



HOLGER SCHMELING

SQL Server 2008 Performanceoptimierung

Das Praxisbuch für Entwickler und Administratoren

3

Ausführung von Abfragen

Nach dem in Kapitel 1 vorgestellten Optimierungsmodell spielt die Optimierung von Abfragen eine entscheidende Rolle bei der Performance-Optimierung. Damit Sie wissen, welche Ansatzpunkte sich hier bieten, ist es wichtig zu verstehen, in welchen Schritten die Verarbeitung von Abfragen durchgeführt wird, das heißt, wie SQL Server also letztlich aus einer SQL-Anweisung ein Abfrageergebnis ermittelt.

In diesem Kapitel werden Sie die an der Verarbeitung von Abfragen beteiligten Komponenten kennenlernen, wobei der Schwerpunkt auf dem Abfrageoptimierer liegt. Ausgangspunkt ist hierbei zunächst die logische Ausführung einer Abfrage, die sich direkt aus der SQL-Anweisung ergibt. Wir werden anschließend untersuchen, wie aus einer SQL-Anweisung in verschiedenen Schritten ein physikalischer Ausführungsplan erstellt wird, der das logisch korrekte Abfrageergebnis auf dem (hoffentlich) günstigsten Weg ermittelt.

3.1 Logische Schritte bei der Abfrageausführung

Wenn Sie eine Abfrage entwerfen und ausführen, haben Sie sich (hoffentlich) zuvor überlegt, wie das Ergebnis aussehen soll und welche Daten aus welchen Tabellen Sie benötigen. Hierbei werden Sie auch bedenken, in welcher Weise Ihre Abfrage ausgeführt wird, wobei Sie hier die logische Reihenfolge der Abarbeitung der einzelnen Klauseln berücksichtigen müssen. Mit anderen Worten: Sie sollten wissen, welche Schritte in welcher Reihenfolge erforderlich sind, um ein korrektes Abfrageergebnis zu erhalten.

Wir wollen hierfür ein Beispiel untersuchen. Betrachten Sie bitte die folgende Abfrage, die alle Produktunterkategorien mit Namen und der Anzahl der in ihr enthaltenen Produkte zurückgibt, wobei nur Unterkategorien berücksichtigt werden, die im Namen die Zeichenkette »bike« enthalten, und die mehr als fünf Produkte umfassen:

```
-- Alle Produktkategorien, die mehr als 5 Produkte enthalten
-- und deren Name die Zeichenfolge "bike" enthält.
use AdventureWorks2008;
select sc.Name, count(*) as Anzahl
  from Production.Product as p
    left outer join Production.ProductSubcategory as sc
      on sc.ProductSubCategoryID = p.ProductSubcategoryID
 where sc.Name like '%bike%'
 group by sc.Name
 having count(*) > 5
 order by Anzahl desc
```

Die einzelnen Bereiche der Abfrage, wie zum Beispiel FROM, WHERE oder ORDER BY werden logisch in einer genau festgelegten Reihenfolge nacheinander abgearbeitet, damit das Abfrageergebnis korrekt ist. Abbildung 3.1 zeigt, in welcher Reihenfolge die einzelnen Schritte der Abfrage ausgeführt werden.

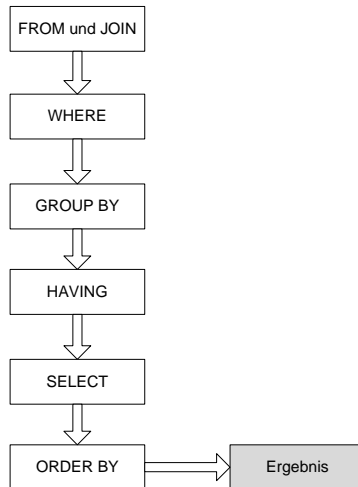


Abbildung 3.1: Logische Abfragereihenfolge

Aus Gründen der Übersichtlichkeit haben wir in unserer Abfrage einige weniger wichtige Klauseln weggelassen (zum Beispiel: DISTINCT, TOP, CUBE und ROLLUP). Es geht hier nur darum, dass Sie sich die folgenden beiden Punkte einprägen:

- ▶ Die logische Ausführungsreihenfolge ist deterministisch. Mit anderen Worten: Wenn Sie eine Abfrage ausführen, so ist immer sichergestellt, dass das Ergebnis der logischen Abarbeitungsreihenfolge entspricht.
- ▶ Die logische Abarbeitungsreihenfolge entspricht nicht der Reihenfolge des Auftretens der Klauseln in einer SELECT-Anweisung. Für SQL-Neulinge ist diese Tatsache stets gewöhnungsbedürftig. Die SELECT-Klausel beispielsweise wird erst beinahe erst zum Schluss ausgewertet, obwohl sie am Beginn einer Abfrage steht.

