HANSER

Holger Schmeling

Datenbankentwicklung mit dem Microsoft SQL Server 2005

ISBN-10: 3-446-22532-3
ISBN-13: 978-3-446-22532-9

Leseprobe

Weitere Informationen oder Bestellungen unter
http://www.hanser.de/978-3-446-22532-9
sowie im Buchhandel

Mit den präsentierten Daten ergibt sich diese Zusammenfassung:

- Totale Abfragekosten: 0,008
- Logische E-/A-Operationen: 8

Insgesamt wurden die Kosten für diese Abfrage also gegenüber der ursprünglichen Version um den Faktor 8500 gesenkt. Die erforderlichen Leseoperationen sind in der Endversion ca. 11500 Mal kleiner.

12.7 Weitere Hinweise für Indizes

Im vorangegangenen Abschnitt haben Sie gesehen, wie immens wichtig Indizes für performante Abfragen sind. An dieser Stelle sollen einige weitere Aspekte, Indizes betreffend, untersucht werden.

Zunächst einmal enthalten Indizes redundante Information, die für eine logische Funktionalität der Datenbank nicht erforderlich ist. Alle Abfragen funktionieren im Prinzip auch ohne Indizes, können dann aber eben erheblich langsamer sein. Mit der Speicherung redundanter Information sind stets Probleme verbunden. Indizes bilden hier keine Ausnahme. Zunächst einmal benötigen Indizes auch Speicherplatz auf der Festplatte. Wenn Spalten verändert werden, die in Indizes enthalten sind, so müssen außer den eigentlichen Tabellendaten auch die entsprechenden Indizes verändert werden. Für Aktualisierungsoperationen sind Indizes also in der Regel eher hinderlich. Es gibt Ausnahmen, nämlich dann, wenn ein Index verwendet werden kann, um für eine Aktualisierungs- oder Löschoperation die Suche der Zeilen zu unterstützen. Dies kann zum Beispiel der Fall sein, wenn Sie eine Abfrage der Form:

```
update BestellKopf set Nr = '0815'
  where Nr = '4711'
```

mit einem existierenden Index auf der Spalte „Nr“ ausführen.

Das Aktualisieren von Index-Spalten kann letztlich dazu führen, dass der Indexbaum nicht mehr ausbalanciert ist, also für unterschiedliche Suchpfade sehr unterschiedliche Tiefen aufweist. Dies kann sogar so weit gehen, dass der Index vom Optimierer nicht mehr verwendet wird.

12.7.1 Index-Selektivität

Die Selektivität eines Index ist ein Kriterium, das der Optimierer für die Verwendung eines Index in Erwägung zieht. Nehmen Sie als Beispiel unsere Tabelle „BestellPos“ und den Index auf der Spalte „LiefNr“. In unserem ersten Beispiel des vorigen Abschnitts haben wir für diese Spalte einen Index erzeugt, der in der Beispielabfrage dann auch verwendet wurde. Der Index wurde deswegen hergenommen, weil der Optimierer entschieden hat, dass die Abfrage hiervon profitiert. Wenn die Filterbedingung in der WHERE-Klausel so formuliert wird, dass sehr viele Zeilen zurückgegeben werden, dann kann ein Tabellen-

Scan unter Umständen performanter sein als die Suche über den Indexbaum mit einem anschließenden Lookup. Schauen Sie sich hierzu diese Abfrage an:

```
select ProduktId from BestellPos
  where LiefNr > '2'
```

Die Abfrage gibt insgesamt über 53000 Zeilen zurück. Obwohl ein Index für die Spalte „LiefNr" existiert, wird dieser nicht verwendet. Stattdessen wird ein Tabellen-Scan durchgeführt, wie Sie sich durch die Betrachtung des Ausführungsplanes überzeugen können. Ein Durchsuchen des Indexbaumes mit anschließendem Lookup wäre in diesem Fall einfach teurer gewesen als der Tabellen-Scan.

