

# Der Entwickler 03/1999

## Doping Performance bei SQL-Servern optimieren

von Andreas Tengicki und Henry Wolf

SQL-Server müssen nicht langsamer sein als Desktopdatenbanken. Bei der Optimierung der Performance von SQL-Servern gilt es zunächst, einige theoretische Aspekte zu bedenken. Daß es darüber hinaus sinnvoll ist, die spezifischen Bedingungen unterschiedlicher Server zu berücksichtigen, zeigt sich am Beispiel der Programmierung des InterBase-Servers auf der einen und des Oracle-Servers auf der anderen Seite.

### 1. Desktop versus SQL-Server

Bei Desktopdatenbanken ist die Art der Programmierung in vielen Fällen eindeutig. Wenn ein Datensatz auf Grund eines Wertes oder Schlüssels gesucht werden soll, wird ein Index über die Tabelle gelegt und der Zugriff erfolgt darüber. Ansonsten beginnt man beim ersten Datensatz mit den Vergleichen und sucht nacheinander bis man alle Sätze gefunden hat. Dabei ist je nach System mehr oder weniger viel Handarbeit erforderlich. Die meisten Desktopdatenbanken unterstützen den Entwickler dabei durch Filter oder ähnliche Mechanismen. Es stehen dann alle Sätze, auf die die Filterbedingung zutrifft, sequentiell zur Verfügung. Der Zugriff geschieht meist lokal auf dem Desktop. Nicht zu viele Datensätze vorausgesetzt, geht alles sehr schnell. Diese Art der Programmierung bzw. Satzauswahl durch den Anwender setzt genaue Kenntnis der Datenstrukturen voraus. Bei Änderungen an den Tabellen ist meist auch eine Änderung der Auswahlmechanismen und damit des Programms notwendig.

Bei der Verwendung von SQL-Servern ist der Ablauf ein anderer. Die Datenbank kennt ihre Struktur. Der Anwender bzw. der Entwickler stellt seine Fragen in der standardisierten Abfragesprache SQL ( Structured Query Language ). Diese Abfrage muß vom SQL-Server erst geparkt, d. h. übersetzt werden, um dann die Ergebnisse aus den dahinter liegenden Tabellen zu extrahieren. Dieses Parsen der Abfrage ist bei den meisten SQL-Servern der zeitkritische Moment. Hier kann viel Zeit verlorengehen. Zudem kann durch eine ungünstige bzw. nicht auf die Bedürfnisse des konkreten Servers zugeschnittene Abfrage zusätzlich eine langsamere Satzauswahl als notwendig erzeugt werden.

Im folgenden sollen sowohl die theoretischen Aspekte, die zu einer Performancesteigerung beitragen können, als auch die praktischen Gesichtspunkte bei der Programmierung von InterBase- und Oracle-Servern vorgestellt werden. Insbesondere beim direkten Vergleich der unter Oracle und InterBase optimalen Abfragen wird festzustellen sein, daß es das alleinseligmachende SQL-Statement nicht gibt. Vielmehr ist die Berücksichtigung der spezifischen Anforderungen eines jeden Servers äußerst sinnvoll.

### 2. Indizes

Bei der Programmierung von Desktopdatenbanken spielen Indizes sichtbar eine große Rolle. Meist muß, um die Daten überhaupt sortieren zu können, ein Index über die zu sortierenden Spalten vorhanden sein. Auch die Auswahl von Sätzen durch Filter wird durch Indizes erleichtert.

Bei SQL-Abfragen sind die Indizes nicht sichtbar. Die Abfrage einer Tabelle mit Index sieht genauso aus wie eine Abfrage ohne. Auch das Sortieren ist ohne Berücksichtigung von Indizes möglich.

```
Select name from Kunde where Plz between  
10000 and 19999 order by Name
```

Natürlich wird diese Abfrage schneller ausgeführt, wenn es einen Index über die Postleitzahlen gibt. Wenn ein zusätzlicher Index ( vielleicht sogar ein aufsteigender ) über "Name" vorhanden ist, geht es natürlich noch schneller. Dies betrifft aber nur die Abfragen. Alle *Einfügen*-, *Ändern*- oder *Löschen*- Operationen benötigen mehr Zeit, da der Index erst gepflegt werden muß.

