Leere Versprechungen?

Optimale Performance mit SQL Server 2000 – Teil 1

von Urs Gehrig

„Microsofts SQL Server 2000 und Performance? Das kann doch wohl nur ein Scherz sein!“ Solche und ähnliche Sprüche höre ich immer wieder. Ist dem wirklich so? Natürlich nicht, ansonsten könnte dieser Artikel ja gleich hier wieder zu Ende sein. In einem Zweiteiler möchte ich Ihnen zeigen, dass MS SQL Server durchaus in den ersten Rängen der Performance-Boliden mithalten kann. Es kommt sogar noch besser: Sie werden Tricks kennen lernen, wie auch Ihre SQL Server-Applikation endlich die Performance bringt, die Sie von einer State-of-the-art-Datenbank erwarten dürfen.

Um es gleich vorweg zu nehmen: Wenn ich hier von Performance spreche, dann meine ich die Leistungsfähigkeit des SQL Servers, mit der dieser T-SQL-Batches abarbeiten kann. Effiziente Backup-, Recovery- oder Replikationsstrategien sind hier genauso wenig ein Thema wie die effiziente Applikationsentwicklung. Hier zählt nur das eine Ziel: die ureigenen Aufgaben einer Datenbank, nämlich das Abfragen und Aktualisieren von Daten, so rasch wie möglich zu erledigen. Im ersten von zwei Teilen werden wir uns auf den Server konzentrieren. Fragen des Clientaufbaus, insbesondere mit ADO.NET, und des Performance-Auditings werden wir uns im zweiten Teil widmen.

Nun aber der Reihe nach. Zuerst wollen wir diejenigen Lügen strafen, die dem SQL Server seine Leistungsfähigkeit mit saloppen Sprüchen immer wieder aberkennen wollen. Bedienen wir uns hierzu doch der aktuellen TPC-C-Testresultate vom 5. Januar 2004 (siehe Tabelle 1). In der Kategorie Non-Clustered Servers steht der SQL Server 2000 an dritter Stelle, hinter zwei Mal Oracle 10g, und in der Kategorie Clustered Servers gar an zweiter Stelle, wiederum hinter Oracle 10g. IBMs DB2 muss sich mit Rängen hinter Oracle und Microsoft begnügen. Weitere Anbieter schaffen es gar nicht, unter die Top Ten zu kommen. Ich denke, diese Resultate sollten die ewigen Zweifler etwas kleinlauter werden lassen. Aber es kommt noch besser: Berücksichtigen wir beim Vergleich lediglich diejenigen Systeme, welche für die Öffentlichkeit zum Zeitpunkt der Abfassung dieses Artikels (Ende Januar) verfügbar sind, dann steht der SQL Server 2000 plötzlich an erster Stelle und zwar in beiden Kategorien. Oracles Testsysteme für die Spitzenreiter beruhen allesamt auf HW- und SW-Komponenten, welche für den allgemeinen Verkauf noch nicht freigegeben sind. Jetzt dürften wir die so saloppen Skeptiker wohl ganz zum Schweigen gebracht haben. SQL Server 2000 führt die Top Ten-Liste gleich zwei Mal an!

Da wir nun wissen, dass der SQL Server es eigentlich kann, wollen wir uns der Herausforderung stellen und die Wege zur Steigerung der Performance suchen und begehen. Die wohl viel versprechendsten Wege finden wir in den Themengebieten Serveraufbau, Abbildung der Applikation auf die Datenbank, Strukturierung des T-SQL-Batches sowie im Aufbau des Clients. Diese Themengebiete wollen wir im Folgenden etwas näher betrachten.

Server Hardware

Der größte Flaschenhals auf Seiten der Hardware ist sicherlich das gesamte I/O-Subsystem, angefangen von der Festplatte über RAM und CPU des Servers und dann weiter über das Netzwerk bis hin zum Client. Es liegt in der Natur der Sache, dass Datenbanksysteme stark datenlastig sind und somit das I/O-Subsystem besonders intensiv nutzen.

