



Hajo Schulz

Entscheidende Maßnahme

c't-Datenbank-Contest: Die Auflösung

Welche Datenbanken sind für den Einsatz in Unternehmensanwendungen besonders geeignet? Diese Frage stellten wir vor einiger Zeit unseren Lesern und baten sie, uns die Überlegenheit ihres persönlichen Favoriten mit einem selbst entwickelten Programm zu demonstrieren.

Ähnlich wie bei Betriebssystemen finden auch zu Datenbanken und Programmiersprachen in den Foren von heise online und anderswo Diskussionen um „die Beste“ statt, die häufig mit beinahe religiösem Eifer geführt werden. Statt objektiver Argumente stehen dabei nicht selten persönliche Angriffe im Mittelpunkt. Um diesen Disputen eine sachlichere Grundlage zu geben, haben wir vor einiger Zeit zu einem Datenbank-Contest aufgerufen [1]: Unsere Leser sollten ihre Lieblings-Datenbank und eine Programmiersprache ihrer Wahl hernehmen, um eine vorgegebene Anwendung zu implementieren. Wir versprachen damals, uns diese Lösungen nicht nur im Hinblick auf die Performance anzusehen, sondern auch die Eleganz der jeweiligen Implemen-

tierung zu begutachten und nach besonders pfiffigen Ideen zu suchen.

Diese Beschau hat aus verschiedenen Gründen viel länger gedauert, als wir uns das je vorgestellt hätten. Daher zunächst an alle Teilnehmer und Leser, die gespannt auf das Ergebnis gewartet haben, ein dickes „Sorry“. Und gleich noch eine Hiobsbotschaft hinterher: Der versprochene Test, bei dem Datenbank und Middleware auf zwei verschiedenen Rechnern laufen, musste leider ausfallen, um die Veröffentlichung der Ergebnisse nicht noch länger hinauszuzögern.

Ausschreibung

Die Aufgabe, die wir interessierten Entwicklern gestellt haben, bestand aus einer typischen kleinen E-Commerce-Anwendung.

Per Web-Interface sollen Kunden in einer Datenbank nach DVDs suchen und sie bestellen können. Die Bedienoberfläche war vorgegeben: Sie stammt aus einer von Dell entwickelten, recht spartanisch anmutenden Webanwendung, zu der für verschiedene Datenbanken und Middleware-Plattformen Referenzimplementierungen vorlagen. Außerdem standen ein Generator für Testdaten und ein Client-Simulator zur Verfügung, der die Anwendung mit Anfragen bombardieren und den dabei erreichten Durchsatz messen kann.

Insgesamt erreichten uns 17 Einsendungen, von denen sich 13 in Betrieb nehmen und zumindest so stabil betreiben ließen, dass eine Messung mit dem Testclient möglich war. Die Bandbreite reichte bei den verwendeten Datenbanken von Selbstgestricktem über populäre Vertreter aus der Open-Source-Szene bis hin zu den kostenlos verfügbaren Express-Versionen der Datenbankserver von Oracle und IBM. Bei den eingesetzten Programmiersprachen dominiert

Java, gefolgt von PHP. Je eine Lösung in Python, Perl, Lisp, C++ und einer datenbankeigenen Skriptsprache komplettieren das Teilnehmerfeld. Wer sich einzelne Implementierungen selbst ansehen will, kann sie sich über den Soft-Link am Ende des Artikels herunterladen.

Alle Kandidaten mussten sich auf einem Rechner mit Hyper-Threading-fähigem 3-GHz-P4 und 1 GByte Hauptspeicher unter Windows Server 2003 oder Suse Linux 10.0 bewähren. Der Testclient lief auf einer zweiten, per 100-MBit/s-Netzwerk angeschlossenen Maschine. Gemessen haben wir jede Anwendung mit der von Dell vorgegebenen mittleren Datenmenge: Die Datenbank enthält dabei 2 Millionen Kunden, 100 000 DVDs sowie 1,2 Millionen bereits ausgeführte Bestellungen und kommt auf eine Größe von ungefähr 1 GByte. Das Clientprogramm startete 20 Threads, die nach einer Aufwärmzeit von einer Minute zwei Minuten lang Anfragen an die Anwendung richteten, ohne zwischen den einzelnen Aufrufen eine Pause einzulegen. Wo im Verlauf der Messung noch ein steigender Durchsatz zu erkennen war, erhöhten wir die Aufwärmzeit auf maximal fünf Minuten – einige Anwendungen laufen mit der Zeit immer schneller, weil mit jeder Anfrage mehr Daten im Hauptspeicher des Servers landen und physikalische Datenbankzugriffe seltener werden.

