in tabel 6.55 getoon.

'n Werkstabel word deur SQL Server vir die navraag met indekse gebruik om die resultate te verwerk. Kasberging lewer 'n merkbare verskil ten opsigte van die SVE-tyd (volgens die normaliseringswaardes 0.810 sonder indekse, en 0.751 met indekse) maar die uitvoertyd wissel nie aansienlik nie (vanaf 1.000 na 0.971 sonder indekse, en vanaf 0.857 na 0.886 met indekse). Die tabel *tabel3* se logiese lees-aksies verskil aansienlik wanneer indekse bygevoeg word. Te oordeel na die SVE-tyd was die twee indekse goeie keuses.

Tabel 6.55 – Die resultaat vir Navraag 7 ten opsigte van uitvoertyd, aantal rekords en "STATISTICS IO"

	Tabel	Sonder kasgeheue Sonder indeks (sekondes)	Met kasgeheue Sonder indeks (sekondes)	Sonder kasgeheue Met indeks(e) (sekondes)	Met kasgeheue Met indeks(e) (sekondes)
Uitvoertyd (sekondes)	Almal	35	34	30	31
Normalisering	Almal	1.000	0.971	0.857	0.886
SVE-tyd (sekondes)	Almal	2.521	2.043	2.330	1.893
Normalisering	Almal	1.000	0.810	0.924	0.751
Aantal rekords	Almal	29599	29599	29599	29599
Logiese lees-aksies	Internet	6675	6675	1677	1677
Fisiese lees-aksies	*Internet*	0	0	2	0
Vooruitlees-aksies	Internet	6698	0	1684	0
Aantal deursoeke	Internet	1	1	1	1
Logiese lees-aksies	Tabel3	95	95	8260	6336
Fisiese lees-aksies	Tabel3	0	0	5	0
Vooruitlees-aksies	Tabel3	95	0	68	0
Aantal deursoeke	Tabel3	1	1	1984	1984
Logiese lees-aksies	Persoon	680	680	519	519
Fisiese lees-aksies	Persoon	0	0	2	0
Vooruitlees-aksies	Persoon	685	0	519	0
Aantal deursoeke	Persoon	1	1	1	1
Logiese lees-aksies	Tabel1	22	22	19	19
Fisiese lees-aksies	Tabel1	0	0	1	0
Vooruitlees-aksies	Tabel1	23	0	18	0
Aantal deursoeke	Tabel1	1	1	1	1
Logiese lees-aksies	Tabel2	8	8	8	8
Fisiese lees-aksies	Tabel2	0	0	1	0
Vooruitlees-aksies	*Tabel2*	*9*	*0*	*7*	*0*
Aantal deursoeke	Tabel2	1	1	1	1
Logiese lees-aksies	Werkstabel	Nie van toepassing	Nie van toepassing	14300	14300
Fisiese lees-aksies	Werkstabel	Nie van toepassing	Nie van toepassing	0	0
Vooruitlees-aksies	Werkstabel	Nie van toepassing	Nie van toepassing	0	0
Aantal deursoeke	Werkstabel	Nie van toepassing	Nie van toepassing	14078	14078

6.3.5.1.3 Die resultaat van die "DBCC SHOW_STATISTICS"-bevel (Navraag 7)

Die bevel toon die digthede, kardinaliteite en sleutellengtes vir die indekse wat geskep is. Tabel 6.56 vertoon die resultaat van die bevel.

Die kardinaliteit stem met vorige navrae se waardes ooreen. Die *datetime*-kolom is van tipe "datetime" en beslaan 8 grepe. Die waardes stem ooreen met die navrae (gewysig met die betrokke kolomme) wat in paragraaf 6.3.1.1.3 vir die eerste keer gebruik is.

Tabel 6.56 – Die digtheid en kardinaliteit vir die indeks(e) in Navraag 7

Kolom	Digtheid	Kardinaliteit	Gemiddelde Sleutellengte
Userid	0.001105	905	5.905
userid, bytes	0.000004	244389	13.905
userid, bytes, datetime	0.000003	325651	21.905
Netwerkid	0.000014	69979	7.936
netwerkid, van	0.000014	69980	15.061
netwerkid, van, voorname	0.000014	69980	27.923
netwerkid, van, voorname, taal	0.000014	69980	35.588

6.3.5.1.4 Die uitvoerplan van SQL Server (Navraag 7)

Die uitvoerplan se waardes vir die navraag sonder indekse word in tabel 6.57 vertoon.

Tabeldeursoeke word eerstens op tabel *tabel2* en daarna op tabel *tabel1* toegepas. Die resultaat van die deursoek op *tabel2* word in 'n "hash"-tabel geplaas en verbind met die resultaat van die *tabel1*-tabel ("Hash Match" / binneverbinding) om weer 'n "hash"-tabel (H1) te vorm. 'n Tabeldeursoek word op die *persoon*-tabel toegepas. Die resultaat van hierdie deursoek word verbind met H1 om weer 'n "hash"-tabel (H2) te vorm. 'n Tabeldeursoek word toegepas op die *tabel3*-tabel en die resultaat word verbind met H2 om "hash"-tabel, H3, te vorm.

Tabel 6.57 – Die uitvoerplan se waardes vir Navraag 7 vir SQL Server (Geen indekse)

	Fases van uitvoerplan						
Waardes ondersoek	**tabel2 Tabeldeur-soek**	**tabel1 Tabeldeur-soek**	**"Hash Match" / binne-verbinding H1**	**persoon Tabeldeur-soek**	**"Hash Match" / binne-verbinding H2**	**tabel3 Tabeldeur-soek**	**"Hash Match" / binne-verbinding H3**
Fisies	Tabeldeursoek	Tabeldeursoek	"Hash Match"	Tabeldeursoek	"Hash Match"	Tabeldeursoek	"Hash Match"
Logies	Tabeldeursoek	Tabeldeursoek	Binne-verbinding	Tabeldeursoek	Binne-verbinding	Tabeldeursoek	Binne-verbinding
Verwagte rye	2,000	4,741	1,999	69,980	1,999	2,000	2,283
Rygrootte	25	38	32	85	73	274	324
Lees/skryf-aksie-koste	0.0427	0.0531	0	0.540	0	0.107	0
SVE-koste	0.00227	0.00529	0.116	0.0770	0.564	0.00227	0.134
Uitvoerings	1	1	1	1	1	1	1
Koste van fase	0.045042 (0%)	0.058428 (0%)	0.116384 (0%)	0.617598 (0%)	0.564223 (0%)	0.109487 (0%)	0.134658 (0%)
Subboomkoste	0.0450	0.0584	0.219	0.617	1.40	0.109	1.65

(vervolg)	Fases van uitvoerplan			
Waardes ondersoek	**internet Tabeldeursoek**	**"Hash Match" / binneverbinding H4**	**"Hash Match / Aggregate"**	**Seleksie**
Fisies	Tabeldeursoek	"Hash Match"	"Hash Match"	Seleksie
Logies	Tabeldeursoek	binneverbinding	"Aggregate"	Seleksie
Verwagte rye	338,859	520,173	520,173	520,173
Rygrootte	237	339	330	N.v.t.
Lees/skryf-aksie-koste	4.98	0	135	N.v.t.
SVE-koste	0.372	3.43	145	N.v.t.
Uitvoerings	1	1	1	N.v.t.
Koste van fase	5.354106 (2%)	3.428683 (1%)	279.889052 (96%)	N.v.t.
Subboomkoste	5.35	10.4	290	290

Die laaste tabeldeursoek word op die *internet*-tabel toegepas en die resultaat word verbind met die "hash"-tabel H3 om 'n volgende "hash"-tabel, H4, te vorm (die verbinding gebruik die *userid*-kolom). Die rye van H4 word gegroepeer (soos met 'n "GROUP BY"-argument) en nadat saamgestelde funksies op die groepering toegepas is, word die resultaat in 'n finale "hash"-tabel gestoor ("Hash Match / Aggregate"). Die resultaat word uit die laaste "hash"-tabel geselekteer.

Tabel 6.58 vertoon die uitvoerplan se waardes nadat die indeks bygevoeg is. Die eerste drie fases (tot by die vorming van H1) is presies dieselfde as die navraag sonder indekse se uitvoerplan. 'n Indeksdeursoek word op die *persoon*-tabel toegepas en die resultaat word verbind met die bestaande "hash"-tabel H1 om 'n tweede "hash"-tabel te vorm (H2).

Die volgende twee fases stem dan ooreen met die navraag sonder indekse se fases tot by die vorming van H3. 'n Werkstabel word geskep wat die gesorteerde rye van H3 bevat. 'n Indeksdeursoek word op die *internet*-tabel toegepas en die resultaat word met 'n saamvoegverbinding aan die werkstabel se data gekoppel (saamvoegverbinding/binneverbinding). Die volgende fase pas groepering en saamgestelde funksies op die resultaat van die verbinding toe. Seleksie is die laaste fase in die uitvoerplan. Die indeks se uitvoerplan toon 'n verbetering in die koste van ongeveer 26% ((216 – 290) / 290 * (-100)), wat 'n aanduiding is dat die indeks 'n groot verbetering ten opsigte van die koste aangebring het.

Tabel 6.58 – Die uitvoerplan se waardes vir Navraag 7 vir SQL Server (Met indeks(e))

	Fases van uitvoerplan						
Waardes ondersoek	**tabel2 Tabeldeur-soek**	**tabel1 Tabeldeur-soek**	**"Hash Match" / binne-verbindin g H1**	**persoon Indeks-deursoek**	**"Hash Match" / binne-verbindin g H2**	**tabel3 Tabeldeur-soek**	**"Hash Match" / binne-verbindin g H3**
Fisies	Tabeldeursoe k	Tabeldeursoe k	"Hash Match"	Indeks-deursoek	"Hash Match"	Tabeldeursoe k	"Hash Match"
Logies	Tabeldeursoe k	Tabeldeursoe k	Binne-verbinding	Indeks-deursoek	Binne-verbinding	Tabeldeursoe k	Binne-verbinding
Verwagte rye	2,000	4,741	1,999	69,980	1,999	2,000	2,217
Rygrootte	25	38	32	56	73	274	333
Lees/skryf-aksie-koste	0.0427	0.0531	0	0.209	0	0.107	0
SVE-koste	0.00227	0.00529	0.116	0.0385	0.564	0.00227	0.142
Uitvoerings	1	1	1	1	1	1	1
Koste van fase	0.045042 (0%)	0.058428 (0%)	0.116384 (0%)	0.496857(0%)	0.564223 (0%)	0.109487 (0%)	0.142102 (0%)
Subboomkost e	0.0450	0.0584	0.219	0.496	1.28	0.109	1.53

(vervolg)	**Fases van uitvoerplan**				
Waardes ondersoek	**Sortering (Werkstabel)**	**internet Indeksdeursoek**	**Saamvoeg-verbinding / binneverbinding**	**"Hash Match / Aggregate"**	**Seleksie**
Fisies	Sorteer	Indeksdeursoek	Saamvoegverbinding	"Hash Match"	Seleksie
Logies	Sorteer	Indeksdeursoek	Binneverbinding	"Aggregate"	Seleksie
Verwagte rye	2,217	338,859	412,089	412,089	412,089
Rygrootte	324	38	354	330	N.v.t.
Lees/skryf-aksie-koste	0.0112	1.28	0.175	97.0	N.v.t.
SVE-koste	0.0385	0.372	1.18	114	N.v.t.
Uitvoerings	1	1	1	1	N.v.t.
Koste van fase	0.049817 (0%)	1.650402 (1%)	1.355572 (1%)	211.385637 (98%)	N.v.t.
Subboomkoste	1.58	1.65	4.59	216	216

6.3.7.2 MySQL-interpretasie (Navraag 7)

Die navraag word beskryf volgens die model wat in paragraaf 6.2 (punt 2) gegee is.