Sie können sich die Verarbeitung so vorstellen, dass eine virtuelle Tabelle von Schritt zu Schritt »durchgereicht« wird. Jeder Verarbeitungsschritt verändert die virtuelle Tabelle, indem er Zeilen bzw. Spalten hinzufügt oder entfernt und anschließend die modifizierte Tabelle an den nächsten Verarbeitungsschritt übergibt, bis zum Schluss das Ergebnis feststeht.

Aus der Ausführungsreihenfolge lassen sich einige Besonderheiten feststellen: Zunächst einmal werden bei Abfragen, die Gruppierungen verwenden, zwei Zeilenfilter ausgewertet: WHERE und HAVING. Da die WHERE-Klausel deutlich vor der HAVING-Klausel ausgewertet wird, ist es generell günstiger, eine Filterung über die WHERE-Klausel durchzuführen, da die Ergebnismenge in diesem Fall bereits sehr früh eingeschränkt wird und die nachfolgenden Schritte dadurch weniger Zeilen verarbeiten müssen. Hinzu kommt noch, dass für die in der WHERE-Klausel angegebenen Bedingungen Indizes verwendet werden kön-

nen, wodurch die Abfrageleistung erheblich verbessert werden kann. (Hierauf kommen wir in den Kapiteln 5 und 7 noch zurück.) Da die HAVING-Klausel berechnete Werte auswertet, können für diese Art der Filterung keine Indizes verwendet werden. Wann immer möglich, sollten Sie also WHERE verwenden, um die Ergebnismenge bereits vor der Gruppierung einzuschränken.

Ein weiterer Punkt, der vor allem SQL-Neulinge regelmäßig in Erstaunen versetzt, betrifft OUTER JOINS. Es ist so, dass die Auswertung von JOIN-Bedingungen ganz am Anfang der Ausführungsreihenfolge steht. Die WHERE-Klausel wird erst nach dem Ergebnis des OUTER JOIN ausgewertet. Da im Ergebnis eines OUTER JOIN immer alle Zeilen der inneren Tabelle enthalten sind (auch dann, wenn keine äußeren Zeilen gefunden werden; in diesem Fall sind die Spaltenwerte für die äußere Tabelle sämtlich NULL), existiert dadurch beispielsweise auch die Möglichkeit, das Ergebnis in der WHERE-Klausel hinsichtlich NULL-Werten in der äußeren Tabelle einzuschränken. Aus dieser Tatsache ergibt sich die Konsequenz, dass die JOIN-Bedingung in einem INNER JOIN wahlweise über die ON-Klausel oder über die WHERE-Klausel angegeben werden kann. Beide Varianten liefern stets das gleiche Ergebnis. Bei einem OUTER JOIN ist dies nicht so. Hier hat die über die WHERE-Klausel angegebene Bedingung eine andere Bedeutung, als wenn die Bedingung über die ON-Klausel des OUTER JOIN angegeben wird.

Betrachten Sie bitte das folgende Beispiel:

```
-- OUTER JOIN: Die Abfragen liefern verschiedene Ergebnisse!
select p.Name as ProductName
       ,sc.Name as SubcategoryName
from Production.Product as p
     left outer join Production.ProductSubcategory as sc
                on sc.ProductSubcategoryID = p.ProductSubcategoryID
                and sc.Name like '%bike%'
-----
select p.Name as ProductName
       ,sc.Name as SubcategoryName
from Production.Product as p
     left outer join Production.ProductSubcategory as sc
                on sc.ProductSubcategoryID = p.ProductSubcategoryID
where sc.Name like '%bike%'
```

Beide Abfragen verwenden einen OUTER JOIN, wobei in der ersten Abfrage eine Filterung in der ON-Klausel der JOIN-Bedingung und in der zweiten Abfrage eine Filterung in der WHERE-Klausel des SELECT erfolgt. Die beiden Abfragen sind logisch nicht äquivalent! In der ersten Abfrage erfolgt keine Filterung im Sinne einer Einschränkung von Zeilen. Stattdessen bewirkt die Bedingung in der ON-Klausel hier, dass alle Spaltenwerte für die Ausgabespalte SubcategoryName, die nicht die Zeichenfolge »bike« enthalten, NULL sind. In der zweiten Abfrage hingegen wird das Ergebnis über die WHERE-Klausel reduziert. Hier werden tatsächlich alle Zeilen aus dem Ergebnis entfernt, die nicht die Zeichenfolge »bike« in der Spalte SubcategoryName enthalten. Die zuvor über den OUTER JOIN hinzugefügten Zeilen mit NULL-Werten in der Spalte SubcategoryName werden dadurch wieder aus dem Ergebnis gestrichen – das Ergebnis entspricht dadurch dem eines INNER JOIN.

Sie könnten nun vermuten, dass ein OUTER JOIN generell »teurer« ist als ein INNER JOIN, da bei einem OUTER JOIN mehr Zeilen verarbeitet werden müssen. Generell trifft diese Aussage zu. Wann immer möglich, sollten Sie einen OUTER JOIN vermeiden und durch einen günstigeren INNER JOIN ersetzen. Falls Sie dennoch OUTER JOINS verwenden, die durch einen entsprechenden INNER JOIN abgebildet werden können, ist dies in der Regel nicht so dramatisch. Der Optimierer erkennt eine solche Situation und übernimmt die Ersetzung für Sie. Hierauf kommen wir später noch einmal zurück, wenn wir die Arbeitsweise des Optimierers ein wenig genauer untersuchen.

