„Ich liebe es wenn ein Plan funktioniert!“
Der Ausführungsplan

Thomas Klughardt
Quest Software
Köln

Schlüsselworte:

SQL Statement, Ausführungsplan, Explain Plan, Tuning, Optimierung, Query Optimizer

Einleitung

Jedem, der mit relationalen Datenbanksystemen arbeitet, ist die Sprache SQL ein Begriff. Unter Anderem lassen sich damit Daten abfragen und verändern. Allerdings beschreibt man mit SQL nur, mit welchen Daten man arbeiten will. Wie genau eine Abfrage abgearbeitet wird, das entscheidet dann bei einer Oracle Datenbank der Query Optimizer mit dem sogenannten Ausführungsplan. Dieser Plan beschreibt, in welcher Reihenfolge auf Tabellen zugegriffen wird, ob eventuell vorhandene Indizes genutzt werden und mit welchen Algorithmen die Daten verknüpft werden. Meistens findet die Datenbank einen sinnvollen Weg, um an die Daten zu kommen, ab und zu ist der Zugriff aber auch sehr umständlich. Gerade bei umfangreichen, teuren Abfragen ist es deshalb durchaus sinnvoll, zu überprüfen, wie der Ausführungsplan aussieht und ob man ihn vielleicht von vorneherein verbessern kann.

In diesem Vortrag geht es um Fragen wie: Wie komme ich an einen Ausführungsplan, wie lese ich ihn und welche Informationen ziehe ich daraus? Was muss ich in der Datenbank einrichten, um einen Ausführungsplan zu sehen und welche Tools können mir hier helfen? Was ist der Unterschied zwischen den Ausführungsplänen aus "explain plan for" Statements, der V$SQL_PLAN View oder aus den Trace Files? Was bedeuten Begriffe wie "Full Table Scan", "Nested Loop Join" oder "Kartesisches Produkt"? Was sind Plankosten und warum sind Statements mit höheren Plankosten manchmal doch schneller, als die mit geringeren Plankosten?
Kurz, es geht darum, einen Einblick in die Prozesse zu erhalten, die in einer Datenbank unter der Haube stattfinden und idealer Weise, bessere Statements zu schreiben.

Definition: Was ist ein Ausführungsplan?

SQL ist eine standardisierte, mengenbasierte Sprache, mit der über SQL Statements unter anderem Daten abgefragt und manipuliert werden können. In SQL wird aber nur definiert, mit welchen Daten man arbeiten möchte. Wie die Datenbank an die Daten kommen soll, wird nicht vorgegeben. Darum muss sich also das Datenbankmanagementsystem selbst kümmern, es muss einen Weg finden um an die Daten zu kommen, der natürlich möglichst effizient sein sollte. Dieser Weg heißt Ausführungsplan.
Je komplexer die Abfrage ist und je mehr Objekte wie Tabellen und Indizes beteiligt sind, umso schwieriger ist es, einen guten Ausführungsplan zu finden. Ein guter Ausführungsplan ist aber wichtig, weil er direkten Einfluss darauf hat, wie viel Zeit die Abarbeitung des Statements in der Datenbank in Anspruch nimmt und wie viel Last auf dem System dabei entsteht. Je besser der Ausführungsplan, umso geringer sind Antwortzeit und/oder die entstandene Last.

Deshalb wird eine Menge Aufwand investiert, das Datenbanksystem dabei zu unterstützen, gute Ausführungspläne zu finden.

Wie wird ein Ausführungsplan erzeugt?

Wird ein SQL Statement abgesetzt, das der Datenbank nicht bekannt ist, dann muss es zunächst geparst werden. Neben Dingen die geprüft werden müssen wie: Darf dieser Anwender überhaupt diese Art der Operationen ausführen, oder darf er überhaupt an diese Daten kommen, wird auch der Ausführungsplan erzeugt. Das ist aufwändig, deshalb bekommt das Statement dann einen Schlüssel, eine Hash ID, und der Ausführungsplan wird in einem Cache abgelegt. Beim nächsten Ausführen des Statements erkennt die Datenbank, dass das Statement schon da ist und wird den Ausführungsplan aus dem Cache lesen.
Doch zuerst einen Schritt zurück, wie wird der Ausführungsplan denn jetzt erzeugt? Die Anforderung ist ja, dass er möglichst effizient auf die Daten zugreift. dazu gibt es ein Programm, das versucht, diesen Ausführungsplan zu finden. Dieses Programm ist der Query Optimizer, er ist das komplexeste Programm in einem Oracle Datenbanksystem.