6.3.7.2.1 Moontlike indekse wat toegepas kan word

Indekse wat soortgelyk is aan die indekse vir SQL Server in paragraaf 6.3.6.1.1 word hier ondersoek. Tabel 6.59 toon die moontlike indekse en verskeie kombinasies, die uitvoertye en

die aantal verwagte rye. Soos verwag, sal MySQL die oordekkingsindekse kies bo die indekse wat net een kolom bevat.

Tabel 6.59 – Moontlike indekse vir Navraag 7 (MySQL)

Indeks(e)	Uitvoertyd (sekondes)	Aantal verwagte rye (*internet*)	Aantal verwagte rye (*persoon*)	Aantal verwagte rye (*tabel1*)	Aantal verwagte rye (*tabel2*)	Aantal verwagte rye (*tabel3*)
Indeks *<userid, bytes, datetime>* op *internet*-tabel, Indeks *<netwerkid, van, voorname, taal>* op *persoon*-tabel, Indeks *<verbindkolom, kolom1>* op *tabel1*-tabel, Indeks *<verbindkolom, kolom2>* op *tabel2*-tabel, Indeks *<verbindkolom, kolom1>* op *tabel3*-tabel	21 (22)	374 (94)	1 (1)	4741 (4831)	1 (1)	1 (1)
Indeks *<userid >* op *internet*-tabel, Indeks *<netwerkid >* op *persoon*-tabel, Indeks *<verbindkolom, kolom1>* op *tabel1*-tabel, Indeks *<verbindkolom, kolom2>* op *tabel2*-tabel, Indeks *<verbindkolom, kolom1>* op *tabel3*-tabel	53 (37)	338859 (222)	1 (1)	1 (4831)	1 (1)	1 (1)
Indeks *<userid, bytes, datetime>* op *internet*-tabel, Indeks *<netwerkid, van, voorname, taal>* op *persoon*-tabel	436 (440)	374 (165)	1 (1)	4741 (4831)	2000 (1965)	2000 (2048)
Indeks *<userid >* op *internet*-tabel, Indeks *<netwerkid >* op *persoon*-tabel	431 (508)	374 (204)	1 (1)	4741 (4831)	2000 (1965)	2000 (2048)
Geen indekse (slegs tabeldeursoeke)	> 900	338859 (341605)	69980 (70372)	4741 (4653)	2000 (1899)	2000 (2045)

MySQL verkies die oordekkingsindekse bo die ander indekse. Dit blyk dat die oordekkingsindekse op al die betrokke kolomme van elke tabel die beste in resultaatterme van uitvoertyd lewer. Die produk van die aantal verwagte rye vir die eerste indeks-kombinasie in tabel 6.59 gee die waarde 1773134 (vir MyISAM) en 454114 (vir InnoDB), wat meer is as die tweede indekskombinasie se 338859 en 1072482. Die eerste indekskombinasie lewer 'n beter uitvoertyd, en word gekies.

6.3.7.2.2 Die resultaat van die navraag ten opsigte van uitvoertyd en die aantal rekords

Tabel 6.60 toon die resultate vir Navraag 7.

Tabel 6.60 – Die resultaat vir Navraag 7 ten opsigte van uitvoertyd en die aantal rekords

	Sonder kasgeheue Sonder indeks	Met kasgeheue Sonder indeks	Sonder kasgeheue Met indeks(e)	Met kasgeheue Met indeks(e)
Uitvoertyd (sekondes)	> 900 (> 900)	> 900 (> 900)	21 (22)	20 (20)
Normaliseer	1.000 (1.000)	1.000 (1.000)	0.023 (0.024)	0.022 (0.022)
Aantal rekords	29599	29599	29599	29599

Kasberging het nie 'n groot uitwerking op die uitvoertye gelewer nie. Die indeks lewer wel 'n groot verbetering (volgens normaliseringswaardes 0.023 en 0.022 vir MyISAM en 0.024 en 0.022 vir InnoDB) vir die uitvoertye, en is 'n goeie keuse.

6.3.7.2.3 Die "EXPLAIN"-bevel se resultate met en sonder indekse

Die "EXPLAIN"-bevel word eerste ondersoek vir 'n navraag sonder indekse. Tabel 6.61 toon die resultaat van die bevel.

Tabel 6.61 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 7 (Geen indekse).

id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	SIMPLE	i	ALL	NULL	NULL	NULL	NULL	338859 (341605)	Using temporary
1	SIMPLE	p	ALL	NULL	NULL	NULL	NULL	69980 (70372)	Using where
1	SIMPLE	t3	ALL	NULL	NULL	NULL	NULL	2000 (2045)	Geen waarde
1	SIMPLE	t1 (t2)	ALL	NULL	NULL	NULL	NULL	4741 (1899)	Using where (Geen waarde)
1	SIMPLE	t2 (t1)	ALL	NULL	NULL	NULL	NULL	2000 (4653)	Using where

Sekere van die waardes verskil tussen MyISAM-stoorenjin en InnoDB, en die waardes van InnoDB word in hakies getoon. Al die seleksie-tipes is "SIMPLE", wat aandui dat geen "UNIONS" en subnavrae gebruik is nie. 'n Tydelike tabel word vir die resultaat van die deursoek op die *internet*-tabel gebruik (aangedui deur "Using temporary"). Die tabel *tabel3* (en tabel *tabel2* vir InnoDB) toon geen inligting in die *ekstra*-kolom nie.

Tabel 6.62 toon die "EXPLAIN"-bevel se resultaat nadat die indeks bygevoeg is.

Tabel 6.62 – Die resultaat van die "EXPLAIN"-bevel vir Navraag 7 (Met indeks(e)).

id	Seleksie-tipe	Tabel	Tipe	Moontlike sleutels	Sleutel gebruik	Sleutel lengte	Verwysing	Verwagte Aantal rye	Ekstra
1	SIMPLE	t1	index	Tabel1	Tabel1	35	NULL	4741 (341605)	Using index Using temporary
1	SIMPLE	P	ref	Persoon1	Persoon1	22	Internet. t1.verbindkolom (Internet_innodb. t1.verbindkolom)	1 (70372)	Using where Using index
1	SIMPLE	t3	ref	Tabel3	Tabel3	4	Internet.t1.kolom1 (Internet_innodb. t1.kolom1)	1 (2045)	Using index
1	SIMPLE	t2	ref	Tabel2	Tabel2	4	Internet.t1.kolom1 (Internet_innodb. t1.kolom1)	1 (1899)	Using index
1	SIMPLE	i	ref	Internet1	Internet1	21	Internet.p.netwerkid (Internet_innodb. p.netwerkid)	374 (4653)	Using where Using index

Die resultaat van die indeksdeursoek van *tabel1* word in 'n tydelik tabel gestoor. Die waarde "index" van die *tipe*-kolom dui aan dat 'n volledige indeksdeursoek op die *tabel1*-tabel toegepas word. Die res van die rye se resultate stem ooreen met resultate van vorige navrae ten opsigte van die *tipe*-, *verwysing*- en *ekstra*-kolomme. Dit is duidelik uit die uitvoertyd dat die indeks 'n goeie keuse is, en die aantal verwagte rye reflekteer ook die resultaat.

6.4 SQL Profiler

Die paragraaf dui die voorstelle aan wat die indeksoptimeringsghoeroe maak nadat die werksladinglêer van SQL Profiler vir die ghoeroe gevoer is. Die werksladinglêer word geskep deur SQL Profiler deurdat al die seleksienavrae wat ondersoek is saam met 'n ekwivalente hoeveelheid bywerkings en byvoegings op elke tabel uitgevoer is. Die aantal bywerkings en byvoegings stem ooreen met die aantal tabelverwysings wat oor al die seleksienavrae geskied. Al die seleksies, bywerkings en byvoegings word 10 keer herhaal. Letters word in die plek van die werklike waardes gebruik en die letters stel heelgetalle, reële getalle, karakterstringe of selfs "NULL"-waardes voor. Die gebruik van aanhalingstekens dui karakterstringe aan.

Die volledige lys van instruksies wat gebruik is, word hier gelys:

```
SELECT * FROM internet WHERE userid = 'X'
GO
INSERT INTO internet VALUES (A,B,C,D,E,F,G,H)
GO
UPDATE internet SET bytes = A, sourceip = B WHERE userid = C
GO
SELECT * FROM internet WHERE userid = 'X' ORDER BY datetime DESC
GO
INSERT INTO internet VALUES (A,B,C,D,E,F,G,H)
GO

UPDATE internet SET bytes = A, sourceip = B WHERE userid = C
GO
SELECT *
FROM internet
WHERE userid IN (SELECT netwerkid FROM persoon WHERE voorletters = 'X')
ORDER BY datetime DESC
GO
INSERT INTO internet VALUES (A,B,C,D,E,F,G,H)
INSERT INTO persoon VALUES (A,B,C,D,E,F,G,H,I)
GO
UPDATE internet SET bytes = A, sourceip = B WHERE userid = C
UPDATE persoon SET noemnaam = A, titel = B, aktief = C WHERE netwerkid = D
GO

SELECT * FROM persoon WHERE van LIKE 'A%' AND titel = 'B'
union
SELECT * FROM persoon WHERE voorname LIKE 'C%' AND aktief = 'D'
GO
SELECT * FROM tabel1 t1 WHERE kolom2 IN (SELECT t2.kolom2 FROM tabel2 t2 WHERE
t2.verbindkolom = t1.kolom1)
GO
INSERT INTO tabel1 VALUES (A,B,C)
INSERT INTO tabel2 VALUES (A,B,C)
GO
```

```
UPDATE tabel1 SET kolom1 = A, kolom2 = B WHERE verbindkolom = C
UPDATE tabel2 SET kolom1 = A, kolom2 = B WHERE verbindkolom = C
GO

SELECT userid, sum(bytes) bytes, van, voorletters
FROM internet i JOIN persoon p ON i.userid = p.netwerkid
GROUP BY userid, van, voorletters
ORDER BY sum(bytes) DESC;
GO
INSERT INTO internet VALUES (A,B,C,D,E,F,G,H)
INSERT INTO persoon VALUES (A,B,C,D,E,F,G,H,I)
GO
UPDATE internet SET bytes = A, sourceip = B WHERE userid = C
UPDATE persoon SET noemnaam = A, titel = B, aktief = C WHERE netwerkid = D
GO

SELECT distinct Datetime, bytes, userid, van, voorname, taal, t1.kolom1, t2.kolom2,
t3.kolom1
FROM internet i
JOIN persoon p ON i.userid = p.netwerkid
JOIN tabel1 t1 ON p.netwerkid = t1.verbindkolom
JOIN tabel3 t3 ON t1.kolom1 = t3.verbindkolom
JOIN tabel2 t2 ON t1.kolom1 = t2.verbindkolom
GO
INSERT INTO internet VALUES (A,B,C,D,E,F,G,H)
INSERT INTO persoon VALUES (A,B,C,D,E,F,G,H,I)
INSERT INTO tabel1 VALUES (A,B,C)
INSERT INTO tabel2 VALUES (A,B,C)
INSERT INTO tabel3 VALUES (A,B,C)
GO
UPDATE internet SET bytes = A, sourceip = B WHERE userid = C
UPDATE persoon SET noemnaam = A, titel = B, aktief = C WHERE netwerkid = D
UPDATE tabel1 SET kolom1 = A, kolom2 = B WHERE verbindkolom = C
UPDATE tabel2 SET kolom1 = A, kolom2 = B WHERE verbindkolom = C
UPDATE tabel3 SET kolom1 = A, kolom2 = B WHERE verbindkolom = C
GO
```