Bei einem INNER JOIN ist es unerheblich, ob Sie die Zeilen in der ON-Klausel der JOIN-Bedingung oder über die WHERE-Klausel herausfiltern. Die folgenden beiden Abfragen sind daher logisch äquivalent:

```
-- INNER JOIN: Beide Abfragen liefern dasselbe Ergebnis.
select p.Name as ProductName
       ,sc.Name as SubcategoryName
       from Production.Product as p
          inner join Production.ProductSubcategory as sc
                 on sc.ProductSubcategoryID = p.ProductSubcategoryID
                 and sc.Name like '%bike%'
-----
select p.Name as ProductName
       ,sc.Name as SubcategoryName
       from Production.Product as p
          inner join Production.ProductSubcategory as sc
                 on sc.ProductSubcategoryID = p.ProductSubcategoryID
       where sc.Name like '%bike%'
```

Kapitel 9 beschäftigt sich ausführlich mit physikalischen JOIN-Operationen.

3.2 Physikalischer Ausführungsplan

Damit der Abfrageprozessor eine Abfrage verarbeiten kann, benötigt er einen entsprechenden physikalischen Ausführungsplan. Dieser Plan wird aus dem Abfragetext in drei Schritten und durch drei unterschiedliche Komponenten erstellt.

3.2.1 Parser

Der erste Schritt bei der Erstellung des Abfrageplans wird – wie bei Übersetzungsvorgängen üblich – durch einen Parser ausgeführt. Hierbei werden unter anderem Syntaxüberprüfungen durchgeführt und Namen (zum Beispiel von Tabellen und Spalten) auf eine korrekte Schreibweise hin überprüft. Der Parser erstellt letztlich einen Baum, der die Ausführungslogik der Abfrage repräsentiert.

3.2.2 Algebrizer

Der vom Parser erzeugte Baum wird vom sogenannten *Algebrizer* im zweiten Schritt weiterverarbeitet. In diesem Schritt werden unter anderem Spalten- und Tabellennamen überprüft und zugeordnet sowie Datentypen auf ihre Korrektheit hin kontrolliert. Der vom Parser erzeugte Baum wird weiter vereinfacht und normalisiert – zum Beispiel durch das Entfernen redundanter Operationen. Das Ergebnis ist am Ende ein vereinfachter Baum, der die Abfrage logisch widerspiegelt. Dieser Baum dient schließlich als Eingabe für den Optimierer.

3.2.3 Optimierer

Der Optimierer ermittelt für eine Abfrage einen physikalischen Ausführungsplan. Hierzu kann der Optimierer aus einer Vielzahl von physikalischen Operatoren auswählen, auf die wir im weiteren Verlauf dieses Buches noch zurückkommen werden. Der Optimierer legt bei der Erstellung des Plans unter anderem fest, welche Indizes verwendet werden, in welcher Reihenfolge auf die beteiligten Tabellen zugegriffen wird, oder wie Verknüpfungen (Joins) physikalisch ausgeführt werden. Sie können sich sicherlich leicht vorstellen, dass – gerade bei komplexeren Abfragen – nicht nur eine Möglichkeit existiert, einen solchen physikalischen Ausführungsplan zu erstellen. Für die meisten Abfragen gibt es mehrere Möglichkeiten; bei komplexen Abfragen können dies sogar einige Millionen sein. Der Optimierer erstellt mehrere (aber in der Regel nicht alle möglichen) Ausführungspläne, die alle gültig sind, also alle zum selben korrekten Ergebnis führen. Unter allen gefundenen Ausführungsplänen wird dann letztlich derjenige mit den geringsten »Kosten« ausgewählt. Dies ist das Prinzip eines kostenbasierten Optimierers. Der einzige Kostenfaktor, der hierbei berücksichtigt wird, ist die Zeit, die benötigt wird, um das Abfrageergebnis zu ermitteln. Etwas weiter unten werden Sie sehen, in welcher Weise Sie sich die Abfragekosten anzeigen lassen können.

Die vom Optimierer ausgegebenen erwarteten Abfragekosten werden allerdings nicht etwa in einer uns bekannten Zeiteinheit, also zum Beispiel in Sekunden, ausgegeben. Stattdessen verwendet der Optimierer ganz offensichtlich eine eigene Uhr, welche die Zeit in einer nur dem Optimierer bekannten Zeiteinheit misst. Verstehen Sie also bitte die ausgegebenen Abfragekosten als eine Art Kennzahl, die einen direkten Bezug zur Ausführungsdauer der Abfrage besitzt (aber keinesfalls linear ist). Generell gilt hier die Aussage, dass niedrigere Kosten für kürzere Ausführungszeiten stehen.