Fangen wir also bei der Festplatte an. Wenn diese immer mehr Daten in immer kürzerer Zeit liefern soll, bleiben uns im Wesentlichen zwei Wege offen, welche wir im Idealfall sogar gleichzeitig beschreiten. Erstens, wir erhöhen den Durchsatz der Festplatte, kommen also weg von einer ordinären IDE-Disk und gehen hin zum Nonplusultra, einer UltraWide SCSI 3-

Rank	tpmC	System Availability	Database	Operating System	Cluster
1	1.184893	04/30/04	Oracle Database 10g Enterprise Edition	Red Hat Enterprise Linux AS 3	Y
2	1.008144	04/14/04	Oracle Database 10g Enterprise Edition	HP UX 11.iv2 64-Bit Base OS	N
3	824.164	01/30/04	Oracle Database 10G Enterprise Edition	HP UX 11.iv2 64-Bit Base OS	N
4	786.646	10/23/03	Microsoft SQL Server 2000 Enterprise Ed. 64-Bit	Microsoft Windows Server 2003 Datacenter Edition 64-Bit	N
5	768.839	02/29/04	Oracle Database 10g Enterprise Edition	IBM AIX 5L V5.2	N
6	763.898	11/08/03	IBM DB2 UDB 8.1	IBM AIX 5L V5.2	N
7	709.220	10/15/01	Microsoft SQL Server 2000 Enterprise Edition	Microsoft Windows 2000 Advanced Server	Y
8	707.102	10/23/03	Microsoft SQL Server 2000 Enterprise Ed. 64-Bit	Microsoft Windows Server 2003 Datacenter Edition	N
9	680.613	11/08/03	IBM DB2 UDB 8.1	IBM AIX 5L V5.2	N
10	595.702	12/31/03	Oracle Database 10g Enterprise Edition	Sun Solaris 8	N

Tabelle 1: Top Ten TPC-C vom 5. Januar 2004 [1]

Festplatte. Zweitens, wir vervielfachen den Durchsatz einer Festplatte, indem wir uns gleich mehrerer davon parallel bedienen. Also lieber drei Festplatten mit je 20 GB Speicher als eine Festplatte mit 90 GB. Sind die Daten optimal auf die einzelnen Platten verteilt, kann das System bereits eine weitere Anfrage an eine zweite Festplatte senden, während es noch auf die Antwort der ersten wartet. Achten Sie darauf, dass Sie nie Devices mit verschiedenen Technologien an ein und demselben Disk-Controller anschließen. Also nie eine SCSI 3-Festplatte zusammen mit einem SCSI 1-Bandlaufwerk auf ein und demselben Controller betreiben. Der Controller wird sich immer auf den langsamsten Teilnehmer einspielen und aus Ihrer SCSI 3- wird so auf einmal eine SCSI 1-Platte.

Wenn wir schon mehrere Festplatten nutzen, dann stellen wir uns doch auch gleich der Frage, wie wir diese zusammenschalten sollten und ob wir damit nicht auch gleichzeitig noch ein bisschen mehr Betriebssicherheit erreichen können. Aber sicher. Aber bitte nie mit einem RAID 5-System. In puncto Write-Performance ist nichts so schlecht wie RAID 5, nicht einmal eine einzelne Platte für sich alleine. Wenn schon RAID, dann bitte RAID 10. Weitere Diskussionen über RAID-Systeme finden Sie unter [2]. Und zu guter Letzt: Formatieren Sie Ihre Festplatte immer als NTFS und achten Sie darauf, dass Ihre Platten nie mehr als 80 Prozent voll sind. Überschreiten Sie diese Grenze, wird die Performance wegen immer größerer Fragmentierung der Daten spürbar schlechter.