Die uitdrukkings neem gemiddeld 1 minuut en 48 sekondes om uit te voer. Die werksladinglêer is ongeveer 2,434 megagrepe. Die lêer word vir die indeksoptimeringsghoeroe gevoer, en die resultaat is soos volg:

- Agt indekse word voorgestel:
  - 'n Gebondelde indeks op die *internet*-tabel se *userid*-kolom (*internet1*).
  - 'n Ontbondelde indeks op die *internet*-tabel se *userid*-, *bytes*- en *datetime*-kolomme (*internet5*).
  - 'n Gebondelde indeks op al die *persoon*-tabel se kolomme (*persoon2*).
  - 'n Ontbondelde indeks op die *persoon*-tabel se *netwerkid*- en *voorletters*-kolomme (*persoon6*).
  - 'n Gebondelde indeks op die *tabel3*-tabel se *verbindkolom*-kolom (*tabel33*).
  - 'n Gebondelde indeks op die *tabel1*-tabel se *verbindkolom*-kolom (*tabel14*).
- Al die indekse is stygend gesorteer.
- Die indekse korreleer redelik met die indekse wat op die individuele navrae toegepas is.
- Die ghoeroe voorspel 'n 10%-verbetering in die produktiwiteit van die onderskeie navrae.

Tabel 6.63 toon 'n tabel wat 'n verslag voorstel wat deur SQL Server geskep word. Die tabel dui die gebruik van elke indeks volgens die navrae in die werksladinglêer aan.

Tabel 6.63 – Persentasie navrae in werksladinglêer wat indekse gebruik

Indeks	Gebruik (%)	Grootte (kilogrepe)
internet1	33.3	54400
persoon2	14.8	5440
tabel33	7.4	768
tabel14	18.5	176
internet5	14.8	7248
persoon6	11.1	5440

Die indeks *internet1* word die meeste (33,3%) van al die ander indekse gebruik. Die indeks *tabel33* dui aan dat die derde indeks op *tabel3* gebruik is. Die indeks *tabel14* dui aan dat die vierde indeks wat probeer is, vir die tabel *tabel1* gebruik is. Die indeks *persoon2* is nie 'n goeie besluit van die indeksoptimeringsghoeroe nie aangesien dit al die kolomme van die *persoon*-tabel insluit.

6.5 Slotparagraaf

Die hoofstuk het 'n paar navrae se resultate ondersoek. Hoofstuk 6 maak gebruik van die riglyne soos uiteengesit in hoofstuk 4 (spesifiek paragraaf 4.4) om die keuses van indekse te steun. Moontlike indekse is vir elke navraag ondersoek en bespreek, en 'n keuse is gemaak tussen die onderskeie indekse. Verskeie metings is geneem, soos byvoorbeelde die uitvoertye, SVE-tye, aantal rekords en die aantal logiese lees-aksies vir SQL Server en die uitvoertye, aantal rekords en aantal verwagte rye vir MySQL. Die metings wat vir SQL Server van belang is, is die SVE-tyd en die aantal logiese lees-aksies. Die metings wat vir MySQL van belang is, is die uitvoertyd en die aantal rye wat MySQL verwag om te verwerk. Die vier metings word gebruik om navrae te beoordeel en te optimeer. Die program "SQL Profiler" se resultaat (in paragraaf 6.4) gee 'n aanduiding van 'n stelsel wat nader aan die werklikheid is aangesien verskeie navrae, bywerkings en byvoegings ondersoek word, en nie net seleksienavrae nie. Dit blyk dat die indekse wat uit die werksladinglêer verkry word redelik ooreenstem met die statiese databasis se ondersoek. Individuele navrae kan moontlik verbeter word deur eerder die indekse van paragraaf 6.3 te gebruik. Dit is duidelik dat die indeksoptimeringsghoeroe nie altyd die beste oplossing bied nie, dat die "EXPLAIN"-bevel van MySQL net 'n hulpmiddel is, en dat die keuse van indekse steeds deeglik oorweeg moet word vir beide databasisbestuurstelsels.

HOOFSTUK 7

SLOTHOOFSTUK

7.1 Inleiding

Die vorige hoofstukke het die teorie sowel as die eksperimentele aspekte van die verhandeling hanteer. Hierdie hoofstuk konsentreer op die samevatting van die resultate en die vergelyking van die resultate met die doelwitte wat in hoofstuk 1 gestel is. Die verhandeling het die optimering van navrae met behulp van databasisbestuurstelsels ondersoek. Daar is onderskeidelik gekyk na indekse as primêre optimeringstegniek, en die infrastruktuur van MySQL en SQL Server ten opsigte van optimering. Laastens is 'n ekperiment gedoen om die teorie en die praktiese aspekte bymekaar te bring.

Die proses om 'n databasis en navrae te optimeer, is 'n proses wat begin by die konsepsiële ontwerp van die databasis en die opstelling van die databasis se parameters, en deurvolg tot by die resultaat van die navraag. Dit wil voorkom of eenvoudige stelsels normaalweg voldoende produktiwiteit kan verkry met slegs indekse. Groter stelsels vereis egter meer indieptekennis van beide die databasisbestuurstel en die navrae wat moontlik op die databasis uitgevoer gaan word. Optimeringshulpmiddels soos SQL Server se indeksoptimeringsghoeroe bied kragtige hulpbronne waarmee die meeste databasisse se optimeringsbehoeftes bevredig sal word. Dit blyk egter dat gevalle voorkom waar die totale proses van ontwerp tot onderhoud breedvoerig bestudeer moet word om die effektiefste optimeringstegnieke te kan toepas.

Hardeware is normaalweg die antwoord en ook die duur komponent in die databasisstelsel. Die algemene konsep van opgradering van hardeware sal in die meeste gevalle die optimeringsvereistes kan hanteer. Die ontwerp van die databasis en die formulering van navrae kan egter werkverrigtingsprobleme veroorsaak indien die ontwerp en formulering nie behoorlik ondersoek is nie. In die meeste gevalle kan goeie databasisontwerp en 'n goeie formulering van navrae 'n groter verhoging in produktiwiteit as hardeware lewer. Die werksopdrag van 'n databasisadministrateur behels normaalweg ook optimering.

As programmeerder, hanteer die skrywer van die verhandeling daagliks navrae en databasisse. Uit ervaring blyk dit dat die teorie optimering net só ver kan neem voordat 'n ervaringskomponent intree. Die ervaringskomponent stem ooreen met logiese positivisme in terme van die eksperimentering. Die teorie is normaalweg gebaseer op veralgemening, en spesifieke gevalle mag buite die norm beweeg.

Die riglyne vir optimering (sien hoofstuk 4) wat in die verhandeling voorgestel is, moet dus nie as volledig beskou word nie. Die riglyne mag egter in die praktyk nie voldoende wees om aan al die produktiwiteitsvereistes te voldoen nie. Ongeag die graad van optimering gaan 'n stelsel met miljoene rekords steeds 'n tyd neem vir komplekse navrae. Normaalweg word gestreef na aanvaarbare diensvlakke, en tyd en geld word gespandeer om die aanvaarbare vlakke te bereik.

'n Algemene wanbegrip wat bestaan is dat optimering met die skep van die laaste indeks afgehandel is. Alle databasisse benodig onderhoud om in 'n optimale toestand te bly. Nuwe kolomme kan bygevoeg word of bestaande kolomme kan verwyder word. Aangesien die databasisbestuurstelsels meestal benaderde waardes gebruik, kan dit gebeur dat die stelsel nie die mees optimale plan kies nie, in welke geval die databasisadministrateur moet ingryp om produktiwiteit te behou. Optimering beïnvloed elke aspek van 'n databasis, en kennis oor die onderwerp moet voortdurend uitgebrei word soos wat nuwe tegnieke beskikbaar raak.

In paragraaf 7.2 van hierdie hoofstuk word die opsomming met betrekking tot die doelwitte kortliks bespreek.

7.2 Opsomming

'n Spesiale bylaag (bylaag B) is ingesluit om die terme wat nie normaalweg in Afrikaans beskikbaar is nie te verduidelik. Die interne argitektuur van beide SQL Server (sien paragraaf 3.2.3) en MySQL (sien paragraaf 3.3.3) is ondersoek, wat die stoorenjins van SQL Server (sien paragraaf 3.2.3.6) en MySQL (sien paragraaf 3.3.3.1) insluit. SQL Server se stoorenjin is meer kompleks as enige van MySQL se stoorenjins aangesien SQL Server se stoorenjin bestaan uit verskeie bestuurders (sien paragrawe 3.2.3.6.1 tot 3.2.3.6.10). MySQL het egter verskeie stoorenjins wat elkeen geskik is vir spesifieke take (sien paragrawe 3.3.3.1.1 tot 3.3.3.1.10). Verskeie optimeringstegnieke en ook die navraagoptimeerder is vir elke databasisbestuurstelsel ondersoek (sien paragraaf 3.2.7.2 vir SQL Server en paragraaf 3.3.5 vir MySQL), en optimeringsriglyne wat van die meer algemene tegnieke saamvat, is beskikbaar gestel (sien hoofstuk 4).

Verskeie toepassings (sien paragraaf 3.2.8 vir SQL Server en paragraaf 3.3.6 vir MySQL), soos SQL Query Analyzer, SQL Profiler, Index Tuning Wizard, MySQL Query Browser en die "EXPLAIN"-bevel, is bestudeer en gebruik in die eksperimentele fase van die verhandeling.

Die fisiese opstelling van die eksperiment is volledig beskryf in hoofstuk 5. Die "stadiger" rekenaar het voldoende resultate gelewer en die besluit op die rekenaar is geregverdig. Die

resultate het meetbare verskille getoon wanneer indekse toegepas is. 'n Model is ontwikkel (sien paragraaf 6.2) om die resultate volgens 'n spesifieke formaat te meet, sodat 'n vergelyking tussen die verskillende opsies in terme van die keuses van indekse verkry kan word.