Der Optimierer wird also den Plan mit der vermeintlich geringsten Ausführungszeit auswählen. Hierbei ist es nicht etwa so, dass dieser Plan generell auch den geringsten Ressourcenverbrauch aufweist. Es kann durchaus sein, dass der ausgewählte Ausführungsplan mehr CPU und/oder mehr E/A-Vorgänge benötigt als ein vergleichbarer Plan, der vom Optimierer verworfen wurde. Ausschlaggebend ist allein die geschätzte Ausführungszeit.

Auf Grund der Komplexität ist die Optimierung zugleich der wichtigste und auch komplizierteste Teil bei der Erstellung eines physikalischen Plans für die Abfrageausführung. Hierbei kann die physikalische Ausführungsreihenfolge durchaus von der logischen Reihenfolge abweichen. So ist es zum Beispiel bei einem INNER JOIN unerheblich, in welcher Reihenfolge auf die beteiligten Tabellen zugegriffen wird. Auch die Anwendung zusätzlicher Filterbedingungen kann bei einem INNER JOIN vor oder nach der Verknüpfung erfolgen – das Ergebnis der Abfrage ist in beiden Fällen identisch.

Für die Erstellung des Plans zieht der Optimierer diverse Kriterien in Betracht. Einerseits muss selbstverständlich die logische Ausführungsreihenfolge berücksichtigt werden, also letztlich der Text der SQL-Anweisung. Hinzu kommt, dass der Optimierer auch auf Datenverteilungsstatistiken zurückgreift, um Kardinalitätsschätzungen vorzunehmen. Anhand dieser Schätzungen werden Operatoren für die einzelnen Schritte des Plans ausgewählt. Um zum Beispiel einen JOIN auszuführen, kann der Abfrageprozessor aus diversen physikalischen JOIN-Operatoren auswählen. Die Wahl hängt wesentlich von der Abschätzung der Zeilenanzahl der am JOIN beteiligten Tabellen ab. Anders gesagt: Die durch einen JOIN-Operator erzeugten Kosten werden durch die Anzahl der beteiligten Zeilen bestimmt. In Kapitel 9 gehen wir hierauf genauer ein.

Die Erstellung eines Abfrageplans wird oftmals auch als Kompilieren der Abfrage bezeichnet. Hiermit sind alle Phasen der Planerstellung gemeint, also vom Parsen bis zum Optimieren. Die Optimierung ist hierbei derjenige Schritt, der am kostspieligsten ist. Wie bei jedem Kompilier-Vorgang erzeugt dieser Vorgang vor allem Prozessorlast. Hierbei kann es durchaus vorkommen, dass die Prozessoren zum »Flaschenhals« werden, falls Ihre Anwendungen die Abfragen so stellen, dass viele Übersetzungsvorgänge erforderlich sind.

SQL Server trifft einige Vorkehrungen, um die durch Kompilier-Vorgänge erzeugte Prozessorlast zu minimieren. Hierzu zählt unter anderem das Cachen von kompilierten Abfrageplänen zur späteren Wiederverwendung ohne vorherige (erneute) Übersetzung. Diese Thematik greifen wir ebenfalls in Kapitel 9 auf.

Eine weitere Maßnahme zur Minimierung der Prozessorlast ist die Art und Weise der Erstellung eines Abfrageplans. Wie bereits erwähnt, gibt es bei komplexen Abfragen eine Vielzahl von in Frage kommenden physikalischen Plänen. Dadurch ist es in den meisten Fällen nicht möglich, dass die Kosten aller in Frage kommenden Pläne miteinander verglichen werden, um den günstigsten Plan zu finden. Vielmehr wendet der Optimierer diverse Heuristiken an, die darauf abzielen, dass mit einer möglichst geringen Anzahl erstellter Pläne der optimale Plan gefunden werden kann. Andernfalls wäre die Erstellung eines Abfrageplans in den meisten Fällen teurer als die eigentliche Ausführung der Abfrage. Diese Tatsache sollten Sie sich gut einprägen. In der Regel sind die angewandten Heuristiken vollkommen ausreichend, um einen sehr guten Plan, also einen, der möglichst nahe am theoretisch möglichen Optimum liegt, zu finden. Genau genommen wird jedoch in vielen Fällen nur ein »fast optimaler« Plan erstellt. Hierbei kann es durchaus auch einmal vorkommen, dass ein solcher Plan Kosten verursacht, die deutlich über den des Optimums liegen.

Um einen möglichst kostengünstigen Plan zu finden, durchläuft die Optimierung verschiedene Phasen:

Trivialer Plan

Die Optimierung beginnt mit der Prüfung, ob ein trivialer Plan existiert. Ist die Abfrage einfach genug, so existiert unter Umständen nur ein einziger möglicher Abfrageplan. In diesem Fall ist die Optimierung mit dem Auffinden dieses trivialen Plans abgeschlossen. Betrachten Sie hierzu bitte die folgende Abfrage:

```
select Color
  from Production.Product
```

Hier ist keine Optimierung erforderlich. Es werden einfach alle Color-Spaltenwerte aller Zeilen der Tabelle gelesen. Der Optimierer erkennt, dass es nur einen vernünftigen Plan gibt und verwendet diesen.