Wie wird ein Ausführungsplan erzeugt?

Wird ein SQL Statement abgesetzt, das der Datenbank nicht bekannt ist, dann muss es zunächst geparst werden. Neben Dingen die geprüft werden müssen wie: Darf dieser Anwender überhaupt diese Art der Operationen ausführen, oder darf er überhaupt an diese Daten kommen, wird auch der Ausführungsplan erzeugt. Das ist aufwändig, deshalb bekommt das Statement dann einen Schlüssel, eine Hash ID, und der Ausführungsplan wird in einem Cache abgelegt. Beim nächsten Ausführen des Statements erkennt die Datenbank, dass das Statement schon da ist und wird den Ausführungsplan aus dem Cache lesen.
Doch zuerst einen Schritt zurück, wie wird der Ausführungsplan denn jetzt erzeugt? Die Anforderung ist ja, dass er möglichst effizient auf die Daten zugreift. dazu gibt es ein Programm, das versucht, diesen Ausführungsplan zu finden. Dieses Programm ist der Query Optimizer, er ist das komplexeste Programm in einem Oracle Datenbanksystem.

Rule Based Optimizer vs. Cost Based Optimizer
In der Vergangenheit gab es nur den Rule Based Optimizer (RBO). Dabei gibt es 15 Regeln für den Datenzugriff, die nacheinander abgearbeitet werden, bis man die erste findet, die zutrifft. Die erste Regel ist: Wenn die RowID, also die physikalische Adresse des Datensatzes, bekannt ist, greife darüber zu. Dann gibt es Regeln über vorverknüpfte Tabellen (Cluster), über eindeutige Indizes, zusammengesetzte Indizes, das Scannen von Bereichen auf Indizes, Sortierungen zur Abfragezeit bis zum schlechtesten möglichen Zugriff, dem Full Table Scan. Der Vorteil des RBO ist, dass es einfach ist vorherzusagen, wie ein Statement abgearbeitet wird. Der Nachteil ist, dass diese Regeln oft nicht zutreffen. Der Full Table Scan ist oft gar keine schlechte Idee, auf Daten zuzugreifen, aber das ist natürlich situationsabhängig.
Deshalb wurde mit Version 8i der Cost Based Optimizer (CBO) eingeführt, der anhand von Statistiken versucht, den besten Zugriffsweg zu finden. Der RBO ist zwar noch da, allerdings wurde mit Oracle 10g jede Entwicklung daran eingestellt, so dass man ihn nicht mehr verwenden sollte und wir ihn auch nicht näher beleuchten. Der Vorteil des CBO ist, dass er situationsabhängig entscheidet, der Nachteil ist, dass sich auch manchmal falsch entscheidet und die Entscheidungen aufgrund ihrer Komplexität schlecht vorhersagbar sind. Damit er überhaupt die Chance hat, eine gute Entscheidung zu treffen, müssen die Statistiken natürlich akkurat sein.

Statistiken
Wie schon geschrieben sind die Statistiken für den CBO essentiell. Üblicherweise werden Statistiken über einen Job aktualisiert, aber auch da gibt es verschiedene Sichtweisen. Während es auf der einen Seite gut ist, gute Statistiken zu haben, ist es auf der anderen Seite ein Change, wenn man Statistiken nimmt. Änderungen in den Statistiken können natürlich dazu führen, dass sich Ausführungspläne ändern. Im Extremfall sind dann große Teile der Anwendung betroffen.

Wie liest man einen Ausführungsplan?

Es gibt verschiedene Stellen in einer Oracle Datenbank, um an den Ausführungsplan zu kommen. So kann man ihn für ein Statement in eine dafür erzeigte Plan Tabelle schreiben lassen, beim Ausführen des Statements mit angeben lasse, über die V$SQL_PLAN View aus dem Shared Pool lesen oder beim Tracen von Sessions mit in die Trace Files schreiben. Der Ausführungsplan hat dann eine Baumstruktur, auch wenn er tabellarisch aufgelistet ist und es pro Planschritt eine Zeile gibt. Ausgeführt und dann auch gelesen wird er von innen nach außen, es werden jeweils die Teilergebnisse für ie Äste gebildet und sie dann zusammengefügt, bis man an der Wurzel ankommt.