Navrae se optimering stem nie altyd volledig ooreen met die teorie nie – soos duidelik blyk uit Navraag 1 en Navraag 2 waar die ontbondelde indeks vinniger is as die gebondelde indeks (sien paragrawe 6.3.1 en 6.3.2). Dit het ook duidelik na vore gekom dat SQL Server se uitvoertyd nie betroubaar is as 'n maatstaf vir optimering nie en dat daar eerder na die SVE-tyd en die aantal logiese lees-aksies gekyk moet word. MySQL se uitvoertyd kan saam met die aantal verwagte rye gebruik word as 'n maatstaf vir optimering. Die indeksoptimeringsghoeroe van SQL Server bied normaalweg 'n goeie resultaat en die voorgestelde indeks is vir die meeste situasies geskik. In sommige gevalle is die voorgestelde indeks egter nie die optimale indeks nie en moet die indeks heroorweeg word. Die resultate van paragraaf 6.3 word in tabel 7.1 saamgevat:

Tabel 7.1 – Samevatting: Resultate van die eksperimentele fase van hoofstuk 6.

Navraag	Ghoeroe se voorspelde verbetering	Indeks verbetering - SQL Server	Indeks verbetering - MySQL (normaliseringswaarde)	Beskrywing
1	91%	90%	0.667 (0.875)	Die gekose indeks stem ooreen met die ghoeroe se indeks.
2	60%	Nie van toepassing	1.000 (1.300)	Dit blyk dat hierdie navraag se resultate nie na verwagting was nie.
3	25%	25%	0.019 (0.014)	In hierdie geval het die indeks 'n groter verbetering vir MySQL opgelewer as SQL Server.
4	47%	46%	0.906 (0.810)	Gekose indeks verskil van ghoeroe, maar lewer soortgelyke resultate. InnoDB is beter as MyISAM.
5	Geen	8%	0.013 (0.014)	Indeks lewer 'n klein verbetering vir SQL Server.
6	18%	6%	0.004 (0.012)	Ghoeroe se voorstel is beter, maar dalk onprakties.
7	Geen	26%	0.023 (0.024)	Gekose indeks lewer goeie verbetering vir beide SQL Server en MySQL.

Die onderskeie toepassings van die twee databasisbestuurstelsels bied 'n groot bron van inligting en hulp, wat in baie gevalle voldoende behoort te wees. Uit tabel 7.1 blyk dit dat die ghoeroe nie altyd die regte keuse maak nie, maar bied in meeste gevalle 'n goeie keuse. Verdere navorsing kan moontlik gedoen word om spesifieke optimeringsalgoritmes te ontwikkel om sodoende miskien meer effektiewe resultate te verkry. Die moontlikheid bestaan ook om 'n besluitsteunstelsel te ontwikkel wat die keuses van indekse kan vergemaklik.

Databasisse en databasisbestuurstelsels sal vir 'n geruime tyd deel uitmaak van die moderne rekenaarwêreld en sal groei in terme van kompleksiteit, werkverrigting en dienslewering.

BYLAAG A

ALGEMENE SQL SERVER- EN MYSQL-NAVRAAGKONSEPTE EN -BEVELE

1. Inleiding

SQL Server gebruik die "ANSI (American National Standards Institute) SQL-92"-standaard as basis (Delaney, 2001:34) en het sekere uitbreidings op die standaard. MySQL gebruik ook die SQL-standaard, maar is nie tot SQL-92 beperk nie. MySQL maak gebruik van sekere elemente van SQL-1999 en SQL-2003 (MySQL AB, 2005:5). Die eindprodukte van elkeen is steeds baie dieselfde, met relatief klein verskille. Die onderskeie databasisbestuurstelsels het slegs bykomende funksionaliteit bygevoeg. Aangesien hierdie twee SQL-tale so min verskil, word SQL Server se navraagtaal kortliks verduidelik met, waar nodig, 'n verwysing na MySQL se navraagtaal wat uit MySQL AB (2005:694-818) volg. MySQL mag dalk bykomende funksionaliteit besit wat nie in die volgende verduideliking voorkom nie.

Paragraaf 2 in hierdie bylaag bevat 'n kort beskrywing van die sintaksis wat in die bylaag gebruik word. Paragraaf 3 verduidelik die basiese selekteerbevel. Paragraaf 4 beskryf die byvoegbevel, en paragraaf 5 in die bylaag beskryf kortliks die bywerkbevel.

2. Notasie

Die notasie is die notasie van SQL Server se "Transact SQL" en word in Tabel A1 getoon:

Tabel A1 – SQL Server-notasie

Simbool	Betekenis
[...]	Opsionele terme
|	Vervang met logiese "of"
A ::=	Beskrywing van voorafgaande term A
< ... >	Stel een of meer verpligte terme voor
{ ... }	Groep van moontlikhede wat gebruik mag word.

3. Die basiese selekteerbevel

Die bevel word gebruik om inligting uit die databasis te onttrek. Die verduideliking van die bevel volg aan die hand van Microsoft Corporation (2005g) se beskrywing.

Die sintaksis lyk soos volg:

SELECT *select_list*
[INTO *new_table*]
FROM *table_source*
[WHERE *search_condition*]
[GROUP BY *group_by_expression*]
[HAVING *search_condition*]
[ORDER BY *order_expression* [ASC | DESC]]

3.1 Die "SELECT"-gedeelte van die selekteerbevel

Die gedeelte spesifiseer die kolomme wat vertoon moet word.

Sintaksis:

SELECT [ALL | DISTINCT]
  [TOP *n* [PERCENT] [WITH TIES]]
  < select_list >

< select_list > ::=
  { *
    | { *table_name* | *view_name* | *table_alias* }.*
    | { *column_name* | *expression* | IDENTITYCOL | ROWGUIDCOL }
      [[AS] *column_alias*]
    | *column_alias* = *expression*
  } [,...*n*]

Argumente:

- **"ALL"** – Alle moontlike rye word vertoon. Die argument is ook die verstekwaarde.
- **"DISTINCT"** – Slegs unieke rye mag in die resultaat voorkom.
- **"TOP *n* [PERCENT]"** – Slegs die eerste *n* rye word in die resultaat vertoon. Indien die persentasie-argument gebruik word, word slegs *n* persent van die resultaat vertoon. MySQL verskil hier in dié opsig dat die argument "LIMIT *n*" is en dat dit aan die einde van die selekteerbevel se sintaksis voorkom.
- **"WITH TIES"** – Addisionele rye wat voldoen aan die "ORDER BY"-argument word vertoon, indien "TOP n [PERSENT]" gebruik word. MySQL bevat nie hierdie argument nie.

- **"< select_list >"** – Die kolomme (met kommas geskei) wat vertoon word, word in die argument gespesifiseer. Die "[,...*n*]"-gedeelte dui daarop dat verskeie kombinasies moontlik is. Die moontlike waardes vir die seleksie-argument sluit in:
  - "*" – Al die kolomme van die tabelle in die "FROM"-gedeelte word vertoon.
  - "{ table_name | view_name | table_alias }.*" – Al die elemente van die betrokke tabel of skyntabel word vertoon. ('n Skyntabel is 'n navraag wat vooraf opgestel is, gestoor is en as 'n tydelike tabel funksioneer.)
  - "column_name" – Die naam van die kolom om te vertoon. Indien kolomme van twee tabelle ooreenstem, moet die tabel van die kolom gespesifiseer word – soortgelyk aan die sintaksis vir die "*"-argument in die punt hierbo.
  - "expression" – Die uitdrukking bestaan uit konstantes, funksies, konstantes wat met 'n operator (byvoorbeeld "+","-","*", en "/") aan funksies verbind is, verskeie kombinasies van kolomme of 'n selekteerbevel binne 'n ander selekteerbevel, wat bekend staan as 'n subnavraag.
  - "IDENTITYCOL" – Vertoon die identiteitskolom. Die kolom is 'n teller van die aantal rye wat elke keer vermeerder wanneer 'n rekord bygevoeg word. MySQL besit nie die argument nie.
  - "ROWGUIDCOL" – Vertoon die unieke kolom wat vir enige databasis, en selfs ander rekenaars met databasisse, uniek is. MySQL besit nie die argument nie.
  - "column_alias" – Elke kolom kan 'n skuilnaam kry waarmee na die kolom in die sintaksis verwys kan word. Die gebruik van die "AS"-deel is opsioneel.

3.2 Die "INTO"-gedeelte van die selekteerbevel

Die gedeelte neem die geselekteerde rekords vanaf die selekteerbevel en voeg dit in 'n nuwe tabel in. Die tipes vir elke kolom word bepaal deur die geselekteerde kolom. MySQL ondersteun die formaat "INSERT ... SELECT" (MySQL AB, 2005:723-724).

Sintaksis:

[INTO *new_table*]

Argumente:

- **"new_table"** – Die argument spesifiseer die tabelnaam van die tabel waar die rekords van die selekteerbevel gestoor moet word.

3.3 Die "FROM"-gedeelte van die selekteerbevel

Die gedeelte spesifiseer die bron waarvandaan die rekords verkry word. Die argument word volledig verduidelik vir die selekteer, verwyder en bywerkbevele.

Sintaksis:

[FROM { < table_source > } [,...*n*]]

< table_source > ::=
  table_name [[AS] *table_alias*] [WITH (< table_hint > [,...*n*])]
  | *view_name* [[AS] *table_alias*] [WITH (< view_hint > [,...*n*])]
  | *rowset_function* [[AS] *table_alias*]
  | *user_defined_function* [[AS] *table_alias*]
  | *derived_table* [AS] *table_alias* [(*column_alias* [,...*n*])]
  | < joined_table >

< joined_table > ::=
  < table_source > < join_type > < table_source > ON < search_condition >
  | < table_source > CROSS JOIN < table_source >
  | [( ] < joined_table > [ )]

< join_type > ::=
  [INNER] | { { LEFT | RIGHT | FULL } [OUTER] }]
  [< join_hint >]
  JOIN

< table_hint > ::=
  { INDEX (*index_val* [,...*n*])
    | FASTFIRSTROW
    | HOLDLOCK
    | NOLOCK
    | PAGLOCK
    | READCOMMITTED
    | READPAST
    | READUNCOMMITTED
    | REPEATABLEREAD
    | ROWLOCK
    | SERIALIZABLE

| TABLOCK
| TABLOCKX
| UPDLOCK
| XLOCK
}
< view_hint > ::=
{ NOEXPAND [, INDEX (*index_val* [,...*n*])] }

< join_hint > ::=
{ LOOP | HASH | MERGE | REMOTE }

Argumente:

- **"< table_source >"** – Die argument kan tabelle, skyntabelle, afgeleide tabelle (tabelle wat verkry is deur twee selekteerbevele wat op mekaar toegepas word) of verbinde tabelle (tabelle wat verbind is met een of meer gemeenskaplike kolomme) spesifiseer vir die selekteerbevel.
  - "table_name" – Dit is die naam van 'n tabel.
  - "[AS] table_alias" – Stel 'n skuilnaam vir die betrokke tabel of skyntabel voor, wat gebruik word in die sintaksis om na die betrokke tabel te verwys.
  - "WITH (< table_hint >)" – Die argument laat toe dat optimeringstegnieke aan die ingeboude optimeerder voorgestel word vir die betrokke navraag. Die tegnieke sluit in tabelsoektogte, een of meer indekse en sluittegnieke (word gebruik om te keer dat verskillende veranderings oorvleuel ten opsigte van hulpbronne). MySQL gebruik die sintaksis "USE INDEX (key_list)" (MySQL AB, 2005:740-742). Al die argumente wat vir "table_hint" gespesifiseer is, behalwe die gedeelte oor indekse, het betrekking op sluittegnieke.
  - "view_name" – Die naam van die skyntabel.
  - "WITH (view_hint >)" – Die argument bepaal 'n deursoek van 'n geïndekseerde skyntabel (skyntabel wat 'n gebondelde indeks bevat). Die deel is slegs betrokke by die selekteerbevel. Die term "NOEXPAND" spesifiseer dat die skyntabel soos 'n tabel met 'n gebondelde indeks gebruik word. Die MySQL-weergawe wat gebruik word, bevat nie hierdie opsie nie.
  - "rowset_function" – Die funksies verskaf 'n objek wat soos 'n tabel gebruik kan word. Die funksies is egter buite die bestek van die verhandeling. Dit sluit funksies in soos "OPENXML", wat 'n "XML (Extensible Markup Language)"-dokument kan oopmaak en as 'n tabel kan weergee. MySQL bied nog nie hierdie funksionaliteit nie.
  - "user_defined_function" – Die gebruiker kan self 'n funksie ontwerp en hier gebruik. MySQL ondersteun nie hierdie funksie nie.