Ausfälle

Vier Einsendungen waren leider nicht für eine Messung zu gebrauchen. So schickte uns Rolf Stakemann ein Framework zum Erzeugen und Pflegen von Datenbanktabellen mithilfe eines dynamisch erzeugten PHP-Frontend. Der Beitrag macht zwar einen sehr interessanten Eindruck, ist aber keine Implementierung des DVD-Shops, sondern eine reine Proof-of-Concept-Anwendung, wie der Autor auch selbst anmerkt.

Den Beitrag von Alvar C. H. Freude mussten wir der Gerechtigkeit halber von der Teilnahme

ausschließen: Einige Konfigurationsdateien und die Dokumentation erreichten uns erst nach dem Einsendeschluss. Die vorgeschlagene Lösung beruht zu einem großen Teil auf den Konfigurationsdateien, mit deren Hilfe man den Apache-Webserver und das Perl-Modul mit speziellen Optionen hätte kompilieren sollen. Die vorgeschlagenen Änderungen an Linux-Kernel-Parametern passten zudem nicht zu der von uns vorgegebenen Hauptspeicherausstattung der Testmaschine.

Bei der Lösung von Tommi Mäkitalo scheiterte die Inbetriebnahme daran, dass der Autor trotz umfangreichem E-Mail-Verkehr nicht in der Lage war, uns die verwendeten Bibliotheken in Versionen zukommen zu lassen, sie sich ohne Konflikte bei den Abhängigkeiten untereinander kompilieren ließen.

Einen recht interessanten Eindruck machte zunächst die Einsendung der Firma tdb Software: Die verwendete tdbengine ist keine SQL-Datenbank, sondern eine Datenablage mit eigener Skriptsprache namens EASY, die nicht nur die Datenverwaltung, sondern auch die HTML-Ausgaben erledigt. Zur Installation kopiert man einfach die Engine samt den verwendeten Skripten ins cgi-bin-Verzeichnis des Webservers. Die Programme muss man dann nur noch in einen speziellen P-Code übersetzen lassen. Bei der Datenverwaltung geht die tdbengine nicht gerade sparsam mit dem Plattenplatz um: Die Testdaten belegen nach dem Import satte 2,27 GByte. Dafür bekommt man aber einen funktionierenden Volltextindex. Bei näherem Hinsehen hat die Anwendung dann leider gehakt: Die Seite zum Anlegen eines neuen Kunden ließ sich nicht anzeigen, und beim Versuch, einen Einkauf durchzuführen, bekam der Testclient hin und wieder eine Antwort, die nicht in sein Schema passte und ihn zum Abbruch bewog. Eine Messung war demzufolge nicht möglich – schade, denn für einen der vorderen Plätze unter den Exoten hätte es sonst sicher gereicht.

Mindestmaß

Unter den funktionierenden Lösungen fand sich überraschenderweise nur eine einzige, die

die Open-Source-Datenbank PostgreSQL verwendet – sie trägt bei der Performance-Messung mit 120 Bestellungen pro Minute (opm) die rote Laterne. Dabei hat sich das Team „bluegap“ redlich bemüht, für Dampf zu sorgen: Der Einsendung lag eine Konfigurationsdatei bei, die der Datenbank einen sinnvollen Umgang mit dem Hauptspeicher beibringen soll, und die Stored Procedure zum Abwickeln einer Bestellung wurde gar in C geschrieben. Zur Ehrenrettung von PostgreSQL sei gesagt, dass die Autoren mit ihrer Implementierung offenbar nicht rechtzeitig fertig geworden sind: Das eingeschickte Paket enthielt auch ein Programmfragment, das die Verbindungen zwischen der Anwendung und der Datenbank in einem Connection Pool verwaltet – eine Maßnahme, die bei anderen Einsendungen für einen deutlichen Performance-Schub gesorgt hat. Leider beherrschte dieses Programm erst die Benutzeranmeldung, aber noch keine Artikelsuche und -bestellung.