Bei der Verwendung kann man wenig Fehler machen, da die Verwendung von Indizes keinerlei Auswirkung auf die Syntax der Abfragen und sonstiger Datenbankoperationen hat.

Wenn Sie also Ihr System in einem ersten Schritt beschleunigen wollen, bestimmen Sie, auf welche Tabellenspalten Sie am häufigsten auswählend oder sortierend zugreifen. Legen Sie für diese Spalten einen Index an. Wenn die Performance des Systems sich dadurch nicht beschleunigt oder aber andere Operationen zu langsam werden, entfernen Sie die Indizes wieder.

Sie können die Indizes auch für eine bestimmte Zeit inaktiv schalten, z. B. während eines Datenimports. Dazu dient der SQL-Befehl

Alter Index <IndexName> Inactive

Dann wird der Index nicht nach dem Einfügen eines jeden Datensatzes aktualisiert, sondern erst wenn er wieder aktiviert wird. Diese Aktivierung dauert dann wiederum einen Augenblick länger, weil nun der ganze Index vom SQL-Server überprüft werden muß. Es können aber Probleme auftreten, wenn Sie mit einem solcherart deaktivierten Index die Eindeutigkeit einer Tabellenspalte überwachen. Denn während des Imports wird es keine Fehlermeldung geben, wenn ein doppelter Wert eingefügt wird, denn der Index, der dies feststellt, ist inaktiv. Bei der Aktivierung des Index' wird dann allerdings der Fehler gemeldet und die Operation abgebrochen. Sie müssen dann nachträglich und durch ein geeignetes SQL-Statement selbst überprüfen, welcher Satz betroffen ist. Der Fehler muß dann korrigiert und der Index erneut aktiviert werden.

### 3. Parsen und Optimieren

Die Beantwortung einer SQL-Anfrage durch einen SQL-Server teilt sich in mehrere Phasen. Zuerst wird das SQL-Statement analysiert und festgestellt, welche Daten aus welchen Tabellen geholt werden sollen. Dann versucht der SQL-Server durch Optimierung einen möglichst schnellen Zugriffspfad auf die Daten zu realisieren.

Wenn mehrere Tabellen bei der Abfrage angesprochen werden, muß der Server eine schnelle Reihenfolge festlegen. Je nach dem, wie gut diese Optimierung die tatsächlichen Gegebenheiten trifft, kann eine SQL-Abfrage schneller oder langsamer beantwortet werden. Das Ergebnis dieses Optimierungsprozesses kann meist durch eine geeignete Formulierung der Anfrage positiv gestaltet werden.

Diese Analyse- und Optimierungsphase beansprucht den größten Teil der Zeit, die der Server zum Beantworten einer Abfrage braucht. Bei Desktopdatenbanken mit indexsequentiell oder ähnlichem Zugriff entfällt diese Analyse- und Optimierungsphase ganz. Diese Aufgabe wurde zuvor vom Entwickler für jede Abfrage des Programms selbst erledigt.

Um die Abfragezeiten weiter zu verkürzen, ist es daher sinnvoll und notwendig, die Zugriffspfade für komplexe Abfragen in der Datenbank zu hinterlegen, damit diese nicht jedesmal neu erstellt werden müssen.

### 4. Views

Views und Stored Procedures sind das Mittel der Wahl, um Zugriffspfade für Abfragen in der Datenbank zu hinterlegen. Eine View definiert neue Spalten und das SQL-Statement, um diese zu bestimmen. Aus Sicht der Anwendung kann eine View wie eine Tabelle behandelt werden. Ist für den SQL-Server eindeutig oder vermittelbar, welche Sätze der Tabellen hinter einer View stehen (Eindeutigkeit), können die Sätze einer View auch bearbeitet werden.