- "derived_table" – 'n Subnavraag wat rye van 'n databasis selekteer en wat dan as invoer dien vir die buitenste navraag.
- "column_alias" – Skuilname vir die kolomme in die afgeleide tabel.

- **"< joined_table >"** – Die resultaat wat verkry is indien twee tabelle verbind word.
  - "< join_type >" – Die tipe verbinding wat gebruik word. 'n Meer breedvoerige verduideliking volg onder.
  - "ON <search_condition>" – Verskaf die kriteria waarvolgens verbind word.
  - "CROSS JOIN" – Die argument laat 'n kruisproduk van die tabelle as resultaat verkry word, met ander woorde elke ry van die een tabel word teenoor elke ry van 'n ander tabel vertoon.
- **"< join_type >"** – Die volgende verbindings is beskikbaar:
  - "INNER" – Laat slegs rekords toe uit beide tabelle waarvan die kolomme se waardes ooreenstem volgens die verbindingskriteria. Hierdie tipe is die verstektipe.
  - "FULL [OUTER]" – Kombineer die resultate van die vorige verbinding met al die rekords wat nie aan die verbindingskriteria voldoen nie. Die tabel wat nie aan die kriteria voldoen nie se kolomme vertoon dan as "NULL". Die weergawe van MySQL, wat ondersoek word, maak nie voorsiening vir hierdie volledige verbinding nie. Dit is egter op die lys om geïmplementeer te word vanaf weergawe 5.1. (MySQL AB, 2005:16)
  - "LEFT [OUTER]" – Al die rye van die linkerkantste tabel wat nie aan die kriteria voldoen nie, word vertoon met "NULL" as waardes. Uit die regterkantste tabel vertoon slegs die rye wat wel aan die kriteria voldoen.
  - "RIGHT [OUTER]" – Die argument is dieselfde as die vorige verbinding, maar net vir die regterkantste tabel.
  - "< join_hint >" – Dit bepaal die verbindingstrategie wat deur die ingeboude navraagoptimeerder gebruik word vir die verbinding. MySQL bied nie hierdie opsie nie. Die beskrywing volg onderaan.
- **"< join_hint >"** – Die tipe verbinding wat toegepas moet word.
  - "LOOP | HASH | MERGE" – Enigeen van die metodes kan gespesifiseer word. Die verbindingmetodes is in die verhandeling beskryf (sien paragrawe 3.2.7.3.3.1, 3.2.7.3.3.2 en 3.2.7.3.3.3).
  - "REMOTE" – Die verbinding vind plaas waar die bediener, wat die regterkantste tabel besit, nie dieselfde is as die ander tabelle se bediener nie.

3.4 Die "WHERE"-gedeelte van die selekteerbevel

Die gedeelte stel die voorwaarde(s) vas waaraan die rekords moet voldoen om deur die selekteer-, verwyder- of bywerkbevel gebruik te word. Die argument word volledig verduidelik

vir die selekteer-, verwyder- en bywerkbevele, en maak addisioneel gebruik van die verwysing Microsoft Corporation (2005h).

Sintaksis:

[WHERE < search_condition > | < old_outer_join >]
< old_outer_join > ::=
column_name { * = | = * } *column_name*

< search_condition > ::=
{ [NOT] < predicate > | (< search_condition >) }
[{ AND | OR } [NOT] { < predicate > | (< search_condition >) }]
} [,...*n*]
< predicate > ::=
{ *expression* { = | < > | ! = | > | > = | ! > | < | < = | ! < } *expression*
| *string_expression* [NOT] LIKE *string_expression*
[ESCAPE '*escape_character*']
| *expression* [NOT] BETWEEN *expression* AND *expression*
| *expression* IS [NOT] NULL
| CONTAINS
({ *column* | * } , '< contains_search_condition >')
| FREETEXT ({ *column* | * } , '*freetext_string*')
| *expression* [NOT] IN (*subquery* | *expression* [,...*n*])
| *expression* { = | < > | ! = | > | > = | ! > | < | < = | ! < }
{ ALL | SOME | ANY} (*subquery*)
| EXISTS (*subquery*)
}

Argumente:

- **"< search_condition >"** – Beperk die resultaat van die selekteer-, byvoeg- of bywerkbevele volgens die voorwaardes hier gestel.
  - "NOT" – Gee die negatief van die opvolgende voorwaarde deur.
  - "AND" – Verbind twee voorwaardes en gee 'n waar-resultaat slegs as beide voorwaardes bevredig is.
  - "OR" – Verbind twee voorwaardes en gee 'n waar-resultaat indien enige voorwaarde bevredig is.
- **"< predicate >"** – 'n Uitdrukking of voorwaarde wat waar, vals of onbekend teruggee.

- "expression" – Die term kan kombinasies van kolomname, konstantes, funksies, veranderlikes of skalaarsubnavrae wees wat almal verbind is met operator(e) (byvoorbeeld "+", "-", "*" en "/") of subnavrae.
- "=" – Toets gelykheid tussen die uitdrukkings.
- "<>", "!=" – Toets ongelykheid tussen die uitdrukkings.
- ">", ">=", "!>", "<", "<=", "!<" – Toets onderskeidelik groter as, groter en gelyk, nie groter nie, kleiner as, kleiner en gelyk, en nie kleiner nie tussen die uitdrukkings.
- "string_expression" – Stel 'n uitdrukking voor wat bestaan uit 'n groepering van karakters en "wildcard"-karakters. "Wildcard"-karakters sluit die volgende in:
  - "%" – Enige groep van nul of meer karakters.
  - "_" – Enige een karakter.
  - "[...]" – Enige karakter of karakters kan in die blokhakies gebruik word. Slegs daardie karakter(s) mag dan voorkom. Byvoorbeeld [a-f] of [abcdef] (Laat slegs een karakter toe wat in die reeks a-f is.)
  - "[^ ...]" – Dieselfde as "[ ]", behalwe dat die karakters nie mag voorkom nie.
- "[NOT] LIKE" – Die karakterstring wat hierna volg, moet gebruik word om 'n patroon te identifiseer wat moontlik kan voorkom in die karakterstring wat voor die term is. MySQL besit ook nog 'n argument "REGEX" wat toelaat dat "regular expressions" gebruik kan word. SQL Server besit nie hierdie funksionaliteit nie.
- "ESCAPE 'escape_charater'" – Die gedeelte laat toe dat die "wildcard"-karakters normaal gebruik kan word en nie as 'n "wildcard"-karakter geïnterpreteer word nie.
- "[NOT] BETWEEN" – Die argument word gebruik om 'n reeks waardes, waaraan die rye moet voldoen, te spesifiseer.
- "IS [NOT] NULL" – Sekere kolomme mag die waarde "NULL" bevat. Hierdie argument toets vir die "NULL"-waarde.
- "CONTAINS" – Die argument gee rye terug wat, of presies of min of meer ooreenstem met die karakters, woorde of sinne verskaf. Die argument "MATCH" word in MySQL gebruik om min of meer dieselfde te doen.
- "FREETEXT" – Soek vir rye wat eerder die betekenis van die sin volg as die presiese woorde van die predikaat.
- "[NOT] IN" – Soek vir rye wat in (of nie in) 'n spesifieke lys van waardes is (nie). Die lys van waardes mag ook 'n subnavraag wees. Die subnavraag is 'n beperkte selekteerbevel wat nie die argumente "ORDER BY", "COMPUTE" en "INTO" bevat nie. SQL Server laat toe dat net een kolom vooraf gebruik kan word in die argument. MySQL neem hierdie argument verder deurdat daar meer as een kolom op hierdie manier vergelyk kan word.
- "ALL" – Die argument word saam met gelykheidstoetse (toetse wat byvoorbeeld ">" en "<" insluit) en 'n subnavraag gebruik. Die argument toets dat al die waardes wat

terugkom vanaf die subnavraag die gelykheidstoets slaag en vertoon dan slegs die rye wat hieraan voldoen.

- "{ SOME | ANY }" – Die argument word ook gebruik saam met 'n gelykheidstoets en 'n subnavraag. Enige waarde uit die subnavraag wat aan die toets voldoen se ekwivalente ry word vertoon.
- "EXISTS" – Toets die subnavraag se bestaan, met ander woorde of daar enige rye teruggekry word.

- **"< old_outer_join >"** – Gebruik die simbole "*=" en "=*"om 'n linkerkantste of regterkantste verbinding te maak tussen twee tabelle. Die manier is egter nie volgens die sql99-standaard nie en word nie in die verhandeling gebruik nie. MySQL bied nie ondersteuning vir hierdie simbole nie. Die normale verbinding waar net 'n "="-simbool gebruik word, sal wel in die verhandeling gebruik word. MySQL ondersteun wel die "=" sintaksis.

3.5 Die "GROUP BY"-gedeelte van die selekteerbevel

Hierdie argumente groepeer die rekords in spesifieke groepe en laat toe dat die selekteerbevel groeperingsfunksies gebruik. Elke kolom in die selekteer gedeelte wat nie in 'n groeperingsfunksie is nie moet in die "GROUP BY"-gedeelte wees. MySQL laat egter toe dat kolomme gespesifiseer word wat nie in die "GROUP BY"-gedeelte is nie (MySQL AB, 2005:737). Indien "ORDER BY" nie gebruik word vir SQL Server nie, is die rekords nie gesorteer nie.

Sintaksis:

[GROUP BY [ALL] *group_by_expression* [,...*n*]
    [WITH { CUBE | ROLLUP }]
]

Argumente:

- **"ALL"** – Die argument sluit alle groepe, resultate en rye in wat nie voldoen aan die voorwaardes in die "WHERE"-argument nie ("NULL"-waardes word gebruik vir hierdie rye).
- **"group_by_expression"** – Die argument is 'n uitdrukking waarop groepering plaasvind. Die argument kan 'n kolom wees, of 'n funksie (wat nie 'n groeperingsfunksie is nie) wat na 'n kolom verwys.
- **"CUBE"** – 'n Opsomming van die "GROUP BY"-argument se waardes word vertoon. MySQL bied nie hierdie opsie nie.
- **"ROLLUP"** – Ongeveer dieselfde as die "CUBE"–argument, met die verskil dat die groepe in hiërargiese orde vertoon word – volgens die orde waarin die kolomme in die "GROUP BY"-argument voorkom.