Das gerade für die Index-Selektivität Gesagte gilt übrigens auch generell für die Anzahl von Zeilen in einer Tabelle. Stellen Sie sich diesen Extremfall vor: eine kleine Tabelle, die insgesamt in eine Datenseite passt. Diese Datenseite muss in jedem Fall gelesen werden, wenn Zeilen bzw. Spalten zurückgegeben werden sollen. Hier hilft ein Index also überhaupt nicht. Egal, was für einen Index Sie für diese Tabelle auch erzeugen, er wird in SELECT-Anweisungen immer ignoriert. Für Aktualisierungsoperationen allerdings gilt dies nicht! Der Index muss natürlich immer mit den Tabellendaten übereinstimmen. Also ist der Index nicht nur nutzlos, sondern sogar hinderlich.

12.7.2 Index-Fragmentierung

Mit der Zeit kann ein Indexbaum entarten, wie bereits etwas weiter oben erläutert. Möglich ist auch, dass der Index im Laufe der Zeit fragmentiert wird, also auf der Festplatte nicht mehr zusammenhängend gespeichert wird. Dies kann natürlich immer dann, wenn die Indexdaten von der Festplatte gelesen werden, erheblichen Einfluss auf die Performanz haben. Daher sollten Sie Ihre Indizes in periodischen Abständen reorganisieren.

Der Grad der Index-Fragmentierung kann durch die dynamische Verwaltungsfunktion `sys.dm_db_index_physical_stats` abgefragt werden. Diese Funktion haben wir bereits etwas weiter oben in diesem Kapitel verwendet, um die Anzahl der Stufen in einem Index abzufragen. Die Funktion gibt auch eine Spalte zurück, die den Grad der Fragmentierung eines Index anzeigt: „avg_fragmentation_in_Percent". Ist dieser Wert zu hoch, dann sollten Sie den Index reorganisieren. Es ist eine gute Idee, eine solche Reorganisation von Zeit zu Zeit durchzuführen. Hierzu existiert die Anweisung ALTER INDEX .. REORGANIZE, die so verwendet wird:

```
alter index ix_BK_Nr on BestellKopf reorganize
```

12.7.3 Fehlende Indizes

Eine interessante Frage ist, wie man mitbekommt, dass durch Indizes eine Verbesserung der Abfrageleistung erzielt werden kann. In unseren obigen Beispielen haben wir dies manuell durch die Untersuchung von Ausführungsplänen erledigt.

Der Abfrageoptimierer von SQL Server verfügt für diesen Zweck über ein fantastisches Feature: Er informiert im Ausführungsplan über fehlende Indizes und sogar über die erwartete Verbesserung der Abfrageleistung bei Verwendung eines entsprechenden Index.

Wir wollen uns dies an einem Beispiel ansehen. Zunächst einmal sollen dafür die existierenden Indizes auf allen Tabellen der Datenbank „AWPerformance“ gelöscht werden. Dies wird durch das folgende Skript erledigt:

```
use AWPerformance
declare @cmd varchar(max)
select @cmd=isnull(@cmd + ';','')
           + 'drop index '+name + ' on ' + object_name(object_id)
  from sys.indexes
 where name like 'ix%'
   and object_name(object_id) in ('BestellPos','BestellKopf')
exec (@cmd)
```

Anschließend soll nun der XML-Ausführungsplan für eine Abfrage erzeugt werden. Die Informationen über fehlende Indizes sind nur im XML-Ausführungsplan enthalten. Im bislang meist verwendeten grafischen Ausführungsplan finden Sie diese Informationen nicht. Hier ist das Skript für dieses Experiment:

```
set showplan_xml on
go
 select Preis * Menge from BestellPos
  where LiefNr = '8E3A-4564-99'
go
set showplan_xml off
go
```