Sind die Daten erst einmal weg von der Festplatte, landen sie im RAM des Servers, wo sie bereits um ein Vielfaches schneller verarbeitet werden können als noch kurz zuvor auf der Platte. Diesen Performancevorsprung nutzt der SQL Server intensiv, indem er ein ausgeklügeltes System von Buffern unterhält. Daten, welche für das Abarbeiten eines T-SQL-Batches benötigt werden, werden zuerst im Buffer gesucht und erst bei einem Nichtvorhandensein (Mismatch) von der Festplatte gelesen. Je weniger Mismatches auftreten, desto weniger Disk-I/Os müssen unternommen werden und die Daten werden direkt aus dem viel performanteren RAM gelesen. Für uns heißt dies also: SQL Server ist RAM-hungrig – geben wir ihm, was er braucht. Praktisch bedeutet dies: Unter 512 MB RAM läuft fast gar nichts, 400 MHz ist das Minimum und ein großes Swap-File unterstützt das Buffersystem des SQL Servers in keiner Weise. Wenn wir's ganz perfekt machen wollen, dann spendieren wir dem SQL Server eine eigene Servermaschine, damit das RAM auch exklusiv für den SQL Server zur Verfügung steht.

Was für die Beziehung Festplatte/RAM gilt, gilt selbstverständlich auch für die Beziehung RAM/Level 2 Cache des Prozessors. Achten Sie also darauf, dass Sie für Ihr Geld möglichst viel Level 2 Cache erhalten; 512 KB sollten es schon sein. Wenn Sie sich zwischen mehr GHz und neuerer Familie des Prozessors entscheiden müssen, entscheiden sie sich für den neueren Prozessor. Die üblicherweise größeren Level 2 Caches sowie verbesserten Code Prefetching-Algorithmen machen den Rückgang in der Taktfrequenz normalerweise allemal wett. SQL Server ist für Multiprozessor-Systeme wie geschaffen. Es ist für ihn ein Leichtes, einen T-SQL-Batch auf zwei oder mehrere, parallel laufende Prozessoren aufzuteilen und somit rascher zu den Ergebnissen zu kommen.

SQL Server-Versionen

Selbstverständlich haben auch die Versionen des SQL Servers und des Windows-Betriebssystems einen Einfluss auf die Performance, auch wenn dies auf den ersten Blick nicht so scheint. Der Kernel des SQL Servers ist zwar bei allen Versionen derselbe, doch nutzen nicht alle Versionen gleich viel RAM und Prozessoren. Auf der anderen Seite bietet auch Windows nicht in allen Versionen Unterstützung für die selbe Menge von RAM und Prozessoren. Logisch, dass wir von einem Server, der mehr RAM und Prozessoren nutzt, auch die bessere Performance erwarten dürfen. Tabelle 2 zeigt die maximalen Werte für RAM und Prozessoren für die einzelnen Serverkombinationen auf. Wenn Sie die MSDE benutzen, achten Sie auf den von Microsoft speziell eingebauten Workload Governor. Bereits ab einer bescheidenen Last von acht gleichzeitig verarbeitenden T-SQL-Batches tritt dieser in Aktion. Bei jedem logischen Diskzugriff wird ein fixer Wartezyklus eingeschalten. Dies hat zur Folge, dass zwar noch immer alle Anfragen beantwortet werden, aber mit einer sehr schlechten Antwortzeit. Microsoft fordert Sie damit höflich auf, doch endlich von der lizenzfreien SQL Server-Version Abstand zu nehmen und echtes Geld in ihr Produkt zu investieren. Auch wenn Sie in der Regel mit den 32 Prozessoren von Windows 2000 Datacenter auskommen dürften, so hat Windows Server 2003 dennoch einen Vorteil: Gegenüber Windows 2000 weist dieser einen bis zu 35 Prozent höheren Diskdurchsatz bei 30 Prozent geringerer CPU-Auslastung auf. Auch über den TCP/IP-Stack lassen sich bis zu 25 Prozent mehr Daten versenden.

Logisches Datenbankmodell

Da die Hardware nun bereit ist, können wir damit beginnen, uns dem Design der

Was sind TPC-C Tests?