3.6 Die "HAVING"-gedeelte van die selekteerbevel

Die gedeelte soek rekords volgens die gegewe voorwaarde en soek hoofsaaklik volgens groepe en groeperingsfunksies. Die gedeelte word gewoonlik saam met die "GROUP BY"-gedeelte gebruik, maar werk soos 'n "WHERE"-gedeelte indien "GROUP BY" nie gebruik is nie.

Sintaksis:

[HAVING < search_condition >]

Argument:

- **"< search_condition >"** – Die argument kan op dieselfde manier opgebou word as die "WHERE"-gedeelte se voorwaarde.

3.7 Die "ORDER BY"-gedeelte van die selekteerbevel

Die gedeelte sorteer die rekords volgens die orde wat gegee is.

Sintaksis:

[ORDER BY { *order_by_expression* [ASC | DESC] } [,...*n*]]

Argumente:

- **"order_by_expression"** – Die argument gee die kolom, kolomskuilnaam, uitdrukking of 'n positiewe heelgetal wat die posisie van die kolom, kolomskuilnaam of uitdrukking verteenwoordig in die selekteergedeelte. Die orde van die elemente bepaal hoe die rekords gesorteer word. "NULL"-waardes is die laagste moontlike waarde.
- **"ASC"** – Stygende volgorde.
- **"DESC"** – Dalende volgorde.

4. Die basiese byvoegbevel

Die bevel word gebruik om 'n nuwe rekord by te voeg. Die verduideliking volg aan die hand van die verwysing van Microsoft Corporation (2005i).

Sintaksis:

INSERT [INTO]
{ *table_name* WITH (< table_hint_limited > [...*n*])
| *view_name*
| *rowset_function_limited*

```
    }

    {   [ ( column_list ) ]
        { VALUES
            ( { DEFAULT | NULL | expression } [ ,...n] )
            | derived_table
            | execute_statement
        }
    }
    | DEFAULT VALUES
< table_hint_limited > ::=
    { FASTFIRSTROW
        | HOLDLOCK
        | PAGLOCK
        | READCOMMITTED
        | REPEATABLEREAD
        | ROWLOCK
        | SERIALIZABLE
        | TABLOCK
        | TABLOCKX
        | UPDLOCK
    }
```

Argumente:

- **"[INTO]"** – Opsionele sleutelwoord wat tussen die "INSERT"-deel en die bestemmingtabel kom.
- **"table_name"** – Die naam van die tabel of tabelveranderlike wat die data ontvang.
- **"WITH (< table_hint_limited > [...n])"** – 'n Verkleinde weergawe van die argument wat in die "FROM"-gedeelte, soos voorheen bespreek, voorkom, word hier gebruik.
- **"view_name"** – Skuilnaam van die skyntabel. Skyntabelle mag 'n tabel slegs bywerk indien die kolomme wat bygewerk word uit slegs een tabel kom.
- **"rowset_function_limited"** – Die gedeelte val buite die bestek van die verhandeling.
- **"(column_list)"** – Lys die kolomme waarvoor daar data bygevoeg kan word. Indien 'n kolom nie in die lys is nie, moet die kolom 'n verstekwaarde besit (ook datumtipe), 'n teller wees of in staat wees om "NULL"-waardes te hanteer. Indien hierdie gedeelte uitgelaat word, moet die byvoegwaardes met al die kolomme van die tabel ooreenstem.
- **"VALUES"** – Sleutelwoord wat aandui dat die kolom se waardes hierna volg.

- **"DEFAULT"** – Indien 'n waarde nie gespesifiseer word vir 'n kolom nie, word hierdie waarde gebruik.
- **"expression"** – Verteenwoordig 'n konstante, 'n veranderlike of 'n uitdrukking, en mag nie "SELECT"- of "EXECUTE"-uitdrukkings wees nie.
- **"derived_table"** – Stel 'n geldige selekteerbevel voor waarvan die data in die tabel bygevoeg kan word.
- **"execute_statement"** – Geldige "EXECUTE"-bevele (voer SQL-uitdrukkings uit) word hier toegelaat. MySQL bevat nie hierdie argument nie.
- **"DEFAULT VALUES"** – Die nuwe rekord word gedwing om die verstekwaardes te gebruik eerder as die waardes wat gespesifiseer word. MySQL bevat nie hierdie argument nie.

5. Die basies bywerkbevel

Die bevel word gebruik om 'n bestaande rekord by te werk. Aangesien die meeste van die argumente alreeds voorgekom het in die vorige bevele, gaan slegs die argumente wat nog nie verduidelik is nie, hier beskryf word. Die verduideliking volg aan die hand van die verwysing Microsoft Corporation (2005j):

Sintaksis:

```
UPDATE
    {
     table_name WITH ( < table_hint_limited > [ ...n ] )
     | view_name
     | rowset_function_limited
    }
    SET
    { column_name = { expression | DEFAULT | NULL }
    | @variable = expression
    | @variable = column = expression } [ ,...n ]

  { { [ FROM { < table_source > } [ ,...n ] ]
    [ WHERE
       < search_condition > ] }
    |
    [ WHERE CURRENT OF
    { { [ GLOBAL ] cursor_name } | cursor_variable_name }
    ] }
    [ OPTION ( < query_hint > [ ,...n ] ) ]
```

< query_hint > ::=

```
{   { HASH | ORDER } GROUP
    | { CONCAT | HASH | MERGE } UNION
    | {LOOP | MERGE | HASH } JOIN
    | FAST number_rows
    | FORCE ORDER
    | MAXDOP
    | ROBUST PLAN
    | KEEP PLAN
}
```

Argumente:

- **"SET"** – Sleutelwoord wat gebruik word om die kolomme se nuwe waardes te spesifiseer.
- **"@variable"** – Die argument stel 'n veranderlike voor wat die waarde aanneem van die uitdrukking wat daarna volg. MySQL besit nie hierdie argument nie.
- **"CURRENT OF"** – Argument wat aandui dat die bywerkbevel plaasvind by die huidige posisie van die wyser ("cursor": 'n Tegniek wat toelaat dat rekords een ry op 'n slag verwerk kan word (Microsoft Corporation, 2005b)). MySQL 4.1 implementeer nog nie wysers nie.
- **"GLOBAL"** – Dui op 'n globale wyser.
- **"cursor_name"** – Die naam van die wyser.
- **"cursor_variable_name"** – Die naam van 'n wyserveranderlike wat verwys na 'n wyser.
- **"OPTION (< query_hint > [,...n])"** – Gee vir die optimeerder inligting oor die manier om die bywerk uitdrukking te verwerk. MySQL ondersteun nie hierdie argumente nie.
  - "{ HASH | ORDER } GROUP" – Die groeperingsfunksies in "GROUP BY" moet óf "hashing" óf sortering gebruik.
  - "{ LOOP | MERGE | HASH | } JOIN" – Die hele navraag moet 'n herhaal- ("loop"), saamvoeg- ("merge") of 'n "hash"-verbinding gebruik.
  - " {MERGE | HASH CONCAT } UNION" – Alle "UNION"-bewerkings (voeg twee selekteerbevele se resultate saam) moet "merging", "hashing" of saamvoeg gebruik.
  - "FAST number_rows" – Optimeer die navraag vir die aantal rye gespesifiseer.
  - "FORCE ORDER" – Die verbindvolgorde wat in die navraag gebruik word, moet behoue bly gedurende die navraagoptimering.
  - "MAXDOP number" – Die argument het te make met die aantal rekenaarverwerkers wat gebruik word in parallelleplanuitvoering en is nie van toepassing nie.
  - "ROBUST PLAN" – Die optimeerder word gedwing om 'n plan te volg wat werk vir die grootste moontlike rygrootte – selfs al verminder dit die werkverrigting.
  - "KEEP PLAN" – Verminder die aantal kere wat 'n navraag verwerk word wanneer daar verskeie bewerkings op die tabel plaasvind.

BYLAAG B

'N LYS VAN AFRIKAANSE EN ENGELSE TERME

1. Inleiding

Die bylaag lys 'n paar konsepte van die verhandeling wat beter bekend is in die Engelse taal. Elke konsep word kortliks beskryf en 'n paragraaf-verwysing waar die konsep gebruik word, word ook ingesluit. Die Afrikaanse terme wat in hierdie verhandeling gebruik is, is gebaseer op die Stigting vir Bemagtiging deur Afrikaans (2001) se terme.

2. Woordelys

Tabel B1 bevat die alfabetiese woordelys, Engelse vertaling, beskrywings en die paragraafverwysings:

Tabel B1 – 'n Lys van Afrikaanse en Engelse terme

Afrikaans	Engels	Beskrywing	Paragraaf
aanstuurwyser	"forwarding pointer"	'n Aanstuurwyser dui die posisie van 'n ry aan nadat die ry na 'n volgende bladsy geskuif het.	3.2.6.6.3
afgeleëprosedure-opdragte	"remote procedure calls"	Funksies wat oor netwerke en ander databasisbestuurstelsels geroep kan word.	3.2.8.3
atomisiteit	"atomicity"	Transaksies moet as een onskeibare eenheid van werk funksioneer.	3.3.3
begrensde boks	"bounding box"	Die boks dui die area aan wat elke ruimtelike data-objek beslaan.	2.5.1.1.3
beheerwyser	"handle"	'n Beheerwyser word gebruik om 'n navraag uit te voer wat vooraf berei is.	3.2.7.5
bevel	"attention"	'n Bevel wat SQL Server uitgee om 'n proses te begin of die einde van aan te kondig.	3.2.3.3
bevelontleder	"command parser"	Interpreteer bevele om prosesse te begin.	3.2.3.3
bevestig	"commit"	Bevestig transaksies sodat die transaksies finaal gestoor word en terugrol nie meer kan geskied nie.	3.3.2
bevestigde lees	"committed read"	Transaksie wag vir die bevestiging voordat data gelees word.	3.2.3.6.1
bufferarea	"buffer pool"	Stoorarea in die geheue.	3.2.3.6.8
data-elemente	"entries"	Elemente in 'n indeks.	2.4
defragmenteer	"defragment"	Die fragmentasie van indekse word verwyder.	3.2.6.4
deursigtigheid	"transparency"	Die gebruikers is onbewus van die opstelling ten opsigte van die ladingbalansering wat gebruik word.	4.7
deursnee	"intersection"	Die rye van twee tabelle wat oorvleuel en met ander woorde in beide tabelle voorkom.	3.3.5.2
digte indeks	"dense index"	'n Digte indeks bevat ten minste een data-element vir elke	2.4.1