Der erzeugte Ausführungsplan enthält tatsächlich eine Sektion `<MissingIndexes>`, wie das folgende Listing zeigt:

```
<MissingIndexes>
  <MissingIndexGroup Impact="99.9521">
    <MissingIndex Database="[AWPerformance]"
                  Schema="[dbo]" Table="[BestellPos]">
      <ColumnGroup Usage="EQUALITY">
        <Column Name="[LiefNr]" ColumnId="3" />
      </ColumnGroup>
      <ColumnGroup Usage="INCLUDE">
        <Column Name="[Preis]" ColumnId="5" />
        <Column Name="[Menge]" ColumnId="6" />
      </ColumnGroup>
    </MissingIndex>
  </MissingIndexGroup>
</MissingIndexes>
```

Die in dieser Sektion enthaltenen Informationen können leicht verwendet werden, um zu entscheiden, ob ein entsprechender Index erstellt werden sollte. Im präsentierten Beispiel kann diese Frage sicherlich mit einem klaren „Ja“ beantwortet werden, da eine Verbesserung der Abfrageleistung um über 99% prognostiziert wird. Das entsprechende CREATE INDEX-Kommando sieht dann so aus:

```
create nonclustered index IX_BP_LiefNr
 on BestellPos(LiefNr) include Preis,Menge
```

Da SQL Server eine Ablaufhistorie über dynamische Verwaltungssichten bzw. -funktionen zur Verfügung stellt, sind Sie nicht einmal darauf angewiesen, für interessierende Abfra-

gen den XML-Ausführungsplan anzuzeigen, so wie oben geschehen. Es ist tatsächlich möglich, auch **im Nachhinein** abzufragen, in welchen Fällen der Optimierer Indizes vermisst hat. Dadurch können Sie periodisch überprüfen, ob sich das Hinzufügen von Indizes lohnen würde. Dies kann zum Beispiel durch die dynamische Verwaltungssicht `sys.dm_db_missing_index_details` geschehen, die detaillierte Informationen über fehlende Indizes zurückgibt:

```
select * from sys.dm_db_missing_index_details
```

Das Ergebnis sieht zum Beispiel so aus:

	index_handle	database_id	object_id	equality_columns	inequality_columns	included_columns	statement
1	1	6	2073058421	[LiefNr]	NULL	[Preis], [Menge]	[AWPerformance].[dbo].[BestellPos]

Weiter oben haben wir bereits gespeicherte Ausführungspläne verwendet, die im Prozedurcache gespeichert sind. Diese Pläne können über die dynamischen Verwaltungssicht `sys.dm_exec_cached_plans` abgefragt werden. Selbstverständlich enthalten diese Pläne auch die Informationen über fehlende Indizes, also entsprechende `<MissingIndexes>`-Sektionen. Mit den in Kapitel 8 erworbenen Kenntnissen über XQuery können wir die folgende Abfrage formulieren, um Abfragepläne zu erhalten, in denen fehlende Indizes „beanstandet" werden:

```
with MI_Plan(XmlPlan) as
(
 select p.query_plan
   from sys.dm_exec_cached_plans
        cross apply sys.dm_exec_query_plan(plan_handle) as p
  where p.query_plan.exist(
         'declare namespace
          mi="http://schemas.microsoft.com/sqlserver/2004/07/showplan";
            //mi:MissingIndexes')=1
)
select c.query('.') as MissingIndexInfo
  from MI_Plan
  cross apply XmlPlan.nodes(
       'declare namespace
           mi="http://schemas.microsoft.com/sqlserver/2004/07/showplan";
            //mi:MissingIndex') as mi(c)
```

Diese Abfrage gibt als Ergebnis die `<MissingIndex>`-Sektionen zurück, enthält also Informationen über alle vom Optimierer vermissten Indizes.