Ein sehr sauber implementierter DVD-Shop erreichte uns von Gerhard Kalab: Seine (bis auf den Datenbankservers) komplett in Python geschriebene Lösung verwendet ein Framework namens TurboGears und ist damit gemäß der MVC-Architektur (Model, View, Controller) strukturiert. Die Model-Schicht definiert die Geschäftsobjekte wie Kunde, Artikel oder Bestellung und lädt und speichert sie in der Datenbank. Mit wenigen Änderungen an einigen Konfigurationsdateien lässt sich die verwendete Datenbank austauschen: TurboGears unterstützt neben dem hier verwendeten MySQL noch PostgreSQL, sqlite und Firebird; Anbindungen an Server von Sybase und Oracle sind in Arbeit. Die View-Schicht einer TurboGears-Anwendung besteht aus HTML-Dateien, bei denen einige Tags spezielle Attribute tragen, die ansonsten aber gültigen XHTML-Code enthalten. In puncto Eleganz und Wartbarkeit gehört die vorliegende Lösung in die Spitzengruppe der Wettbewerbsbeiträge.

Die Einsendung der FH Ulm enthielt gleich mehrere Lösungen: Hier hat ein Informatikkurs seinen eigenen Wettbewerb gestartet und uns insgesamt neun Anwendungen von sehr unterschiedlicher Qualität präsentiert.

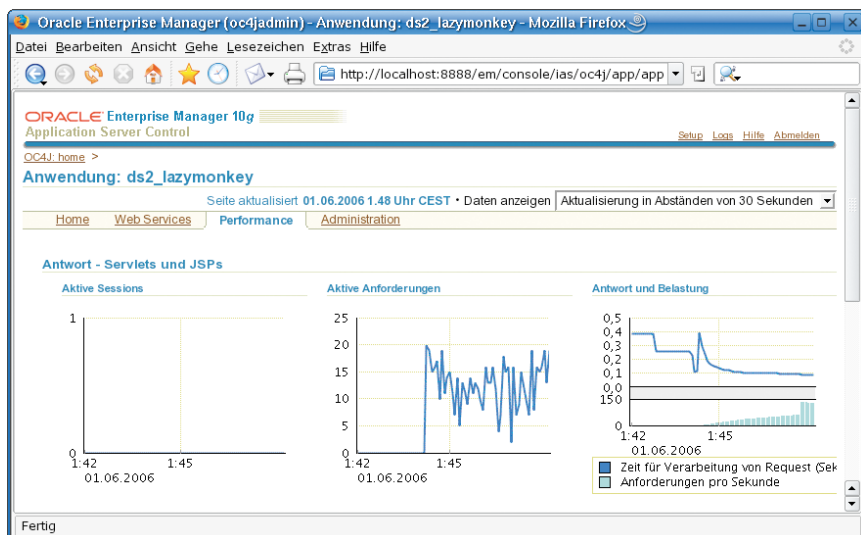
Allen gemeinsam ist, dass sie als Programmiersprache PHP 5 verwenden und an der Datenbankstruktur der Referenzimplementierung keine Änderungen vorgenommen haben. Die in der Tabelle auf Seite 193 aufgeführte Lösung ist die schnellste aus dem Paket, die sich zu Messungen überreden ließ. Sie kommuniziert über die native Schnittstelle mit der MySQL-5-Datenbank. Im Unterschied dazu verwenden die anderen auf MySQL basierenden Programme Frameworks, die sie von der Datenbank unabhängig machen sollen: ADODB, ezSQL, PDO und PEAR::DB kamen dabei zum Einsatz. Leider funktionierten nicht alle Lösungen stabil. Am schnellsten war diejenige mit ADODB; sie schaffte 219 opm gegenüber den 444 opm bei direkter Kommunikation. Dass eine Zwischenschicht zur Datenbankunabhängigkeit Performance kostet, leuchtet ein, erstaunt waren wir aber über das Ausmaß der Einbußen, die sich hier bei ansonsten gleichen Bedingungen besonders eindrucksvoll messen ließen.