		soek-sleutelwaarde in 'n rekord van 'n tabel.	
dooiepunte	"deadlock"	Dooiepunte ontstaan wanneer verskeie transaksies slotte op data verkry en twee of meer transaksies dieselfde slotte op die data in verskillende volgordes wil verkry.	4.6
dubbelgeskakelde lys	"doubly linked list"	Elke element in die lys besit wysers na die voorafgaande en opvolgende elemente.	2.4.1.1.2
enkeldeurgang-multiverbinding	"single-sweep multi-join"	MySQL lees een ry van die eerste tabel, en vind 'n ooreenstemmende ry in die tweede tabel, en daarna 'n ooreenstemmende ry in die volgende (derde) tabel totdat al die tabelle deursoek is vir die ry. Die geselekteerde velde word van die ry vertoon indien die ry ooreenstemmende rye vind. Die tabellys word hierna van agter na voor deursoek, totdat 'n tabel (*T*) verkry word wat nog ooreenkomstige rye bevat. Die volgende ry van hierdie tabel *T* word gelees en die proses herhaal na die volgende tabel. As al die ooreenkomstige rye van die ry in die eerste tabel gevind is, word die volgende ry in die eerste tabel verwerk.	3.3.2
gebondelde	"clustered"	Die orde van die datarekords in 'n tabel en die orde van die data-elemente in 'n indeks stem ooreen vir 'n gebondelde indeks.	2.4.1
gebondelde indeksdeursoek	"clustered index scan"	Die gebondelde indeks word van voor tot agter deursoek.	3.2.8.2
geheue-areas	"memory pools"	Spasie wat in die geheue geallokeer is.	3.2.4.1
gekorreleerde subnavraag	"correlated subquery"	Die tipe navraag is 'n subnavraag wat 'n verwysing na 'n tabel of waarde in 'n tabel bevat wat in die navraag, buite die subnavraag, voorkom.	6.3.5
gelyklopendheid	"concurrency"	Verskeie kliënte van 'n databasisbestuurstelsel wat gelyk data gebruik.	4.5
geneste herhaalverbindings	"nested loop joins"	Hierdie soort verbindings maak gebruik van geneste iterasies ('n stel herhalings) om deur die data van een tabel te gaan en vir elke ry dan die rekords in die ander tabel te verkry wat ooreenstem met die verbindingskriteria.	3.2.7.3.3.1
gigagrepe	"gigabyte"	Een gigagreep is gelyk aan 1024 * 1024 * 1024 grepe.	3.3.2
gleuf	"bucket"	Elemente word hier gestoor, nadat 'n "hash"-funksie toegepas is om die korrekte gleuf te verkry.	2.4.1.2
groepering	"clustered"	Verskillende bedieners kan in 'n groep geplaas word. Die groepering dui op bedryfstelseldienste ("services") of prosesse wat ononderbroke voortgaan al is een van die bedieners in die groep buite werking.	3.2.2.6
Herhaalbare lees	"repeatable read"	Die vlak werk bo-op die bevestigdelees-vlak deur te verseker dat die data nie verander vir herhaalde navrae wat kort opmekaar volg nie.	3.2.3.6.1
hervertaling	"recompilation"	Die uitvoerplan word weer vertaal.	3.2.7.4
indeksoptimeringsghoeroe	"Index Tuning Wizard"	'n Toepassing wat indekse aanbeveel.	3.2.8
indekssoektog	"index seek"	'n Spesifieke data-element in die indeks word gesoek. Die proses se koste is normaalweg minder as 'n deursoek proses.	3.2.8.2
isolasie	"isolation"	Transaksies word deur isolasievlakke ten opsigte van lees- en skryf-aksies beheer.	3.2.3.6.1
kasberging	"caching"	Die stoor van data in die kasgeheue.	3.2.3.4
kasgeheue	"cache"	Die geheue-area is 'n vinnige stoorarea.	3.2.3.3

kasgeheue-tref-verhouding	"cache-hit ratio"	Die verhouding van lees/skryf-aksies wat plaasvind in die kasgeheue teenoor die logiese lees/skryf-aksies word die kasgeheue-tref verhouding genoem.	3.2.7.3.2
klasbiblioteek	"class library"	Biblioteek of versameling van klasse (data-strukture).	3.3.2
konsekwentheid	"consistency"	Data se integriteit.	3.3.3
kwalifiserende onttrekking	"qualified retrieval"	Die ontrekking van rye wat aan spesifieke kriterium voldoen.	3.2.3.6.2
lees/skryf-aksies	"inputs / outputs (I/Os)"	Die aksies waarvolgens data vanaf 'n hardeskyf na die geheue gelees en geskryf word. Die aksies kan ook vanaf die geheue na die hardeskyf data lees en skryf.	3.2.8.2
lou subketting	"warm sub-chain"	As 'n indeksblok vanaf die tabel gelees word en in die sleutelbuffer geplaas word, word dit op die einde van die lou subketting geplaas.	3.3.5.3
multiweergawe-gelyklopendheidbeheer	"multi-version concurrency control"	Die tegniek laat nie-sluitende lees-aksies toe, terwyl die nodige rekords steeds gesluit word vir skryf-aksies.	3.3.3
onbevestigde lees	"uncommitted read"	Die transaksie kan enige data op die databladsy lees, ongeag of die data bevestig is of nie.	3.2.3.6.1
ontbondelde	"nonclustered"	Die indeks is nie gebondel nie.	2.4.1
ontleed of formateer	"parsed"	Die navraag word ontleed, sodat 'n volgorde-reeksboom geskep kan word.	3.2.7.1
oopbrongemeenskap	"open-source community"	Die gemeenskap ondersteun die onderlinge deel en beskikbaarstelling van kode.	3.3
oordekkingsindeks	"covering index"	'n Indeks wat al die verwysde kolomme van 'n navraag bevat.	3.2.7.3.2
oorgangspunt	"threshold"	Die optimeringsplan word met hierdie oorgangspunt vergelyk om te bepaal of 'n verdere soektog vir die optimale plan deur die optimeerder uitgevoer moet word.	3.2.7.2
oorsprongskikking	"offset array"	Die beginpunt van die data-ry op elke databladsy.	3.2.5.3
opskrif (buffer- of databladsy)	"header"	Stoorarea van 'n data-ry in die buffer- of databladsy.	3.2.4.3
outo-parametrisering	"autoparameterization"	SQL Server skep 'n basiese formaat ("template") vir eenvoudige navrae waar konstantes gesien word as parameters. Opeenvolgende navrae wat dieselfde formaat is, kan dan dieselfde plan volg met net die parameter wat verander.	3.2.7.5
Rondomtalielees	"Merry-Go-Round scans"	Die tegniek fokus op nie-geordende leeswerk. Indien hierdie tegniek nie gebruik word nie, kan dit gebeur dat 'n proses 'n tabel begin lees en wanneer 'n ander proses dieselfde data wil lees, mag die bladsye alreeds uit die bufferpoel wees en moet die bladsye weer vanaf die skyf gelees word. Die tegniek laat die tweede proses toe om by die punt waar die eerste proses tans is, te begin. Beide prosesse lees dieselfde data. Wanneer die eerste proses afgehandel word, kan die tweede proses die eerste gedeelte lees.	3.2.4.6
ruimtelik	"spatial"	Elke ruimtelike data-objek verteenwoordige 'n spesifieke area wat 'n posisie en grens bevat.	2.4.1.1.3
saamgedrukte sleutels	"packed keys"	Sleutels wat verklein of saamgedruk is.	3.3.4
saamgestelde funksie	"aggregate function"	Funksies wat opsommende inligting oor data bepaal.	6.3.5.1.4
saamvoegverbinding	"merge join"	Twee gesorteerde tabelle word volgens 'n bepaalde kriteria saamgevoeg.	3.2.7.3.3.2
skyntabel	"view"	'n Navraag wat vooraf opgestel en gestoor is en as 'n tydelike tabel funksioneer.	Bylaag A (3.1)

sleutelkompressie	"key compression"	Die sleutels word saamgedruk.	2.4.1.1.2
sluitbestuurder	"lock manager"	Beheer die sluit van transaksies.	3.2.3.6.5
spasie-aanpassing veranderings	"extent switches"	Die aantal kere wat 'n bladsy in die deursoek proses op 'n ander spasie-aanpassing voorkom as die vorige bladsy in die deursoek.	3.2.5
spasie-aanpassings	"extent"	Spasie in SQL Server word geallokeer deur eenhede wat spasie-aanpassings ("extents") genoem word.	3.2.5
spook rekords	"ghost records"	'n Verwyderde ry bly op die bladsy met een bis wat verander is in die ry se opskrif ("header") om aan te dui dat dit 'n spookrekord is en die ry-inhoud nie vir verdere gebruik beskikbaar is nie.	3.2.6.6.2
subboom	"subtree"	Gedeelte van 'n boomstruktuur.	3.2.8.2
subnavraag	"subquery"	Navraag wat in die "WHERE"-gedeelte voorkom.	3.3.6.1
tabel-area	"tablespace"	Die tabelle word in verskeie lêers in die InnoDB stoorenjinstruktuur gestoor.	3.3.2
toeganklikheid	"availability"	As een bediener ophou funksioneer, neem 'n ander bediener die oop plek in. Die gebruiker sien geen verskil in funksionaliteit nie.	4.7
uitbreiding	"extension"	Die tipe van 'n lêer wat in die lêernaam gespesifiseer word.	3.3.3.1.1
verborgenheid	"latency"	Data neem langer om oor die netwerk te beweeg.	4.9
verdeel-met-ineensakking-metode	"split with collapse method"	Die metode word gebruik vir die bywerk van 'n unieke indeks wanneer die indekssleutel verander. Die verwyder- en byvoegbewerkings word saamgevoeg in 'n enkele bywerkbewerking.	3.2.3.6.3
versperring	"blocking"	'n Transaksie veroorsaak 'n versperring indien die spesifieke transaksie (eerste transaksie) wag om 'n slot te kry wat deur 'n ander transaksie vasgehou word en die ander transaksie nie 'n slot van die eerste transaksie benodig nie (wat 'n ketting en dooiepunt tot gevolg sal hê).	4.6
verspreiding	"fan-out"	Die aantal kindernodusse (indekselemente) vir elke indeksbladsy.	2.4.1.1.2
vertaling	"compilation"	Elke instruksie van 'n navraag word ontleed en vertaal om 'n volgorde-reeksboom te kry.	3.2.7
volgorde-reeksboom	"sequence tree"	Die resultaat van die ontleding van 'n navraag in SQL Server.	3.2.7.1
volledige buiteverbinding	"full outer join"	Verbinding wat twee tabelle volgens bepaalde kriterium verbind en rekords bevat wat nie aan die kriterium voldoen nie. Die rekords wat nie aan die kriterium voldoen nie word met "NULL"-waardes in die plek van die kolomme in die tabel wat nie ooreenstemmende rekords bevat nie vertoon.	3.2.8.2
voorvoegselkompressie	"prefix compression"	Skakel algemene voorvoegsels in string uit soos byvoorbeeld "http" in 'n webbladsy se adres.	3.3.4
vreemde sleutel	"foreign key"	Die vreemde sleutel is 'n kolom of kolomme wat na die primêre of alternatiewe sleutel van 'n ander tabel verwys en so die twee tabelle koppel.	3.2.2.4
warm subketting	"hot sub-chain"	As 'n datablok in die lou subketting 'n sekere aantal kere gebruik is, word die blok in die warm subketting geplaas	3.3.5.3
werkverrigtingstoets	"benchmark"	Die toetse dui produktiwiteit aan.	4.3
yl	"sparse"	'n YI indeks bevat een data-element vir elke bladsy van rekords in die datalêer.	2.4.1

BYLAAG C

DIE INSTRUKSIES IN SQL SERVER OM DIE EKSPERIMENTELE TABELLE TE SKEP

1. Inleiding

Die bylaag vertoon die bevele wat gebruik is om die tabelle in SQL Server, wat in hoofstuk 5 beskryf is, te skep (sien paragraaf 2 in hierdie bylaag) en met data te vul (sien paragraaf 3 in hierdie bylaag).