Eine View hat drei wesentliche Vorteile. Erstens können komplexe Sachverhalte vereinfacht dargestellt werden, so daß sich weitere Abfragen leichter formulieren lassen. Zweitens können Benutzerrechte auch auf der Ebene der Views vergeben werden, so daß ein Anwender nur auf die Daten zugreifen kann, die ihm über eine View zur Verfügung gestellt werden. Alle anderen Daten können so vor ihm verborgen werden. Unter dem Aspekt der Performance aber ist am wichtigsten, daß eine View quasi kompiliert in der Datenbank hinterlegt wird, d. h. mit all ihren Zugriffspfaden und optimierten Zugriffsreihenfolgen. Damit vereinfacht sich das Parsen der Anfrage deutlich. Auf diese Weise wird aus der Analyse eines Zugriffs auf mehrere Tabellen bereits der Zugriff auf eine View. Das Ergebnis liegt deutlich schneller vor.

Stored Procedures können je nach SQL-Server auch eine Datenmenge, also mehrere Datensätze, zurückgeben werden. Damit können dann noch komplexere Abfragen als bereits durch Views möglich, kompiliert in der Datenbank hinterlegt werden. Aber gerade bei Stored Procedures muß auf die Eigenheiten eines jeden SQL-Servers besondere Rücksicht genommen werden. Möchte man SQL-Abfragen Server-neutral halten, um sein System auf mehreren Plattformen betreiben zu können, bedeutet dies meistens eine plattformabhängige Implementation der Stored Procedures.

### 5. Theorie versus Performance

In einer gemäß der Theorie entworfenen relationalen Datenbank gibt es keine abhängigen Daten und keine Wiederholungsgruppen. Damit ist die Konsistenz der Datenbank zu jeder Zeit gewährleistet. An zwei Beispielen soll aber verdeutlicht werden, daß bewußte Verstöße gegen die Theorie zu keinen nennenswerten Problemen, aber zu höherer Performance führen können.

Abhängige Daten sind berechnete Felder, deren Daten ebenso gut aus den anderen Daten der Datenbank berechnet werden können. Bei diesen Feldern könnte es vorkommen, daß die Aktualisierung vergessen wird und sie dadurch nicht mehr aktuell, die Datenbank also inkonsistent wäre. Prominentestes Beispiel ist ein Kontostand; dieser stellt eigentlich nur die Summe aller Kontobewegungen dar, ist aber gleichzeitig die zentrale Information zum Beispiel eines Bankkontos. Wenn nun vor jeder Operation wie Kontostandsanzeige, Geldauszahlung etc. erst der Bestand *aller* Buchungen nach denen des anfragenden

Kunden zum Zwecke der Summierung durchsucht werden muß, kann bei mehreren Millionen Bewegungen einer großen Bank davon ausgegangen werden, daß der Server in die Knie geht.

Daher wird man in so einem Fall den aktuellen Kontostand direkt beim Kunden speichern. Die Aktualisierung dieses Feldes muß bei jeder Manipulation der Buchungsdaten geschehen. Bei Desktopdatenbanken geschieht dies dezentral in der Anwendung. Bei SQL-Servern kann diese Aufgabe zentral in der Datenbank durch Trigger geschehen. Dazu implementiert man für die drei möglichen Operationen *Insert*, *Update* und *Delete* je einen Trigger auf der Bewegungstabelle, die dann den Kontostand in der Kundentabelle aktualisieren. Der große Vorteil ist, daß diese Trigger immer greifen, egal aus welcher Anwendung auf die Datenbank zugegriffen wird. D. h. auch bei neuen Anwendungen muß dies nicht erneut implementiert werden. Die Datenkonsistenz ist weiterhin gewährleistet.

Wiederholungsgruppen sind Spalten einer Tabelle, von denen man meint, nicht mehr als  $n$  Stück zu brauchen, also z. B. Unterkostenstellen zu einer Hauptkostenstelle. Aber egal wie viele Unterkostenstellen man implementieren wird, nach Murphy wird die Grenze beim Einsatz des Produktes schnell erreicht werden. Daher werden diese Felder normalerweise herausgetrennt und in einer eigenen Tabelle gespeichert. Um dann alle Unterkostenstellen einer Kostenstelle zu bestimmen, sind zwei Tabellen zu verknüpfen. Dies dauert natürlich länger, als einfach mehrere Felder einer Tabelle abzufragen.