Denselben Trend bestätigt der Beitrag von exedio: Die Java-Lösung verwendet das firmeneigene Persistenzframework COPE

zur Datenbankanbindung, das Treiber für MySQL, PostgreSQL, Oracle und die Java-Datenbank HSQLDB mitbringt und unter anderem Sicherheit vor SQL-Injection verspricht. Besonders gelungen fanden wir bei dieser Anwendung die Installation und das Anlegen der Datenbank: Man kopiert die JSP-Dateien und einige Java-Klassen ins webapps-Verzeichnis eines Tomcat-Servers, trägt in eine Konfigurationsdatei die Verbindungsdaten zum Datenbankservers ein und ruft im Browser die COPE-Administrationsseite auf. Das Anlegen der Datenbank samt aller benötigten Tabellen geschieht dann auf Knopfdruck; die nötigen Informationen sucht sich COPE aus den Java-Klassen zusammen. Im Betrieb liefert die Admin-Seite wertvolle Informationen, etwa zur Auslastung von Speicher und Connection Pool.

Die Einsendung von Niklas Mehner fällt zunächst durch ihren ungewöhnlichen Namen auf: „SiebenGeisslein“ ist keine SQL-Datenbank, sondern ein komplett in Java geschriebener Object-Store. Er kümmert sich lediglich um das Speichern der Objekte und die Synchronisierung der Zugriffe. Die gesamte

Das Java-Persistenzframework COPE kommt mit einer eigenen Administrationsseite, die beim Einrichten einer Anwendung hilft und umfangreiche Diagnosefunktionen bietet.



Abfragelogik samt der Verwaltung eines Cache steckt im Client, beim DVD-Shop in einer Tomcat-Anwendung. Im Test erzeugte die erwartungsgemäß eine höhere CPU-Last als die Datenbank selbst; bei viel frequentierten Anwendungen hat dieses Verhalten den Vorteil, dass man sie zunächst auf mehrere Webserver verteilen kann, bevor man über eine datenbankseitige Clusterteilung nachdenken muss. Bis SiebenGeisslein für Anwendungen dieser Größenordnung reif ist, hat der Autor aber noch einiges zu tun – er bezeichnet die eingereichte Version selbst als 0.70pre. Beim Messen mit 20 simulierten Clients konnte die Anwendung nur etwa jede zweite Anfrage beantworten; die anderen beendeten sich mit einem Timeout. Wiederholt man den Test, nachdem die Anwendung einige Zeit gelaufen und der Cache gut gefüllt ist, bessert sich das Verhalten etwas, aber mit 20 Clients ist die vorliegende Version auf der verwendeten Hardware definitiv überfordert. Von daher sind die gemessenen 600 opm nicht ganz mit den übrigen Kandidaten zu vergleichen.

Große Namen

Einer der eher unauffälligen Testkandidaten war die auf Oracle 10g Express basierende Einsendung von Frank Berger. Dass sie auf Anhieb funktionierte, lag sicher nicht zuletzt an der mitgelieferten, redakteurssicheren Schritt-für-Schritt-Anleitung. Im Unterschied zu der Oracle-Referenzimplementierung von Dell läuft das Programm auf dem Anwendungsserver OC4J von Ora-

cle. Es verwendet JavaBeans und Stored Procedures, was der Übersichtlichkeit des JSP-Quelltextes deutlich zugute kommt. In puncto Performance wäre mit dem vorliegenden Ansatz sicher noch etwas mehr möglich gewesen, aber der Autor hat sie bewusst zugunsten der Bedienfreundlichkeit geopfert. So gibt die Anwendung detaillierte Fehlermeldungen bei Falscheingaben und fahndet per Volltextsuche nach DVD-Titeln und Schauspielern.

Von den großen Herstellern kommerzieller Datenbanken hat sich einzig IBM getraut, mit einer eigenen Lösung anzutreten. Sie fällt nicht durch besonders pfiffige Implementierungsdetails auf, sondern eher durch eine äußerst sorgfältige Optimierung. Das beginnt mit dem verwendeten Webserver Apache Tomcat, der einen Connection Pool verwaltet. Die JSP-Seiten setzen Prepared Statements an die Datenbank ab; die eigentliche Bestellung von DVDs wurde zudem mit einer Stored Procedure realisiert. Eines der Skripte zum Aufsetzen der Datenbank schaltet unter anderem den Health Monitor ab und ändert ein paar Parameter, wodurch der DB2-Server besser mit dem Speicher umgeht. Die Datenbank selbst ist über verschiedene Tablespaces verteilt und enthält gegenüber der Referenzimplementierung einige zusätzliche Indizes. Die Struktur der Datenbank wurde sehr stark auf den Benchmark mit dem Testclient hin optimiert. Das geht so weit, dass man der Produkttabelle keine Einträge mehr hinzufügen kann – diese Transaktion ist beim Test zwar

nicht vorgesehen, wäre aber bei einer echten E-Commerce-Anwendung kaum verzichtbar.