TPC steht für Transaction Processing Performance Council. Dieses ist eine produkt- und lieferantenunabhängige Gemeinschaft, die das Ziel verfolgt, möglichst praxisnahe Performancetests für Datenbanksysteme zu definieren. TPC-C ist ein Online Transaction Processing (OLTP)-Test und simuliert ein komplettes EDV-System, in welchem eine Menge von Benutzern Transaktionen gegen ein Datenbanksystem ausführen. Das Geschäftsmodell orientiert sich dabei an einem Bestell- und Liefersystem mit den Use Cases *neue Bestellung erfassen*, *Warenauslieferung*, *Rechnungserstellung*, *Bestellstatus überwachen* sowie *Lagerüberwachung*. In TPC-C wird der Durchsatz (Transactions per Minute, tmpc) wie folgt definiert: Wie viele neue Bestellungen pro Minute kann das System generieren, wenn gleichzeitig die anderen vier Use Cases angewendet werden. Alle Use Cases müssen dabei eine bestimmte Antwortzeit einhalten, so zum Beispiel fünf Sekunden für die Verarbeitung von neuen Bestellungen. Weiteres hierzu finden Sie unter [1].

TPC definiert lediglich die Tests und nimmt selbst keine vor. Die Tests werden in der Regel in Zusammenarbeit von Datenbanklieferanten und Serverherstellern durchgeführt. TPC übernimmt lediglich die Sammlung, Prüfung und Publikation der Testresultate. Einen Rang unter den zehn Besten, sprich Performantesten, zu erzielen, ist für jeden Lieferanten – sowohl von Datenbank als auch von Serverhardware – ein wichtiges Verkaufsargument. Dies ist offenbar so wichtig, dass in den Lizenzverträgen von Microsoft (wie auch denen von Oracle und IBM) das Veröffentlichen der Resultate eigener Performancetests ohne Zustimmung des Lieferanten ausdrücklich untersagt wird.

Applikation zu widmen. Auf der Ebene der Datenbank sprechen wir hier vom logischen Datenbankmodell. Das logische Datenbankmodell befasst sich mit den Bedürfnissen der Applikation, nicht der Datenbank. Als wohl wichtigstes Artifakt dieses Designprozesses entsteht das Entity Relationship Diagram (ERD), welches die Beziehungen der einzelnen Gruppen von Informationen – oder eben Entitäten – untereinander aufzeigt. Bei der konzeptionellen Gestaltung des ERD wenden wir eine Handvoll wohlbewährter Regeln an – die Datenbank wird normalisiert [3]. Je mehr normalisiert wird, umso mehr Redundanz verschwindet aus der Datenbank. Nun sind aber nicht alle Applikationen von derselben Gestalt. Prinzipiell lassen sich die Applikationen in zwei Gruppen unterteilen: OLTP und OLAP.

Online Transactional Processing (OLTP)-Datenbanken sind die typischen Datenbanken, mit welchen Sie wahrscheinlich am häufigsten zu tun haben werden. Sie wurden dafür entworfen, mit Abfragen wie auch Updates gleichermaßen gut zurecht zu kommen. OLTP-Datenbanken werden in der Regel bis zur dritten Normalform normalisiert. Normalisierung kann die Update-Performance positiv beeinflussen, da ein und dieselben Daten nur einmal geschrieben werden müssen. Normalisieren Sie jedoch nie nur so zum Spaß bis zur fünften Normalform. Je weiter Sie normalisieren, umso aufwändiger werden in der Regel Ihre Abfragen und Ihre Select-Statements überborden mit Table Joins. Behalten Sie folgende Faustregel im Hinterkopf: Ab etwa acht Table Joins pro Select-Statement versagt das komplexe Schaltwerk des Query Optimizers des SQL Servers. Statt den wirklich optimalen Query Plan zu suchen, probiert der SQL Server nach der Brute Force-Methode lediglich ein paar Möglichkeiten aus und arbeitet dann mit einem suboptimalen Query Plan. Das schlägt sich selbstredend in einer drastischen Verschlechterung der Performance nieder.

Online Analytical Processing (OLAP)-Datenbanken sind fast gänzlich denormalisiert. Sie wurden für ein bloßes, effizientes Abfragen der Daten entworfen. Updates werden nur sehr sporadisch und zu unkritischen Tageszeiten mittels eines Batchjobs vorgenommen. Die Datenbank ist so designt, dass im Idealfall zur Beantwortung der häufigsten Abfragen die Daten lediglich von einer Tabelle gelesen werden müssen. Dies führt unweigerlich zu starker Redundanz in der Datenbank: Ein und dieselbe Information ist in mehreren Tabellen abgelegt. Da Updates aber nur selten vorkommen, fällt der Aufwand hierfür kaum ins Gewicht. Data Warehouses sind typische OLAP-Datenbanken.