Bei Unterkostenstellen ist es sicherlich noch sinnvoll, diese herauszunormalisieren oder in der Art einer Stückliste zu implementieren, damit auch größere Hierarchien möglich werden. Aber bei anderen Informationen, die man in der Theorie normalisieren könnte, sollte dies aus Gründen der Performance unterbleiben. Als Beispiel sei hier die Telefonnummer erwähnt. Zu jeder Adresse gehören mehrere Telefonnummern, aber deswegen muß nicht normalisiert werden. Sollte tatsächlich jemand mehr als z. B. 7 Telefonnummern haben ( geschäftliche und private Telefonnummer, geschäftliche und private Faxnummer, Handy, Autotelefon, Pager ) und unter diesen nicht erreichbar sein, so will er vielleicht auch nicht erreichbar sein. Daher muß die Anwendung nicht durch die Möglichkeit zum Erfassen unendlich vieler Telefonnummern durch zu viele Tabellen und damit viele weitere Tabellenverknüpfungen ( JOINS ) belastet werden.

## 6. Andere Fallen

Die Erfahrung zeigt, daß ein logisches Oder ( *or* ) in einer where-Klausel eine SQL-Abfrage deutlich verlangsamt. Der Einsatz von Indizes wird dabei anscheinend deutlich komplexer. Umgehen kann man diesen Effekt durch die Verwendung von Unions. D. h. statt:

```
Select Name from Kunde where Ort = 'Berlin' or  
Ort = 'Frankfurt'
```

verwende man

```
Select Name from Kunde where Ort = 'Berlin'  
UNION  
Select Name from Kunde where Ort = 'Frankfurt'
```

Zudem kann Nested SQL die Ursache für langsame Abfragen sein, doch dazu mehr, wenn wir uns mit den konkreten Anforderungen an schnelles SQL für die beiden SQL-Server InterBase und Oracle beschäftigen.

## 7. InterBase

InterBase ist laut den Ergebnissen verschiedener Untersuchungen eine sehr schnelle SQL-Datenbank. Gleichzeitig haben aber einige Entwickler Probleme, performante Anwendungen mit InterBase zu schreiben. Die Beachtung der oben beschriebenen Regeln wird einen ersten deutlichen Erfolg bei der Geschwindigkeit der Anwendung bringen. Zusätzlich sollte man sich über einige weitere Punkte Gedanken machen.

Da wäre zunächst die Reihenfolge der Tabellen in JOIN-Statements und die Reihenfolge von Ausdrücken in verknüpften Auswahlbedingungen zu nennen. Dabei gilt, daß InterBase von vorne nach hinten auswertet. Das bedeutet für where-Bedingungen, daß die einzelnen durch *and* verknüpften Teilbedingungen ihrer Wichtigkeit nach sortiert werden sollen. Der Ausdruck, der am meisten Sätze ausschließt, kommt an erster Stelle, nach der mit der zweitgrößten Ausschlußwirkung, usw. Das bedeutet, daß die Prüfung, ob die hinteren Bedingungen auch gelten, dann jeweils für weniger Sätze erfolgen muß, als wenn die schwächste Auswahlbedingung zuerst kommt.

Analog verhält es sich bei der Verknüpfung von z. B. drei Tabellen. Der folgende Ausdruck ist eine unglückliche Formulierung:

```
SELECT ArtikelNummer, Anzahl, Preis  
FROM Artikel A  
JOIN Bestellung B ON A.ARKey = B.ARKey
```

```
JOIN Kunde K ON B. KNR. = K. KNR
WHERE KNR = 12
```

Zuerst werden hier alle Artikel mit allen Bestellungen und Kunden verknüpft, bevor die Auswahl der den Kunden mit der Nummer 12 betreffenden Bestellungen erfolgen kann. Geht man genau umgekehrt vor, wird zuerst auf die Kundentabelle zugegriffen und der Kunde mit der Nummer 12 ausgewählt werden. Dann erfolgen alle weiteren Operationen auf einer kleineren Anzahl von Sätzen.

## 8. Nested SQL

Zudem ist die Verwendung von Nested SQL in InterBase problematisch, denn die Substatements werden für alle Sätze der Hauptstatements ausgeführt.