Join Algorithmen

Läuft eine Abfrage über mehrere Tabellen, dann müssen die Datensätze der Tabellen über die referenzierten Werte zu einer Gesamtmenge zusammengeführt werden. Man spricht dann von einem Join, einer Verknüpfung der Tabellen. Dabei gibt es verschiedene Vorgehensweisen, die jeweils ihre Stärken und Schwächen haben. Der Query Optimizer entscheidet auch hier auf Basis der Statistiken, welcher Algorithmus am sinnvollsten ist. Hier ist eine zugegebenermaßen nicht hundertprozentig exakte Beschreibung der Algorithmen.

Nested Loop
Der einfachste Weg, Daten aufeinander abzubilden, ist, eine Zeile nach der anderen von der einen Tabelle zu lesen und jeweils die passende Zeile in der anderen Tabelle zu finden. Es sind also zwei geschachtelte Schleifen, die durchlaufen werden. Der Vorteil ist, dass man ohne Vorbereitung anfangen kann zu lesen und so die ersten Ergebnisse sehr schnell bekommt. Der Nachteil ist, dass im schlechtesten Fall für jede Zeile der äußeren Tabelle, alle Zeilen der inneren Tabelle gelesen werden muss, bis man die Richtige findet. Für große zu verknüpfende Datenmengen ist dieser Algorithmus daher ungeeignet.

Sort Merge
Eine Alternative ist es, vorher die Datensätze der Tabellen nach dem Wert zu sortieren, der das Verknüpfungskriterium darstellt. Danach können die Datensätze ineinander gemischt werden, wobei man jede Tabelle nur einmal lesen muss. Allerdings benötigt das vorherige Sortieren Zeit, in der noch keine Ergebnisse geliefert werden können. Die Sortierung muss natürlich auch im Speicher abgelegt werden, und sollte der verfügbare Speicherbereich nicht groß genug sein, dann wird sie in den Temp Segmenten abgelegt, was natürlich IO Kosten verursacht.

Hash
Statt wirklich alle Datensätze zu sortieren, was teuer ist, kann man sie auch partitionieren. Dabei wird eine Hashtable aufgebaut, die aus eine bestimmten Anzahl an Slots besteht, und eine Hashfunktion genutzt, die jeden referenzierten Wert auf einen dieser Slots abbildet. Dadurch erreicht man eine Vorsortierung, in den Partitionen selbst muss man dann nur noch mit kleineren Datenmengen arbeiten. Üblicherweise werden diese dann wieder mit einem Nested Loop Algorithmus zusammengeführt.

Cartesian Product
Es gibt aber noch einen sehr ungünstigen Fall: Es ist für diese konkrete Verknüpfung gar kein Kriterium definiert. Dann kann also kein Wert genutzt werden, um die Datensätze zuzuordnen, sondern man muss mit der Permutation über alle Datensätze, also dem kartesischen Produkt, weitergearbeitet werden. Hat man zwei Tabellen mit n und m Datensätzen, dann ist das Ergebnis dieses Joins n*m Datensätze groß. Wenn irgendwie möglich, sollte man kartesische Produkte vermeiden und ein passendes Join Kriterium definieren.

Zugriffe

Wichtig bei der Abarbeitung eines Statements ist natürlich, wie auf die Daten zugegriffen wird. Im Idealfall kommt man mit möglichst wenigen Zugriffen an alle Daten.

Full Table Scan
Der einfachste Weg an Daten zu kommen, ist, einfach von vorne bis hinten die komplette Tabelle zu lesen. Das dauert zwar meist lange, allerdings muss man jeden Datenblock nur einmal lesen. Für das IO Verhalten sind Full Table Scans also gar nicht schlecht, muss das Ergebnis aber schnell geliefert werden, dann ist er oft nicht das Mittel der Wahl.

Indizes
Ein Index ist eine Baumstruktur und funktioniert wie ein Inhaltsverzeichnis. Will man an einen bestimmten Datensatz, dann kann man über einen Index schnell darauf zugreifen, deshalb wird der Query Optimizer ihn dann wählen. Idealerweise finden sich alle Daten im Index, so dass gar nicht mehr auf den Datensatz selbst zugegriffen werden muss. Leider ist es oft anders, nicht alle Kriterien sind indiziert, nicht alle Daten im Index zu finden und oft sucht man statt einem mehrere Indizes. Daher muss der Query Optimizer die Entscheidung treffen, ab wann es sich nicht mehr lohnt über den Index zuzugreifen, wann also der Full Table Scan effizienter ist.

Wie kann man einen Ausführungsplan verbessern?