Physisches Datenbankmodell

Jetzt, wo das optimale ERD für unsere Applikation steht, wenden wir uns dem Entwurf des physischen Datenbankmodells zu. Hier werden die einzelnen Entitäten mit ihren Attributen und Relationen zu Tabellen, Spalten, Schlüsseln, Indizes und vielem mehr. Bezüglich Performance müssen wir der Wahl der Indizes, der Aufteilung der Tabellen auf die einzelnen Festplatten sowie der Verwendung von Stored Procedures die meiste Beachtung schenken. Fangen wir mit den Indizes an.

Ein Index in einer Datenbank hat dieselbe Aufgabe wie das Inhaltsverzeichnis dieser *dot.net magazin*-Ausgabe, nämlich das rasche Auffinden der gesuchten Daten respektive Artikel zu gewährleisten. Genau so, wie ein Buch verschiedene Indizes wie z.B. Inhaltsverzeichnis, Stichwortverzeichnis und Verzeichnis der Tabellen und Bilder haben kann, haben auch Datenbanken mehrere Indizes. Je nach Art der Anfrage wird der Query Optimizer des SQL Servers den geeignetsten Index auswählen. Das Unterhalten der Indizes ist selbst aber auch mit einem bestimmten Aufwand verbunden. Je häufiger und umfangreicher die Daten sich ändern, umso mehr Aufwand bedeutet es für den Server, all die Indizes nachzuführen. OLTP-Datenbanken sind für regelmäßige Updates ausgelegt und daher müssen die Indizes gerade hier besonders sorgfältig gewählt werden. Hier gilt: Nur so viele Indizes anlegen wie unbedingt nötig. Bei OLAP-Datenbanken können praktisch beliebig viele Indizes angelegt werden. Da hier die Daten relativ statisch sind, gibt es praktisch keine Arbeit für deren Unterhalt.

Jede Tabelle hat genau einen Clustered Index, entweder explizit durch uns definiert oder implizit durch den SQL Server ausgewählt. Per Default wird der Clustered Index aus dem Primary Key abgeleitet. Der Clustered Index bestimmt, wie die einzelnen Daten physikalisch geordnet abgelegt werden. Er ist äußerst effizient, da das Finden eines gesuchten Datensatzes innerhalb des Indexes gleichbedeutend mit dem Finden des ganzen Datensatzes ist. Nutzen wir also die Leistungsfähigkeit des Clustered Index und wählen ihn so, dass wir ihn für die häufigsten Anfragen nutzen können. Ganz gleich, ob wir ihn explizit anlegen oder nicht, unterhalten werden muss er so oder so. Im Gegensatz zu diesem einen Clustered Index sind alle anderen Indizes so genannte Nonclustered Indexes. Nonclustered Indexes speichern eine Kopie der indizierten Spalten zusam- men mit einem Pointer zu den eigentlichen Daten, und zwar in sortierter Reihenfolge. Somit können Indizes nicht nur für das Finden von Daten, sondern auch für deren Sortierung genutzt werden. Besteht ein Index aus mehreren Spalten, dann reden wir von einem Compound Index. Es kommt immer wieder vor, dass in einer Anfrage nicht alle Spalten einer Tabelle interessieren. In diesen Fällen kann eine spezielle Form des Compound Index, der Covered Index, eine besonders effiziente Lösung darstellen. Ein Covered Index ist in der Lage, die Abfrage gänzlich allein und ohne Nachschlagen via Pointer im Clustered Index zu erledigen. Dies ist möglich, weil der Covered Index nicht nur alle Spalten des Suchkriteriums enthält, sondern auch gleich noch alle Spalten des gewünschten Resultats.