Fassade

Der DVD-Shop, den Martin Windy uns geschickt hat, unterschied sich von Dells PHP-4-Referenzimplementierung für MySQL in genau sieben Zeilen Code und zentralisiert dadurch die Angaben Host, Benutzername und Kennwort für den Verbindungsaufbau mit der Datenbank. Als Zusatz hat der Autor noch zwei PHP-Dateien beigelegt, mit denen sich die Datenbank neu erzeugen und mit Testdaten befüllen lassen soll. Funktioniert haben sie nicht.

Auch der Autor hinter dem Pseudonym „USU ValueCoders“ hat sich bei der eigentlichen Datenbank sehr dicht ans Original gehalten. Die JSP-Implementierung hat er aber immerhin um einen Connection Pool und Prepared Statements ergänzt und um einige überflüssige SQL-Abfragen erleichtert.

Dass eine sauber programmierte Zwischenschicht, die die Anwendung von der verwendeten Datenbank unabhängig macht, nicht immer Performance kosten muss, zeigt die von Jan Mönning entwickelte O/R-Bridge. Sie stellt der Anwendung eine objektorientierte Sicht auf die Daten bereit und stützt sich wahlweise auf MySQL, PostgreSQL, Oracle, Microsofts SQL Server oder Access-Dateien. Getestet haben wir nur mit MySQL 5, weil das nach den Erfahrungen des Autors die schnellste Variante ist. Die verwendeten Optimierungen be-

Der Oracle-Anwendungsserver OC4J gibt per Browser Auskunft über seine Auslastung.

schreibt der Autor ausführlich in der beiliegenden Dokumentation, die auch mögliche Fallstricke des Ansatzes nicht verschweigt. Hervorzuheben sind ein globaler Cache für Objekte aus der Datenbank sowie die Möglichkeit, mehrere Schreiboperationen – auch aus verschiedenen Webanfragen – zu einer einzigen Transaktion zusammenzufassen und so den Datenbank-Traffic zu minimieren. Als spezielle Optimierungen für den DVD-Shop hat der Entwickler einen speziellen, aus Hash-Werten bestehenden Index für Benutzernamen, DVD-Titel und Schauspieler angelegt, der eine Volltextsuche erspart. Mit fast 1800 opm stellt diese Einsendung die schnellste datenbankunabhängige Lösung im Testfeld dar.

Aufs Treppchen

Mit der in den Niederlanden entwickelten, quelloffenen MonetDB hat es eine Datenbank auf Platz drei der Performance-Wertung geschafft, die eigentlich gar nicht für Anwendungen wie den DVD-Shop geschaffen wurde. Nach Angaben der Entwickler dient sie ursprünglich für die Analyse umfangreicher Datenbestände und zum Data-Mining. Dass sie in einem transaktionslastigen Benchmark trotzdem so gut aussieht, verdankt sie unter anderem der Tatsache, dass sie sich im laufenden Betrieb selbst optimiert: Indizes werden nicht beim Erstellen einer Datenbank explizit angegeben, sondern der Server merkt selbst, in welchen Spalten er häufig suchen muss und wo sich das Anlegen eines Index lohnen könnte. Der Preis dafür ist allerdings, dass die Datenbank keine referenzielle Integrität kennt. Nach unseren Erfahrungen beim Messen funktioniert die Datenbank am besten, wenn man sie zunächst eine Zeit lang mit typischen Anfragen bombardiert, ihr dann ein wenig Zeit lässt, in der sie sich neu organisiert, und erst dann die eigentliche Messung durchführt. So ist auch der in der Tabelle verzeichnete Wert von 1833 opm zustande gekommen.

Die automatische Indexerstellung gelingt offenbar besonders gut in großen Tabellen, in denen wenige oder keine Schreiboperationen stattfinden. Das nutzen die Entwickler aus und haben sich für den DVD-Shop folgen-

den Trick ausgedacht: Für Benutzer und Bestellungen, also für die Tabellen, in denen der Test viele neue Einträge erzeugt, haben sie je zwei Tabellen angelegt, von denen eine die bereits vorhandenen Daten enthält und nur die andere im Betrieb befüllt wird. Die in der zweiten Tabelle angefallenen Daten müssen dann hin und wieder, etwa einmal pro Nacht per Batch-Job, in die erste übertragen werden. Der schnelle Index auf der deutlich größeren ersten Tabelle macht den Nachteil mehr als wett, dass bei Leseanfragen in zwei Tabellen gesucht werden muss.