12.7.4 Nicht verwendete Indizes

Zum Schluss unserer Betrachtungen zur Optimierung durch Indizes soll nun noch eine weitere dynamische Verwaltungssicht erwähnt werden: `sys.dm_db_index_usage_stats`. Diese Sicht gibt Auskunft über die Verwendung von Indizes im Rahmen von Such- und Aktualisierungsoperationen. Insbesondere können Sie diese Sicht verwenden, um herauszufinden, ob Indizes überhaupt verwendet werden. Nicht oder nur für Aktualisierungen verwendete Indizes sind natürlich nicht von Nutzen.

Die folgende Abfrage liefert Daten zu allen Indizes der Datenbank „AWPerformance", die seit dem letzten Start von SQL Server nicht verwendet wurden:

```
select object_name(object_id) as Tabelle
      ,i.name                 as [Index]
  from sys.indexes i
 where i.index_id not in (select s.index_id
                            from sys.dm_db_index_usage_stats s
                           where s.object_id = i.object_id
                             and i.index_id=s.index_id
                             and database_id=db_id('AWPerformance'))
 order by object_name(object_id)
```

Seien Sie aber bitte vorsichtig bei der Auswertung des Ergebnisses. Immerhin ist es möglich, dass Indizes zurückgegeben werden, die sehr wohl Verwendung finden, die eben nur nicht seit dem letzten SQL Server-Start benötigt wurden. Außerdem filtert die obige Abfrage nicht diejenigen Indizes heraus, die lediglich für Aktualisierungen verwendet wurden. Auch diese sind ja nicht erforderlich. Probieren Sie es doch einmal aus, auch diese Indizes durch die Abfrage zurückzugeben. Hierzu müssen Sie die Spalte „update“ der Sicht `sys.dm_db_index_usage_stats` berücksichtigen.
tras/Datenbankmodul-Optimierungsratgeber” des Management Studios.

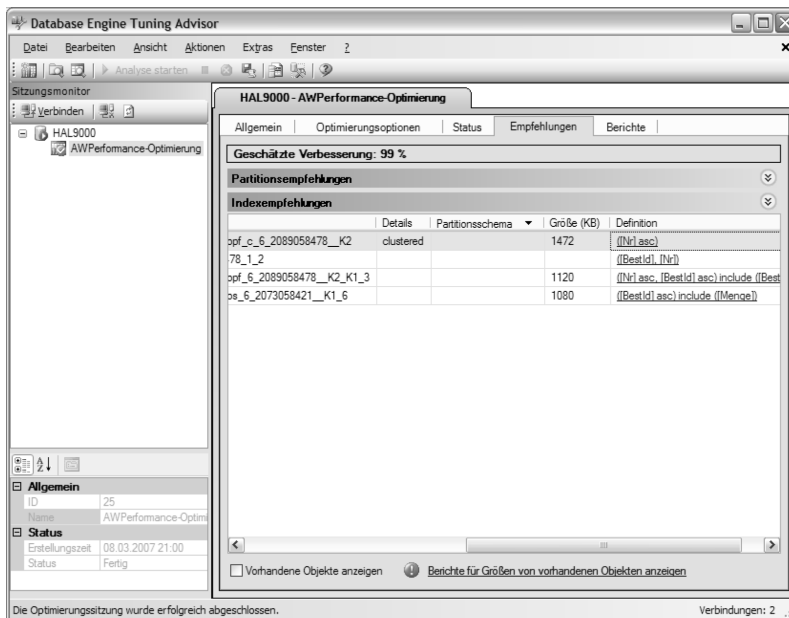

Abbildung 12.26 Ergebnis der Analyse

12.9 Tipps zur Abfrageoptimierung

Zum Abschluss folgen noch einmal eine Zusammenfassung und Ergänzung der Ratschläge für eine Verbesserung der Leistung Ihrer Abfragen.