Ein weiteres Beispiel für einen trivialen Plan ist die folgende Abfrage:

```
select Color
  from Production.Product
 where Name is null
```

Die in der WHERE-Klausel angegebene Bedingung kann niemals wahr werden, weil durch einen entsprechenden CHECK-Constraint deklariert wurde, dass die Spalte Name nicht NULL werden kann. Folglich wird die Abfrage immer ein leeres Ergebnis zurückliefern. Der Optimierer erkennt dies, weil er auch die Meta-Daten untersucht, und erzeugt einen trivialen Plan, der keinerlei Zugriffe auf Tabellendaten erfordert.

Vereinfachung

Falls kein trivialer Plan existiert, erfolgt im zweiten Schritt der Versuch einer Vereinfachung der Abfrage. Dies geschieht größtenteils durch eine syntaktische Umwandlung oder eine Neuordnung der Operationen. Zum Beispiel erfolgt eine Konvertierung von OUTER JOINS in günstigere INNER JOINS, sofern dies möglich ist.

Schauen Sie sich hierzu bitte die folgende Abfrage an:

```
-- Alle Produktkategorien, die mehr als fünf Produkte enthalten
-- und deren Name die Zeichenfolge "bike" enthält.
select sc.Name, count(*) as Anzahl
  from Production.Product as p
     left outer join Production.ProductSubcategory as sc
       on sc.ProductSubCategoryID = p.ProductSubcategoryID
 where sc.Name like '%bike%'
 group by sc.Name
```

Durch die WHERE-Bedingung werden alle Zeilen, in denen die Spalte sc.Name den Wert NULL enthält, aus dem Ergebnis entfernt. Dies sind aber gerade diejenigen Zeilen, die durch den OUTER JOIN hinzugefügt wurden. Folglich ist der OUTER JOIN in diesem Fall identisch mit einem INNER JOIN; der Optimierer verwendet somit den wesentlich günstigeren INNER JOIN-Operator.

Ein weiteres Beispiel für eine Vereinfachung ist das Anwenden von Filteroperationen zu einem möglichst frühen Zeitpunkt, da die nachfolgenden Schritte dann entsprechend weniger Zeilen verarbeiten müssen.

Auch das Entfernen von unnötigen Tabellen übernimmt der Optimierer bei der Vereinfachung. Sehen Sie sich bitte dazu die folgende Abfrage an:

```
select distinct sc.Name
  from Production.Product as p
       right outer join Production.ProductSubcategory as sc
                   on sc.ProductSubCategoryID = p.ProductSubCategoryID
 where sc.Name like '%bike%'
```

Die Tabelle `Production.Product` ist überflüssig, da von ihr keine Spalten in der Abfrage benötigt werden und sie über den JOIN auch nicht zu einer Einschränkung der Ausgabezeilen führt. Daher nimmt der Optimierer die Tabelle erst gar nicht in seinen Plan auf.

Erstellung mehrerer Pläne und Kostenvergleich

Nach der Vereinfachung erstellt der Optimierer einige Abfragepläne, wobei zunächst nur einfache Optimierungsschritte, wie zum Beispiel das Vertauschen von Tabellenreihenfolgen in JOINS, durchgeführt werden. Sobald hierbei ein Plan gefunden wird, dessen Kosten kleiner als 0,2 sind, ist die Optimierung beendet. Das Ende der Optimierung ist auch dann erreicht, wenn alle möglichen Pläne untersucht wurden. Dies gilt auch für die noch folgenden Schritte. Bei weniger komplexen Abfragen kann dieser Fall durchaus eintreten.

Weitere Versuche mit erhöhter Kostenschwelle

Konnte im ersten Schritt kein Plan gefunden werden, so erfolgt im Anschluss eine erweiterte Optimierung. Hierbei werden zum Beispiel mehr Vertauschungen vorgenommen, als im ersten Schritt. Sobald ein Plan gefunden wird, dessen Kosten kleiner als 1,0 sind, wird dieser Plan ausgewählt, und die Optimierung ist abgeschlossen.

Testen paralleler Ausführungspläne

Bis hierher wurden noch keine Pläne erstellt, die eine parallele Verarbeitung durchführen. Falls der Computer über mehr als eine CPU verfügt und die Kosten für den bislang gefundenen günstigsten Plan höher sind als durch den Konfigurationsparameter `cost threshold for parallelism` angegeben (der Standardwert ist 5), wird die vorherige Optimierungsphase wiederholt. Diesmal ist das Ziel der Optimierung aber die Erstellung eines parallelen Abfrageplanes, also eines Planes, der mehrere Prozessoren verwendet. Anschließend wird für den günstigeren der beiden Pläne (parallel oder nicht parallel) eine weitere volle Optimierungsphase eingeleitet. Hierbei werden dann zum Beispiel auch indizierte Sichten berücksichtigt.