Die zweitschnellste Anwendung des Teilnehmerfeldes verwendet keine Datenbank im eigentlichen Sinne: Pico Lisp ist ein von den Einsendern selbst entwickelter und seit einigen Jahren als Open Source gepflegter Lisp-Dialekt, der unter anderem eine Erweiterung enthält, mit der sich Lisp-Symbole automatisch speichern und wiederfinden lassen. Selbst ein eigener Anwendungsserver ist enthalten, sodass man zur Inbetriebnahme der Anwendung keinen externen Webserver benötigt. Die Lektüre des äußerst kompakten Quelltextes ist für Nicht-Lispler gewöhnungsbedürftig, und die verwendeten, oft nur zweibuchstabigen Bezeichner erleichtern die Übersicht nicht gerade. Zu der gemessenen Performance ist anzumerken, dass die Anwendung unter Last immer mal wieder aus dem Tritt kam und einzelne Client-Threads keine Antwort erhielten. Die 2600 opm sind also

mit Vorsicht zu genießen und nicht hundertprozentig mit den anderen Kandidaten vergleichbar.

Die Performance-Krone gebührt eindeutig dem DVD-Shop aus dem Hause MySQL AB. Er zeigt eindrucksvoll, wozu eine sorgfältig konfigurierte MySQL-/PHP-Anwendung unter Zuhilfenahme von ein wenig Zusatzsoftware in der Lage ist. Die durchgeführten Optimierungen betreffen zunächst einmal den Datenbankserver: Er profitiert deutlich, wenn man die Puffergrößen an die verwendete Hardware anpasst und Caches innerhalb der Datenbank abschaltet, die bei der vorliegenden Anwendung keinen Gewinn bringen. Sinnvoller ist der dadurch gewonnene Hauptspeicher im Datenbank-Client, also der PHP-Anwendung genutzt: Hier sorgt das Zusatzmodul APC (Alternative PHP Cache) dafür, dass der PHP-Code nicht bei jeder Anfrage neu übersetzt und anschließend gleich wieder vergessen wird. Wie andere Teilnehmer auch, haben sich MySQL-Entwickler die Datenbankabfragen der Referenzimplementierung vorgenommen, Unnötiges gestrichen und Schreiboperationen zusammengefasst. Statt der mysql-Bibliothek kam die ältere mysql-Extension für die Datenbankbindung zum Einsatz, weil sie persistente Verbindungen beherrscht. Zum Zwischenspeichern bestimmter Abfrageergebnisse haben die MySQL-Entwickler neben dem APC auch noch den memcached von Danga.com eingesetzt.

Den in der Tabelle verzeichneten Performance-Messwert haben wir wie bei allen anderen Kandidaten ermittelt. Allerdings fiel auf, dass die CPU des Servers dabei nur zu etwa 80 % ausgelastet war und die Client-Maschine langsam an die Grenzen ihrer Leistungsfähigkeit kam. Daher hängten wir probierhalber einen dritten Rechner in dasselbe Netz und ließen auf ihm auch noch mal einen Client mit 20 Threads laufen. Durch Addition der beiden Messwerte errechnete sich ein Gesamtdurchsatz von sagenhaften 6000 Bestellungen pro Minute.

Abnahme

Was hat nun der c't Datenbank-Contest an Erkenntnissen gebracht? Zunächst einmal, dass MySQL offenbar mit großem Abstand die bei Entwicklern populärste Datenbank ist: Über die Hälfte der funktionierenden Einsendungen verwenden das Open-Source-Produkt aus Schweden, keine andere Datenbank war überhaupt mehr als einmal erfolgreich vertreten. Wer in erster Linie auf die Performance schaut, liegt mit dieser Wahl richtig, denn mit etwas Sorgfalt bei der Konfiguration und beim Formulieren der Datenbankabfragen sowie mit einigen frei erhältlichen Zusatztools lässt sich aus MySQL mehr als der doppelte Durchsatz der dichtesten Verfolger herauskitzeln. Nachgewiesen ist diese Aussage freilich genau genommen nur für die vorliegende Anwendung, die vermutlich zuerst als MySQL-Programm entstanden ist und schon von daher eine gewisse Affinität zu dieser Datenbank besitzt. Je größer die Unterschiede real existierender Websites zu Dells DVD-Shop werden, desto höher ist auch die Wahrscheinlichkeit, dass sie sich in puncto Geschwindigkeit anders verhalten.