SQL Server bietet mittlerweile großartige Unterstützung für die Abfrageoptimierung an. Verwenden Sie die vorhandenen Möglichkeiten und Werkzeuge unbedingt, aber verstehen Sie sie als Unterstützung für eine Entscheidungsfindung. Denken Sie bitte daran, dass das menschliche Gehirn durch Software unterstützt, aber nicht ersetzt werden kann. Ansonsten würde ich dieses Buch nicht schreiben, und Sie bräuchten es nicht zu lesen.

Geben Sie in einer SELECT-Anweisung nur die benötigten Spalten zurück

Die Verwendung von „*“ zur Rückgabe aller Spalten ist eine bequeme Möglichkeit, die den Vorteil hat, dass bei Änderungen der Tabellenstruktur die Ausgabe automatisch angepasst wird. Sie erkaufen diesen Vorteil allerdings mit erheblichen Nachteilen:

- Die Benutzung von abdeckenden Indizes ist nicht möglich. Dadurch ist in der Regel immer ein zusätzlicher Lookup erforderlich, der unter Umständen eine entscheidende Verschlechterung des Laufzeitverhaltens zur Folge hat.

- Zur Laufzeit der Abfrage muss zunächst auch eine Abfrage der Metadaten erfolgen, welche die Spalten der Tabellen bzw. Sichten ermittelt. Dadurch wird die Leistung der Abfrage natürlich negativ beeinträchtigt.
- Dir Rückgabe nicht benötigter Daten kann die Netzwerklast enorm erhöhen. Nehmen Sie nur einmal an, dass die folgende Anweisung innerhalb einer gespeicherten Prozedur ausgeführt wird:

  ```
  select * from Rechnung where RgNr = @nr
  ```

  Wenn Sie der Tabelle „Rechnung“ nun eine Spalte vom Typ VARCHAR(MAX) hinzufügen, in der jede Rechnung nochmals im PDF-Format abgelegt wird, so kann die obige Abfrage ganz automatisch um Größenordnungen langsamer sein.

Probieren Sie verschiedene Varianten aus

Experimentieren Sie! Bei komplexen Abfragen gibt es immer unterschiedliche Möglichkeiten, zum Ziel zu kommen. Probieren Sie verschiedene Möglichkeiten aus, und benutzen Sie hierzu auch die Ihnen zur Verfügung stehenden Ressourcen, wie zum Beispiel Ihre Kollegen oder News Groups. Sie haben in diesem Kapitel gesehen, wie Sie die Leistung von Abfragen messen können, und sind dadurch in der Lage, die unterschiedlichen Varianten einer Bewertung zu unterziehen. Nutzen Sie diese Möglichkeit zum Vergleich der gefundenen Lösungsvarianten. Dadurch werden Sie im Laufe der Zeit auch ein Gefühl dafür bekommen, wie der Abfrageoptimierer arbeitet, und werden dadurch in der Lage sein, kompakte und performante Abfragen zu entwerfen.

Vermeiden Sie Hinweise zur Abfrageausführung

Arbeiten Sie nicht gegen den Optimierer, sondern versuchen Sie, ihn zu verstehen und zu benutzen. Abfragehinweise sind in der Regel nicht erforderlich. Wenn Sie wissen, wie der Optimierer arbeitet, müssen Sie ihn nicht austricksen, sondern können Ihre Abfragen entsprechend gestalten.

Verwenden Sie geeignete Indizes

Überprüfen Sie Ihre Indizes dahingehend, ob sie tatsächlich auch verwendet werden, also ob sie erforderlich sind. Nicht benötigte Indizes belasten den Server bei Datenaktualisierungen. Indizes sind für folgende Spalten nützlich:

- **Prädikate in einer WHERE-Klausel.** Wie Sie gesehen haben, gilt dies allerdings nur, wenn der Index auch selektiv genug ist.
- **Fremdschlüssel (FOREIGN KEY).** Für Fremdschlüsselbeziehungen sollten Sie in der Regel einen Index erzeugen. Auch dies gilt allerdings nur dann, wenn dieser Index über eine ausreichende Selektivität verfügt.
- **Abdeckende Indizes.** Prüfen Sie, ob abdeckende Indizes Vorteile bringen. Hierbei leistet der DTA Hilfe.