Generell kann der gefundene Ausführungsplan in drei Kategorien eingeteilt werden:

- ▶ Der Plan ist trivial. Eine Optimierung war nicht erforderlich, da es nur einen Plan gibt. Damit ist der Plan zugleich auch optimal.
- ▶ Der Plan ist optimal. Alle möglichen Ausführungsvarianten konnten geprüft werden; der günstigste Plan hat gewonnen.

- ▶ Der Plan ist fast optimal. Für die Überprüfung aller möglichen Ausführungsvarianten ist die Abfrage zu komplex. Aus diesem Grund wurde die Optimierung entsprechend der oben geschilderten Heuristiken verkürzt. Der hierdurch gefundene Plan muss nicht das Optimum sein, er wird jedoch als fast optimal angesehen.

Physikalische Operatoren

Für die Erstellung eines Abfrageplans kann der Optimierer aus über 100 unterschiedlichen physikalischen Operatoren die passenden auswählen. Ich möchte Ihnen an dieser Stelle die reine Auflistung dieser Operatoren ersparen, eine solche Liste finden Sie in der Online-Dokumentation. Wir werden im weiteren Verlauf dieses Buches sehr häufig mit Ausführungsplänen arbeiten und an den entsprechenden Stellen auch etwas zu den verwendeten Operatoren sagen. Haben Sie bitte noch etwas Geduld. – Bereits im folgenden Kapitel werden wir die ersten Abfragepläne untersuchen. An dieser Stelle sollen zunächst nur einige einleitende und allgemeine Kommentare gegeben werden.

Generell verarbeiten alle Operatoren eine gewisse Anzahl von Eingabezeilen und produzieren entsprechende Ausgabezeilen. Die von einem Operator produzierte Ausgabe bildet dann die Eingabe für einen anderen Operator. Hierbei werden prinzipiell zwei Arten von Operatoren unterschieden: Die eine Sorte verarbeitet eintreffende Zeilen sofort und leitet sie an den nächsten Operator weiter. Hierzu zählt zum Beispiel der Filter-Operator, der für jede Zeile unmittelbar ausgewertet wird. Auf der anderen Seite stehen die sogenannten »Stop And Go«-Operatoren, die erst dann eine Ausgabe produzieren können, wenn alle Eingabezeilen vorliegen. Ein Beispiel hierfür ist der Operator zur Sortierung, der natürlich erst dann die Ausgabezeilen erzeugen kann, wenn alle zu sortierenden Zeilen vorliegen.

3.2.4 Anzeigen des Ausführungsplans

SQL Server stellt eine ganze Reihe von Analysemöglichkeiten zur Verfügung, die eine Inspektion der Zustände und einen Einblick die Arbeitsweise ermöglichen. Unter diesen Möglichkeiten sind auch einige, die eine Darstellung bzw. Untersuchung von Abfrageplänen gestatten. Der vom Optimierer erstellte Plan ist also keinesfalls geheim, sondern kann auf unterschiedliche Weise angezeigt werden. Die Ausgabe von Abfrageplänen kann allgemein in zwei Kategorien eingeteilt werden:

- ▶ **Anzeige des geschätzten Ausführungsplanes.** Hierbei wird die eigentliche Abfrage nicht ausgeführt. Sie erhalten nur den Abfrageplan, also das Ergebnis der Optimierung. Diese Möglichkeit ist zum Beispiel dann nützlich, wenn Sie den Plan für eine lang dauernde Abfrage sehen möchten. Sie müssen dann nicht jedesmal auf das Ergebnis der Abfrage warten.
- ▶ **Anzeige des tatsächlichen Ausführungsplans.** Bei Wahl dieser Option wird die Abfrage ausgeführt, und der Abfrageplan wird anschließend angezeigt. Der tatsächliche Ausführungsplan enthält etwas mehr Informationen als der geschätzte Ausführungsplan. Zum Beispiel finden Sie im tatsächlichen Ausführungsplan auch Angaben über die genaue Anzahl verarbeiteter Zeilen – eine Information, die im geschätzten Ausführungsplan nicht enthalten ist. Dass der Plan generell vor der Ausführung einer Abfrage erstellt wird, gilt natürlich gleichermaßen auch für den tatsächlichen Ausführungsplan. Daher ist der Plan, also die verwendeten Operatoren und die Reihenfolge der Abarbeitung, mit dem geschätzten Ausführungsplan identisch.

Anzeige des Ausführungsplans in grafischer Form

Dies ist sicherlich die beliebteste und meistverwendete Möglichkeit. Der Abfrageplan wird als Graph angezeigt, wobei die Knoten des Graphen die Operatoren repräsentieren und die Kanten den Datenfluss veranschaulichen.

Die Anzeige des geschätzten oder tatsächlichen Ausführungsplans können Sie zum Beispiel über die Menüleiste im Management Studio ein- bzw. ausschalten (Abbildung 3.2).