Ähnlich eindeutig geht der Titel der beliebtesten Programmiersprache für Webanwendungen vom Kaliber des DVD-Shops an Java. Allerdings gibt es hier mit PHP einen deutlichen Zweitplatzierten. Für Java spricht im Zweifel die große Auswahl an Bibliotheken, die eine Anwendung nicht nur unabhängig von der Datenbank machen, sondern auch einen sauberen, objektorientierten Datenzugriff ermöglichen.

Und wer ist nun der Sieger im c't Datenbank-Contest? Wie in der Ausschreibung angekündigt, gibt es keinen eindeutigen. Den Titel für die schnellste Anwendung haben sich mit großem Abstand die Entwickler des MySQL-Benchmark-Teams verdient. Die Auszeichnung für die pfiffigste Optimierung innerhalb der Datenbank geht an das MonetDB-Entwicklerteam für die Idee, den Datenzugriff zu beschleunigen, indem man aus einer Tabelle zwei macht und dafür komplexere Anfragen riskiert. Die Frage, ob dieser Trick auch bei anderen Datenbanken funktioniert, harrt allerdings noch der Beantwortung. Die Medaille für das insgesamt originellste und überraschendste Programm darf sich Alexander Burger anheften – wer kommt schon darauf, eine komplette Webanwendung inklusive Datenbank und Anwendungsserver ausgerechnet in Lisp zu programmieren? Eine lobende Erwähnung in dieser Rubrik kassiert – allein schon wegen des Namens der Datenbank – Niklas Mehner, verbunden mit der Hoffnung, dass ihn das anspornt, sein viel versprechendes Konzept SiebenGeisslein weiterzuentwickeln. Den Titel für die insgesamt eleganteste und am besten wartbare Lösung müssen sich Frank Berger und Jan Mönnich teilen, wobei die Lösung von Herrn Mönnich die Nase um einen Sympathiepunkt vorne hat, weil sie mehr eigenes Gehirnschmalz enthält.

Einen Teil der verwendeten Konzepte konnten wir hier nur kurz umreißen, und mancher Teilnehmer hat große Mühe in eine ausführliche Dokumentation seines Beitrags gesteckt. Wer sich näher mit dem einen oder anderen Lösungsansatz beschäftigen oder Implementierungsdetails in eigene Anwendungen übernehmen möchte, sei daher noch einmal auf den Soft-Link zu diesem Artikel hingewiesen, über den alle Einsendungen zum Download bereitstehen. (hos)

Literatur

- [1] Hajo Schulz, Michael Kunze, Gute Nachbarschaft, c't lädt zum Datenbank-Contest ein, c't 20/05, S. 156

Benchmark-Ergebnisse

Einsender	Betriebssystem	Datenbank	Programmiersprache	Messergebnis (opm)
MySQL Benchmark Team	Linux	MySQL 5	PHP 5	3664
Alexander Burger	Linux	Pico Lisp 2.2.1.pre	Pico Lisp 2.2.1.pre	2600 ¹
MonetDB-Entwicklerteam	Linux	MonetDB/SQL 4.9	Java	1833
Jan Mönnich	Windows	MySQL 5	Java	1798
USU ValueCoders	Windows	MySQL 4.1	Java	1564
Martin Winandy	Windows	MySQL 4.1	PHP 5	1542
IBM	Linux	DB2 Express 8.2	Java	1537
Frank Berger	Linux	Oracle 10g Express	Java	1412
Niklas Mehner	Linux	SiebenGeisslein	Java	600 ¹
exedio	Linux	MySQL 5	Java	587
FH Ulm	Windows	MySQL 5	PHP 5	444 ¹
Gerhard Kalab	Windows	MySQL 5	Python 2.4	137
bluegap	Linux	PostgreSQL 8.1	PHP 5	120
tdb Software	Linux	tdbengine	EASY	- ¹
Tommi Mäkitalo	Linux	MySQL 4.1	C++	- ¹
Alvar C.H. Freude	Linux	PostgreSQL 8.x	Perl	- ¹
Rolf Stakemann	Windows	MySQL 3.23	PHP 4	- ¹

¹ siehe Text