- **Kurze Indizes.** Verwenden Sie lieber viele Indizes auf wenige Spalten statt weniger Indizes mit vielen Spalten. Der Optimierer hat dadurch mehr Möglichkeiten.

Verwenden Sie aktuelle Statistiken

Gerade bei gespeicherten Prozeduren ist es nicht ausgeschlossen, dass der Plan für eine Prozedur auf veralteten Daten basiert, dass also die Statistiken nicht mehr aktuell sind. Aktuelle Statistiken sind essentiell für eine korrekte Arbeitsweise des Optimierers. Reorganisieren Sie von Zeit zu Zeit Ihre Indexstrukturen, oder verwenden Sie die Anweisung UPDATE STATISTICS bzw. die gespeicherte Systemprozedur `sp_updatestats` zur periodischen Aktualisierung der Statistiken.

Überprüfen Sie Ihr System regelmäßig

Versuchen Sie, eine Performance Baseline zu erstellen, und testen Sie dann in regelmäßigen Abständen, wie der derzeitige Zustand von dieser Basis abweicht. Hierzu benötigen Sie eventuell spezielle T-SQL-Skripte oder auch Ablaufverfolgungen, die Sie von Zeit zu Zeit einfach messen. Für diese Art Messung ist eine Stoppuhr geeignet.

Optimieren Sie die Datenbankstruktur

Der beste Index nützt nichts, wenn Ihre Datenbank so aufgebaut ist, dass relevante Daten jeweils nur durch einen JOIN über acht bis zehn Tabellen zurückgegeben werden können. Sie benötigen also eine entsprechende Tabellenstruktur, damit Abfragen optimal erstellt werden können. Hierzu ist in der Regel ein ausgewogenes Verhältnis zwischen Normalisierung und Denormalisierung einer Datenbank erforderlich. Sie werden also nicht umhinkommen, sich auch mit der Thematik Datenbank-Design auseinanderzusetzen, die nicht Gegenstand dieses Buches ist.

Verwenden Sie passende Datentypen in Tabellen

Denken Sie daran, dass SQL Server Daten seitenweise liest. Je mehr Tabellenzeilen Sie in eine Daten- oder Indexseite aufnehmen können, umso schneller werden Ihre Abfragen sein, da weniger Lesevorgänge benötigt werden. Verwenden Sie also nach Möglichkeit immer den kleinsten passenden Datentyp, und seien Sie vorsichtig mit Zeichenketten fester Länge, also zum Beispiel NCHAR.

Vermeiden Sie temporäre Datenbankobjekte

Temporäre Tabellen oder Cursor verwenden alle die Systemdatenbank **tempdb**. Diese Datenbank müssen sich alle zu einer SQL Server-Instanz existierenden Verbindungen teilen. In der **tempdb** werden auch SQL Server-Objekte abgelegt, die zum Beispiel während einer Abfrage zwischengespeichert werden müssen. Benutzen Sie die **tempdb** so wenig wie möglich.

Vermeiden Sie Cursor

Dieser Tipp wurde bereits in Kapitel 7 genannt. Cursor benötigen eine Menge SQL Server-Ressourcen und sind daher oftmals eine wesentliche Ursache für ein mangelhaftes Laufzeitverhalten.

Wenn Sie mit der Optimierung beginnen, konzentrieren Sie sich bitte zunächst auf die Optimierung von Abfragen. Hier ist immer am meisten herauszuholen. Ich erlebe es häufig, dass ein Optimierungsansatz bei der Hardware, also beim Server beginnt. In der Regel ist ein solcher Ansatz aber lange nicht so vielversprechend wie die eigentliche Abfrageoptimierung. Dies sollte wirklich Ihr Hauptansatzpunkt sein.