Formuliert man also

```
Update Bestellung B set storno = 'Y' where B.Knr =
Select K.Knr from Kunde K where Name = 'Meier'
```

wird für jeden Bestellsatz erneut nach der Kundennummer gesucht, die zum Kunden Meier gehört. Besser ist es da zweistufig vorzugehen, denn dann wird der Befehl zum Bestimmen der Kundennummer nur einmal ausgeführt. Bei vielen Bestellungen also eine sehr hohe Performancesteigerung.

```
Select K.Knr from Kunde K where Name = 'Meier' into : aKnr
Update Bestellung B set storno = 'Y' where B.Knr = : aKnr
```

## 9. Oracle

Oracle als ein komplexes RDBMS bietet vielfältige Möglichkeiten, eine Anwendung entweder schnell und performant zu gestalten oder sie zu einer "lahmen Ente" zu machen. Viele Möglichkeiten ermöglichen auch viele Fehler. Bei Oracle empfiehlt sich prinzipiell ein zweischichtiges Vorgehen:

1. Konfiguration der Datenbank / Anlegen der Datenbankobjekte
2. Gestaltung der Zugriffsmethoden ( SQL ).

## 10. Konfiguration der Datenbank

Der Oracle-Entwickler wird im Normalfall nie mit physikalischen Informationen über die Datenbank konfrontiert. Er legt seine Objekte in logischen Einheiten, den sogenannten Tablespaces ab. Diese Tablespaces werden wiederum physikalisch durch eine oder mehrere physikalische Dateien repräsentiert. Für die Gestaltung gelten im allgemeinen folgende Regeln:

1. Trennung der Tablespaces für Indizes und Tabellen.
2. Ablage der physikalischen Dateien auf unterschiedlichen Festplatten

Die Trennung der Tablespaces für Indizes ermöglicht Oracle einen internen schnelleren Zugriff auf Daten bzw. Indizes. Die Ablage der physikalischen Dateien auf unterschiedlichen Festplatten nutzt insbesondere bei SCSI-basierender Hardware des Servers die Intelligenz der Hardware aus, um einen schnelleren Zugriff auf die Daten zu ermöglichen. Die Kombination von beiden Regeln, einer ausreichenden Verfügbarkeit von Hauptspeicher und der entsprechenden Konfiguration des Oracle-Servers, stellt ein schnelles System zur Verfügung.

Bezüglich der Konfiguration des Oracle-Servers soll insbesondere auf die Parameter *db\_block\_buffers* und *shared\_pool\_size* hingewiesen werden. Der zentrale Schlüssel für eine schnelle Abarbeitung der Aufgaben eines Oracle-Servers stellt die sogenannte SGA ( System Global Area ) dar. Die SGA stellt den Hauptspeicherbereich dar, in dem alle Daten und Informationen, welche für die Abarbeitung von Aufgaben des Oracle-Servers gespeichert und gepuffert werden. Ein großer Bereich ermöglicht einen hohen Grad der Pufferung dieser Informationen und natürlich eine Minimierung der erforderlichen Plattenzugriffe. Mittels des Parameters *shared\_pool\_size* legt man die Größe dieses Hauptspeicherbereiches fest.

Mittels *db\_block\_buffers* wird festgelegt, wie viele Datenblöcke ( 1024, 2048 oder 4096 ) im Hauptspeicher eingelesen und zwischengepuffert werden. Eine hohe Anzahl von im RAM gepufferten Datenblöcken minimiert die Plattenzugriffe.

## 11. Gestaltung der Zugriffsmethoden

Bezüglich der Reihenfolge der Tabellen in JOIN-Statements und der Reihenfolge von Ausdrücken in verknüpften Auswahlbedingungen gilt, daß Oracle diese von hinten abarbeitet. Was bei InterBase falsch wäre, wäre also logischerweise bei Oracle korrekt.

```
SELECT A.Artikelnummer, B.ARTIKELNAME,  
B.Anzahl, A.PreisFROM Artikel A, LAGER B  
WHERE A.ARTIKELNUMMER = B.ARTIKELNUMMER  
AND A.ARTIKELNUMMER = 10;
```

Die Nutzung von Views, also geparsten SQL-Statements, stellt eine weitere Methode dar, um die Abarbeitungsgeschwindigkeit zu optimieren.