Klingt kompliziert; ist es aber nicht. Machen wir ein Beispiel und sehen uns hierzu den folgenden TSQL Batch an:

	Enterprise	Developer	Standard	MSDE
W2003 Datacenter	64 / 64 GB	64 / 64 GB	4 / 2 GB	2 / 2 GB
W2003 Enterprise	8 / 32 GB	8 / 32 GB	4 / 2 GB	2 / 2 GB
W2003 Standard	4 / 4 GB	4 / 4 GB	4 / 2 GB	2 / 2 GB
W2000 Datacenter	32 / 64 GB	32 / 64 GB	4 / 2 GB	2 / 2 GB
W2000 Advanced	8 / 8 GB	8 / 8 GB	4 / 2 GB	2 / 2 GB
W2000 Server	4 / 4 GB	4 / 4 GB	4 / 2 GB	2 / 2 GB
W2000 Professional		2 / 2 GB		2 / 2 GB

Tabelle 2: Maximal unterstützte Prozessoren und RAM je Serverkombination

```
CREATE TABLE Angestellter (
   ID          int NOT NULL PRIMARY KEY ,
   Vorname     varchar(50),
   Name        varchar(50),
   Strasse     varchar(50)
)
GO

CREATE INDEX IX_Name ON Angestellter(Vorname, Name)
GO

SELECT Name FROM Angestellter WHERE Vorname = 'Don'
SELECT Vorname FROM Angestellter WHERE Name = 'Miller'
SELECT Name, Strasse FROM Angestellter WHERE Vorname = 'Don'
```

Das erste SQL-Statement kann seine Aufgabe alleine mit der Hilfe des Index *IX_Name* erledigen. Hat der SQL Server im Index einen „Don" gefunden, steht gleich dahinter dessen Name. Ein Nachschlagen im Clustered Index ist nicht nötig. Dagegen hat das zweite SQL-Statement schon etwas mehr Mühe: Es kann den Index *IX_Name* nicht zu Hilfe nehmen, da die Reihenfolge der Spalten im Index nicht stimmt. Ein Compound Index kann nur dann als Covered Index verwendet werden, wenn alle Suchkriterien den Resultatspalten vorangestellt sind. Das dritte SQL-Statement kann hingegen den Index *IX_Name* wieder verwenden, muss aber für die Ermittlung der Straße von Don noch im Clustered Index nachschlagen.

Je mehr Einträge eine Tabelle hat, desto wichtiger ist es, die richtigen Indizes zu haben. Ist die Tabelle klein, lohnt es sich für den SQL Server meist nicht, erst in einem Index zu suchen und anschließend im Clustered Index die fehlenden Informationen nachzuschlagen. In einem solchen Fall kann es effizienter sein, den Clusterd Index linear vom Anfang bis Ende durchzugehen und Eintrag für Eintrag zu überprüfen. Diesen Vorgang nennt man Table Scan. Genauso wichtig ist aber auch die Verteilung der Daten im Index. Heißen 90 Prozent der Angestellten „Don", bringt der Index *IX_Name* für das dritte SQL-Statement unseres Beispiels nichts mehr. Hier kann ein Table Scan wiederum effizienter sein. Wir sehen also, die richtigen Indizes anzulegen, ist nicht immer ganz trivial. Hierzu braucht es viel Wissen über die Art und Weise, wie der Query Optimizer arbeitet. Genauso wichtig ist aber, dass wir wissen, mit welchen Anfragen unsere Datenbank zurecht kommen muss. Und noch eine Sache zum Schluss: Überladen wir die Indizes nie mit zu vielen und unnötigen Spalten. Ein Index sollte immer möglichst kompakt sein. Mit „kompakt" meine ich, dass pro Eintrag in der Tabelle möglichst wenig Bytes für den Index benötigt werden. So braucht der SQL Server zum Einlesen des ganzen Index weniger Plattenzugriffe und die sind ja, wie wir gesehen haben, äußerst teuer.