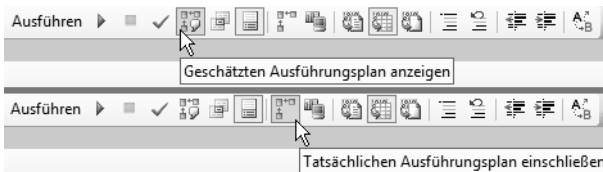


Abbildung 3.2: Anzeigen des grafischen Ausführungsplans

Ist eine Option gewählt, dann erfolgt für jede danach ausgeführte Abfrage in dem geöffneten Fenster die Ausgabe des Abfrageplans in einem separaten Reiter. Abbildung 3.3 zeigt ein Beispiel für einen grafischen Abfrageplan.

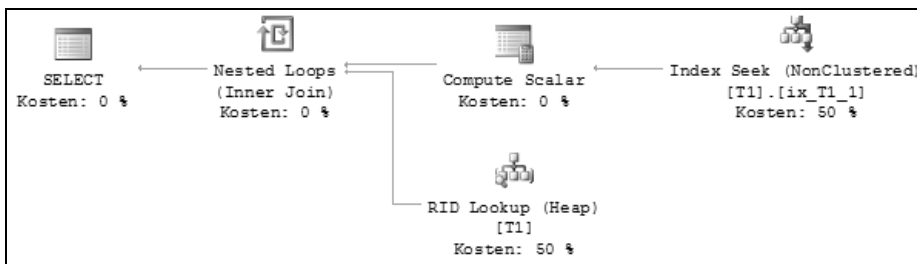


Abbildung 3.3: Ein einfacher Abfrageplan

Der Abfrageplan wird von rechts nach links und von unten nach oben gelesen. Der oben links stehende Knoten repräsentiert also den zuletzt ausgeführten Operator – in unserem Fall die SELECT-Anweisung. Die Pfeile stehen für den Datenfluss, wobei die Stärke eines Pfeils ein Indiz für die Anzahl der Zeilen ist; je dicker also ein Pfeil ist, desto mehr Zeilen werden von rechts nach links übertragen. Für jeden enthaltenen Operator wird außerdem der prozentuale Kostenanteil angezeigt, den diese Operation in Bezug auf die gesamte Abfrage verursacht.

Sobald Sie die Maus über einen Operator oder einen Pfeil bewegen, öffnet sich ein Fenster mit näheren Informationen zum Operator oder Datenfluss. Abbildung 3.4 zeigt dies an einem Beispiel.

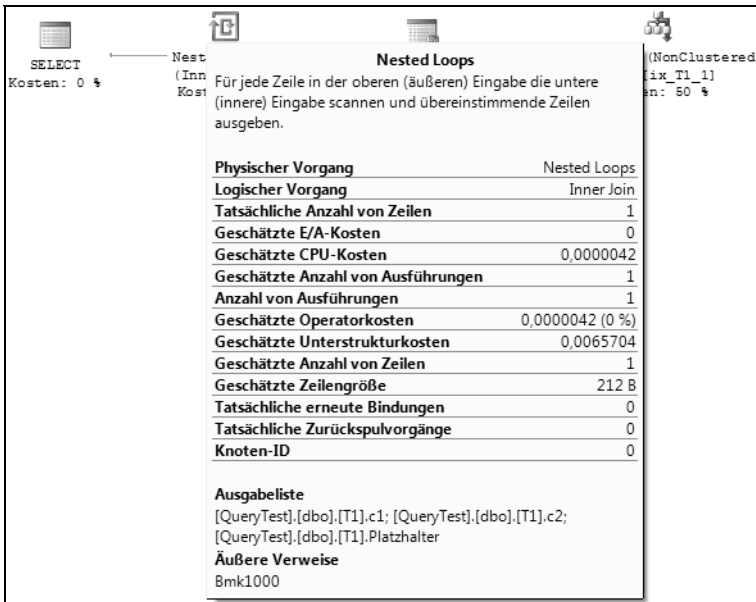


Abbildung 3.4: Informationen über den Nested Loops-Operator

Hier sehen Sie zum Beispiel auch die absoluten Kosten des gesamten Zweiges, also die Kosten des Operators und aller weiter rechts stehenden Operationen. Der linke obere Knoten – also in unserem Fall die SELECT-Anweisung – enthält dann die Kosten des gesamten Plans.

Bei komplexeren Abfragen kann der grafische Ausführungsplan sehr schnell ziemlich groß und damit auch unübersichtlich werden. Dann erfordert es schon einige Übung, den Plan zu interpretieren und eventuelle Schwachstellen aufzufinden. Ich kann Ihnen hier nur empfehlen, dass Sie sich nach und nach in diese Materie einarbeiten. Die hierfür investierte Zeit ist wirklich gut angelegt. Wir werden auch im weiteren Verlauf sehr viel mit Ausführungsplänen arbeiten. Allerdings werden hierbei sehr komplexe Ausführungspläne, mit deren Ausdruck man eine ganze Wand tapezieren könnte, schon allein aus Gründen der Formatierung nicht betrachtet.