Bezüglich Nested SQL muß man sich als Oracle-Entwickler bei einer entsprechenden Hardware des Servers wenig Sorgen machen, da diese im Normalfall auf Grund der Pufferung von Daten im RAM (*db\_Block\_buffers*) schnell abgearbeitet werden. Oracle bietet neben zahlreichen Werkzeugen ( Performance-Pack u.a. ) dem Entwickler die Möglichkeit, von der Abarbeitungszeit der einzelnen Phasen eines SQL-Statements bis hin zu den intern erzeugten SUB-Querys effizient Probleme zu erkennen und zu lokalisieren.

Man kann den gesamten Oracle-Server und die ablaufenden SQL-Statements durch eine einfache Änderung der Parameterdatei überwachen und analysieren (*Parameter SQL\_TRACE = TRUE, timed\_statistics = TRUE* ) oder nur die SQL-Statements der aktuellen Sitzung überwachen und analysieren. Der Start der Analyse für die aktuelle Sitzung erfolgt einfach per SQL-Statement *ALTER SESSION SET SQL\_TRACE = TRUE*. Voraussetzung zur Aufzeichnung der zeitlichen Parameter ist die Einstellung *timed\_statistics = TRUE* in der Server-Parameterdatei.

## 12. Unterschiede zwischen InterBase und Oracle

InterBase ist ein einfach zu pflegender SQL-Server, welcher beim Einhalten bestimmter Regeln bei der Formulierung von SQL-Statements die Schaffung von performanten Anwendungen im Workgroup-Segment gestattet. Besonders der Einbau von Views für SELECT-Statements ist bei InterBase zu empfehlen.

Oracle als ein komplexes System bietet viele Möglichkeiten, auch sehr große Anwendungssysteme performant zu implementieren. Mittels der von Oracle zur Verfügung gestellten Werkzeuge und Analysemöglichkeiten und der Beachtung der Oracle-Regeln zur Gestaltung von SQL-Statements kann man diese komplexe System gut beherrschen.

Schon die unterschiedliche Reihenfolge der Abarbeitung der JOIN-Statements und die Reihenfolge von Ausdrücken in verknüpften Auswahlbedingungen zerstört die Illusion, performante Anwendungen völlig kompatibel gestalten zu können. Besonders bei Oracle muß man mit einem fundierten Wissen die vielfältigen Möglichkeiten zur Performance-Steigerung nutzen. InterBase bietet hier nicht viel, läuft aber auch ohne großartige Konfiguration bis zu bestimmten DB-Größen ohne Probleme. Die eingeschränkten Analyse- und Konfigurationsmöglichkeiten lassen den InterBase-Entwickler bzw. den DBA oft im Dunkeln über die Ursache von Problemen und stellen damit auch eine Limitierung für den Einsatz dar.

Hinsichtlich der Nutzung des vorhandenen Hauptspeichers kann man feststellen, daß InterBase zwar einen geringeren Grundbedarf hat, aber den Hauptspeicher ab einer bestimmten Größe ( ca. 64 MB ) weder strukturiert wie Oracle ( SGA, Blockpuffer, usw. ) noch vollständig nutzen kann. Oracle nutzt den verfügbaren Hauptspeicher mit einer manchmal unverschämten Großzügigkeit. Wenn genug da ist, dann wird er strukturiert voll genutzt und stellt die Basis für eine schnelle Datenbank dar.

InterBase ist damit für kleinere bis mittlere Größen geeignet, während Oracle auch in der Lage ist, hohe Datenmengen bei entsprechender Hardware performant und effizient zu verarbeiten.

## 13. Eine Schlußbemerkung

Die beste Optimierung der SQL-Statements ist natürlich sinnlos, wenn der Rechner auf dem der SQL-Server seinen Dienst tut, unterdimensioniert ist. Bedenken Sie immer, daß die Geschwindigkeit eines Konvois maximal so schnell wie sein langsamstes Mitglied ist. Ein SQL-Server braucht keine schnelle CPU, ein SQL-Server braucht einen großen Hauptspeicher, damit der beim Datenschaukeln nicht auf die ( langsame ) Festplatte ausweichen muß. Jeder noch so langsame Hauptspeicher ist schneller als eine Festplatte!