File Groups

Wenn wir schon bei den Diskzugriffen sind, dann bleiben wir doch noch einen kurzen Moment dabei und schauen uns an, was es mit den Filegroups des SQL Servers auf sich hat. Für jedes Datenbankobjekt (Tabelle, Index etc.) können wir die Filegroup angeben, in der dieses Objekt abgelegt wird. Wenn wir bei der Zuteilung der Datenfiles auf die Filegroups darauf achten, dass Files von ein und derselben Festplatte in nur einer Filegroup erscheinen, können wir einzelne Datenbankobjekte gezielt physikalisch voneinander fernhalten. Was uns das bringt? In puncto Performance eine ganze Menge. Wenn wir in unserer Applikation Tabellen haben, auf welche wir praktisch immer nur sequenziell zugreifen, dann trennen wir diese von den übrigen. Greifen wir auf eine Festplatte streng sequenziell zu, werden wir von dieser mit einem erhöhten Datendurchsatz belohnt. Das kommt daher, dass der Festplattenkopf nicht die ganze Zeit von einem Ort zum andern springen muss. Plattenkopfbewegungen sind besonders zeitintensiv. Protokoll-Tabellen, in welchen einzelne Aktivitäten festgehalten werden, sind typischerweise sequenzielle Tabellen. Trennen wir aber auch Tabellen, welche wir immer wieder in ein und derselben Abfrage gemeinsam beanspruchen. Wir ermöglichen dem SQL Server so, dass er parallel auf die einzelnen Tabellen zugreifen kann und rascher mit der Anfrage fertig wird. Dasselbe gilt auch für die Indizes. Insbesondere gehören die Indizes und die Tabellen nicht auf ein und dieselbe Platte. Damit vermeiden wird, dass beim Nachschlagen der noch fehlenden Informationen im Clustered Index, was die Tabelle ja selber ist, der Diskkopf immer vom Index zur Tabelle und wieder zurück positioniert werden muss.

Stored Procedures?

Kommen wir noch zur Gretchenfrage: Stored Procedures ja oder nein. Aus Sicht der Performanceoptimierung heißt hier die Antwort ganz klar ja. SQL Server kompiliert den ganzen T-SQL-Batch einer Stored Procedure nur ein Mal, und zwar bei dessen erstmaligen Benutzung. Dasselbe gilt auch für die Bestimmung des optimalen Execution Plans durch den Query Optimizer. Zwei Aufgaben, die je nach Abfrage schon mal einen beträchtlichen Teil der gesamten Abarbeitungszeit beanspruchen können. Vermeiden wir aber optionale Argumente, sonst muss der SQL Server den optimalen Execution Plan bei jedem Aufruf erneut bestimmen und der größte Teil des Performancevorsprungs geht wieder verloren. Anstelle einer Stored Procedure mit einem optionalen Argument erstellen wir besser zwei Stored Procedures mit unterschiedlichen Argumenten.

Ausblick

Autsch, fürs erste war das schon recht viel an „Tu dies" und „Tu dies nicht". Ich denke, es ist an der Zeit, hier eine kleine Pause einzulegen. Wir machen weiter im zweiten Teil dieses Artikels in der nächsten Ausgabe. Dort wollen wir uns vor allem mit Aspekten der Erstellung von ADO.NET-Clients beschäftigen. Wie nutze ich ADO .NET am effizientesten? Wo sind die Performancefresser begraben? Wie formuliere ich die T-SQL-Batches am performantesten? Zum Schluss schauen wir uns noch an, welche Mittel wir haben, um vorhandene Flaschenhälse in der Performance zu lokalisieren. Also dann, ich würde mich freuen, wenn Sie das nächste Mal wieder dabei sind. Ich bin es auf alle Fälle. ●

Urs Gehrig ist Head of System Services der bbv Software Services AG. Dieser Service hat sich auf die kundenorientierte Softwareentwicklung mit .NET spezialisiert. Er selber ist seit über 15 Jahren als Entwickler von Windows-Applikationen und EDV-Berater tätig. Seine besonderen Steckenpferd sind Datenbanken und Security. Sie erreichen ihn unter urs.gehrig@bbv.ch.

● Links & Literatur

[1] Homepage des Transaction Processing Performance Council mit den aktuellen Performance Hitlisten: www.tpc.org/

[2] Diskussion der verschiedenen RAID-Konzepte: graphics.adaptec.com/pdfs/ACSP_RAID_Ch4.pdf

[3] Why you need Database Normalization?: www.sqlmag.com/Articles/Index.cfm?ArticleID=4887&pg=1