Anzeige des Ausführungsplans in Textform

Die Ausführungspläne können auch textuell angezeigt werden – eine Möglichkeit von der wir in diesem Buch allerdings keinen Gebrauch machen werden. Die textuelle Form ist vor allem für den Austausch nützlich, wenn Sie also beispielsweise einen Plan in einer Newsgroup veröffentlichen und diskutieren möchten.

Auch für die Textform können Sie zwischen dem geschätztem und dem tatsächlichen Ausführungsplan wählen. Hierzu stehen Ihnen die folgenden SET-Befehle zur Verfügung:

- ▶ **SET SHOWPLAN_TEXT ON.** Es wird der geschätzte Ausführungsplan in einem Kurzformat angezeigt. Kurzformat deshalb, weil in diesem Plan nur die Operatoren ohne geschätzte Kosten enthalten sind.
- ▶ **SET SHOWPLAN_ALL ON.** Auch hier wird der geschätzte Ausführungsplan angezeigt. Der Plan enthält neben den Operatoren auch die geschätzten Kosten, ist also umfangreicher und aussagekräftiger als bei SHOWPLAN_TEXT.
- ▶ **SET STATISTICS PROFILE ON.** Hier wird der tatsächliche Ausführungsplan angezeigt, die Abfrage wird also ausgeführt. Daher werden hier auch Informationen über die Anzahl der tatsächlich verarbeiteten Zeilen ausgegeben.

Anzeige des Ausführungsplans im XML-Format

Ab der SQL Server-Version 2005 können Ausführungspläne auch im XML-Format ausgegeben werden. Diese Variante ist besonders deshalb interessant, weil das Management Studio Abfragepläne, die im XML-Format vorliegen, für die Darstellung in das grafische Format überführen kann. Dadurch erhalten Sie die eine einfache Möglichkeit, Abfragepläne auszutauschen. Hierfür müssen Sie lediglich XML-Dateien versenden bzw. empfangen.

Die folgenden beiden SET-Befehle erzeugen den XML-Plan als Ausgabe:

- ▶ **SET SHOWPLAN_XML ON.** Es wird der geschätzte Ausführungsplan angezeigt; die Abfrage wird also nicht ausgeführt.
- ▶ **SET STATISTICS XML ON.** Bei dieser Option erhalten Sie den tatsächlichen Ausführungsplan. Dieser Plan wird als zusätzliches Abfrageergebnis ausgegeben.

In beiden Fällen können Sie einfach durch einen Klick auf das ausgegebene XML-Dokument den Plan sofort in das grafische Format überführen. Auch die umgekehrte Transformation ist möglich: Wenn Sie den Ausführungsplan in grafischer Form erstellt haben, können Sie aus dem Kontextmenü dieses Plans das XML-Format erstellen (Abbildung 3.5).

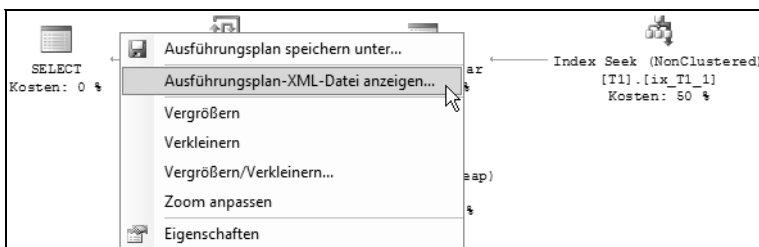


Abbildung 3.5: Den grafischen Ausführungsplan in das XML-Format überführen



Seien Sie bitte vorsichtig mit DDL-Anweisungen, sofern Sie eine Option gewählt haben, die nur einen geschätzten Ausführungsplan anzeigt. Da die DDL-Anweisungen in diesem Fall nicht ausgeführt werden, können Ihre T-SQL-Stapel ein eigenartiges Verhalten an den Tag legen.

Schauen Sie sich zum Beispiel das folgende Skript an:

```
set showplan_text on
go
create table T1(c1 int)
go
insert t1 values(1)
```

Da die Tabelle T1 nicht erzeugt wird, kann für die INSERT-Anweisung kein Ausführungsplan erstellt werden. Stattdessen erhalten Sie einen Übersetzungsfehler, der besagt, dass es keine Tabelle T1 gibt.

3.3 Zusammenfassung

Dieses Kapitel hat Ihnen einige wesentliche Grundlagen über die Ausführung von Abfragen vermittelt. Insbesondere die Analyse von Ausführungsplänen ist eine äußerst nützliche Methode, wenn es gilt, Performance-Engpässe aufzuspüren und zu beseitigen. Diese Methode wird Sie durch den Rest des Buches begleiten. Es ist daher im Moment auch nicht allzu schlimm, wenn bei Ihnen diesbezüglich noch einige Fragen offen geblieben sind. In den verbleibenden Kapiteln werden wir immer wieder mit Ausführungsplänen arbeiten und die offenen Fragen dabei sicherlich beantworten.