2. Die skep van die tabelle

Die lys stel die skep-bevele voor vir die tabelle in SQL Server (MySQL se skep is soortgelyk, behalwe dat die sintaksis verskil):

```
CREATE TABLE [dbo].[Internet] (
        [DateTime] [datetime] NULL ,
        [SourceIP] [varchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Service] [varchar] (4) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Bytes] [bigint] NULL ,
        [UserID] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Resource] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Site] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Elapsed] [int] NULL
) ON [PRIMARY]

CREATE TABLE [dbo].[Persoon] (
        [van] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [voorletters] [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [voorname] [varchar] (45) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [noemnaam] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [titel] [varchar] (4) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [geboortedatum] [char] (8) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [taal] [varchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [aktief] [char] (1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [netwerkid] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
```

```
CREATE TABLE [dbo].[Tabel1] (
        [verbindkolom] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [kolom1] [int] NULL ,
        [kolom2] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]

CREATE TABLE [dbo].[Tabel2] (
        [verbindkolom] [int] NOT NULL ,
        [kolom1] [float] NOT NULL ,
        [kolom2] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]

CREATE TABLE [dbo].[Tabel3] (
        [verbindkolom] [int] NOT NULL ,
        [kolom1] [char] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [kolom2] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
```

3. Die skep van die inhoud vir elke tabel

Die volgende inligting is die bevele wat gebruik is om die willekeurige tabelle met data te vul. Die bevele word nie in besonderhede verduidelik nie, aangesien enige metode gebruik kan word om die willekeurige tabelle te skep. Die spesifieke data wat in die kolomme verkry word, is nie belangrik vir die ondersoek nie. Die twee tabelle *internet* en *persoon* bevat werklike data geneem uit 'n kosteverhalingstelsel vir internetgebruik.

3.1 Die skep van die inhoud van tabel *tabel1*

Die skep-bevele van *tabel1* lyk soos volg:

```
DECLARE @userid varchar(20), @name varchar(20), @counter smallint
DECLARE int_cursor CURSOR FOR
SELECT netwerkid, noemnaam FROM persoon
OPEN int_cursor
FETCH NEXT FROM int_cursor
INTO @userid, @name
SET @counter = 1
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
        if (round(rand(),0) > 0) and (@name <> '')
        begin
                insert into tabel1 values (@userid, @counter, @name)
                SET @counter = @counter + 1
        end
        FETCH NEXT FROM int_cursor
        INTO @userid, @name
END
CLOSE int_cursor
DEALLOCATE int_cursor
```

3.2 Die skep van die inhoud van tabel *tabel2*

Die skep-bevele van *tabel2* lyk soos volg:

```
DECLARE @getal int, @getal2 float, @naam varchar(20), @counter smallint
DECLARE int_cursor CURSOR FOR
SELECT noemnaam FROM persoon
OPEN int_cursor
FETCH NEXT FROM int_cursor
INTO @naam
SET @counter = 1
WHILE @counter < 2001
BEGIN
        if (@naam = '')
                insert into tabel2 values (round(rand()*3000,0), round(rand()*100,2), NULL)
        else
                insert into tabel2 values (round(rand()*3000,0), round(rand()*100,2), @naam)
        SET @counter = @counter + 1
        FETCH NEXT FROM int_cursor
        INTO @naam
END
CLOSE int_cursor
DEALLOCATE int_cursor
```

3.3 Die skep van die inhoud van tabel *tabel3*

Die skep-bevele van *tabel3* lyk soos volg:

```
DECLARE @getal int, @getal2 float, @naam varchar(500), @counter smallint
DECLARE int_cursor CURSOR FOR
SELECT noemnaam + van + voorletters + titel + netwerkid + voorname FROM persoon
OPEN int_cursor
FETCH NEXT FROM int_cursor
INTO @naam
SET @counter = 1
WHILE @counter < 2001
BEGIN
        insert into tabel3 values (round(rand()*3000,0), @naam, @naam)
        SET @counter = @counter + 1
        FETCH NEXT FROM int_cursor
        INTO @naam
END
CLOSE int_cursor
DEALLOCATE int_cursor
```

BIBLIOGRAFIE

ATZENI, P., CERI, S., PARABOSCHI, S. & TORLONE, R. 1999. Database Systems: Concepts, Languages and Architectures. Berkshire: McGraw-Hill. 612 p.

BALLING, D.J. & ZAWODNY, J.. 2004. High Performance MySQL. Sebastopol: Sams Publishing. 294 p.

BLACK, P.E., *ed.* 2005. Binary Search. *Dictionary of Algorithms and Data Structures.* [Web:] http://www.nist.gov/dads/HTML/binarySearch.html
[Datum van gebruik: 17 Oktober 2006].

CHEN, A.N.K. 2006. Robust optimization for performance tuning of modern database systems. *European Journal of Operational Research,* 171(2):412-429.

DATE, C.J. 1995. An introduction to Database Systems. 6th ed. Menlo Park, CA: Addision-Wesley Publishing Company, Inc. 839 p.

DELANEY, K. 2001. Inside Microsoft SQL Server 2000. Washington: Microsoft Press. 1088 p.

DU, T.C. & WOLFE, P.M. 1997. Overview of emerging database architectures. *Computers & Industrial Engineering,* 32(4):811-821.

ELMASRI, R. & NAVATHE S.B. 1994. Fundamentals of database systems. 2nd ed. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc. 873 p.

FELDMAN, Y.A. & REOUVEN J. 2003. A knowledge-based approach for index selection in relational databases. *Expert Systems with Applications,* 25(1):15-37.

HELLERSTEIN, J.M. 1998. Optimization Techniques for Queries with Expensive Methods. *ACM Transactions on Database Systems,* 23(2):113-157.

KIMBALL, R., REEVES, L., ROSS, M. & THORNTHWAITE, W. 1998. The data warehouse lifecycle toolkit. New York: John Wiley & Sons, Inc. 771 p.

KRUCKENBERG, M. & PIPES, J. 2005. Pro MySQL. New York: Apress. 734 p.

LEI, H. & ROSS, K.A. 1999. Faster joins, self-joins and multi-way joins using join indices. *Data & Knowledge Engineering,* 29:179-200.

MCFADDEN, F.R. & HOFFNER, J.A. 1988. Database Management. 2nd ed. Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc. 680 p.

MCGEHEE, B.M. 2005. Use SET STATISTICS IO and SET STATISTICS TIME to Help Tune Your SQL Server Queries. [Web:] http://www.sql-server-performance.com/statistics_io_time.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005a. Replication: Introducing Replication. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/replsql/replintro_5ir2.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005b. Accessing and Changing Relational Data: Cursors. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_con_07_5lpz.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005c. ODBC Start Page. ODBC. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/dasdkodbcoverview.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005d. Transact Sql Reference: SET ARITHABORT. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_set-set_5tys.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005e. Optimize database performance: Logical and Physical Operators. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/optimsql/odp_tun_1_1183.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005f. Optimizing Database Performance: Designing Federated Database Servers for High Availability. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/optimsql/cm_fedserv_8dq1.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005g. Transact Sql Reference: Select. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_sa-ses_9sfo.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005h. Transact Sql Reference: Search Condition. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_sa-ses_154e.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005i. Transact Sql Reference: Insert. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_ia-iz_5cl0.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT CORPORATION. 2005j. Transact Sql Reference: Update. [Web:] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_ua-uz_82n9.asp [Datum van gebruik: 17 Oktober 2006].

MICROSOFT TECHNET. 2005. Unclustered index faster than clustered index?. [Web:] http://www.microsoft.com/technet/community/newsgroups/dgbrowser/en-us/default.mspx?pg=2&guid=&sloc=en-us&dg=microsoft.public.sqlserver.server&fltr=&mid=728ef863-f77d-4e6b-be3c-22aa83f42b84&tid=eb972547-bb0f-42ec-8792-67be19fb0b6a [Datum van gebruik: 17 Oktober 2006].

MYSQL AB. 2005. MySQL 4.1 Reference Manual. [Web:] http://downloads.mysql.com/docs/refman-4.1-en.pdf [Datum van gebruik: 17 Oktober 2006].

NO, J., THAKUR, R. & CHOUDHARY, A. 2003. High-performance scientific data management system. *Journal of parallel and distributed computing,* 63(4):434-447.

OATES, BJ. 2006. Researching Information Systems and Computing. London: SAGE Publications. 341 p.

RAMAKRISHNAN, R. & GERHRKE J. 2003. Database Management Systems. 3rd ed. Singapore: McGraw-Hill Book Co. 1065 p.

ROB, P. & CORONEL, C. 2007. Database Systems: Design, Implementation & Management. 7th ed. Boston, MA: Course Technology. 668 p.

SHASHA, D. & BONNET, P. 2003. Database Tuning: Principles, Experiments, and troubleshooting techniques. San Francisco, CA: Morgan Kaufmann. 415 p.

STIGTING VIR BEMAGTIGING DEUR AFRIKAANS. 2001. Afrikaanse rekenaar- en Internetterme. [Web:]
http://www.afrikaans.com/foldoc_source_01-11-09.txt
[Datum van gebruik 17 Oktober 2006].

SQL SERVER CENTRAL.COM. 2005. Unclustered index faster than clustered index?. [Web:]
http://www.sqlservercentral.com/forums/shwmessage.aspx?forumid=9&messageid=240674
[Datum van gebruik: 17 Oktober 2006].

SQL SERVER PERFORMANCE.COM. 2005a. Tips on Rebuilding SQL Server Indexes. [Web:]
http://www.sql-server-performance.com/rebuilding_indexes.asp
[Datum van gebruik: 17 Oktober 2006].

SQL SERVER PERFORMANCE.COM. 2005b. Unclustered index faster than clustered index?. [Web:]
http://www.sql-server-performance.com/forum/topic.asp?TOPIC_ID=11653
[Datum van gebruik: 17 Oktober 2006].

SQL TEAM.COM. 2005. Unclustered index faster than clustered index?. [Web:]
http://www.sqlteam.com/forums/topic.asp?TOPIC_ID=58385
[Datum van gebruik: 17 Oktober 2006].

TANENBAUM, A.S. & WOODHULL, A.S. 2001. Operating Systems: Design and Implementation. 2nd ed. Upper Saddle River, NJ: Prentice-Hall. 939 p.

TCX DATAKONSULT AB. 2005. TcX AB. [Web:]
http://www.tcx.se
[Datum van gebruik: 17 Oktober 2006].

WHITEHORN, M. & MARKLYN, B. 2003. Inside relational databases. 2nd ed. London: Springer. 345 p.

WITKOWSKI, A., BELLAMKONDA, S., BOZKAYA, T., FOLKERT, N., GUPTA, A., HAYDU, J., SHENG, L. & SUBRAMANIAN, S. 2005. Advanced SQL Modeling in RDBMS. *ACM Transactions on Database Systems,* 30(1):83-121.