Der Query Optimizer kann nur auf Basis der Statistiken entscheiden, wie der Ausführungsplan zu einem Statement aussehen soll. Es lohnt sich deshalb zu prüfen, wie aktuell die Statistiken sind. Doch auch mit aktuellen Statistiken muss der Query Optimizer nicht zwangsläufig den besten Ausführungsplan erwischen, dazu gibt es einfach zu viele verschiedene Möglichkeiten, so ein Statment auszuführen. Als Mensch hat man dann den Vorteil, die Anwendung besser zu kennen als die Datenbank. Man weiß besser, wie die Anwendung die Daten speichert und damit letztendlich auch, wie man am schnellsten darauf zugreifen kann.

Identifizieren schlechter Statements

Natürlich lohnt es sich nicht, alle Statements zu betrachten und zu optimieren. Interessant sind nur die, die man für verbesserungswürdig hält und das sind die, die in irgendeiner Form Probleme verursachen. Um an die Statements zu kommen, gibt es verschiedene Wege. Die Entwickler kennen die Statements meist, alternativ findet man sie im Shared Pool oder kann Sessions tracen um an die Statements und da auch die Wait- und Bindevariablen-Informationen zu kommen.

Kosten
Einen ersten Eindruck kann man sich über die Plankosten verschaffen. Je höher die Kosten, um so aufwändiger ist das Abarbeiten des Statements aus der Sicht des Query Optimizer. Meist sind dann die Statements mit den hohen Plankosten auch die, die in der Produktion die Probleme machen.

Laufzeitmetriken
Die Plankosten basieren auf den Statistiken, sie sind also nur eine Schätzung des Aufwandes. In Wirklichkeit können Statements mit niedrigeren Plankosten aufwändiger und die mit höheren Plankosten günstiger abzuarbeiten sein, deshalb sind die Laufzeitmetriken interessant. Darüber erfährt man, wie lange die Abarbeitung des Statements letztes Mal in Wirklichkeit gebraucht hat und wie viel Resourcen benutzt wurden.

Wartezustände
Die Wartezustände (Waits) geben an, wie lange auf eine bestimmte Ressource gewartet werden musste. Ein Wait kann Idle sein, das heißt es gäbe sowieso nichts zu, oder Non-Idle, dabei muss auf etwas gewartet werden, bevor etwas anderes abgearbeitet werden kann. Die Non-Idle Waits sind die kritischen Wartezustände.
Metriken wie die Antwortzeit stehen für sich und wenn ein Nutzer zu lange auf das Ergebnis wartet, ist das Statement sicher verbesserungswürdig. Schwieriger wird es allerdings bei der Ressourcennutzung. Wie viel CPU Takte sind zu viel oder wann wurde zu viel Speicher in Anspruch genommen? Diese Fragen können nur im Kontext der restlichen Last beantwortet werden und da spielen die Wartezustände eine wichtige Rolle.
Die Laufzeitmetriken im Kontext der Wartezustände sind sehr hilfreich beim identifizieren problematischer Statements. Gibt es hohe Wartezeiten für eine Ressource, dann will man natürlich die Statements verbessern, die diese Ressource viel verwenden.

Offensichtliche Probleme

Einige Probleme im Ausführungsplan kann man sofort erkennen. Ein Beispiel hierfür ist bis auf ganz seltene Ausnahmen ein kartesisches Produkt. Da wurde beim Schreiben des Statements einfach ein Kriterium vergessen und so muss eine sehr große Menge an Daten mitgeschleppt werden.
Ungünstig ist natürlich auch, wenn man nur eine oder wenige Zeilen benötigt, aber trotzdem ein Full Table Scan gemacht wird. Dann gibt es entweder keinen passenden Index, oder er ist unbenutzbar und muss repariert werden.
Meist sind die Probleme aber nicht offensichtlich und man muss etwas genauer untersuchen, wo das Problem liegt.

Optimierung

Hat man problematische Statements identifiziert, dann will man sie natürlich optimieren. Und das ist möglich, solange man realistische Ansprüche an Antwortzeit und Ressourcennutzung stellt. Dabei gibt es einige Möglichkeiten.

Kriterien
Zuerst einmal muss man sich darüber klar werden, was das Optimierungskriterium ist. Bei großen Wartungsjobs, die im Batchbetrieb laufen und große Datenmengen manipulieren optimiert man natürlich nach anderen Kriterien, als bei einer Abfrage, die von einem Anwender gestellt wird der auf das Ergebnis wartet. Mögliche Optimierungskriterien sind unter Anderem die Antwortzeit für das Gesamtergebnis oder die erste Reihe, CPU- oder Speichernutzung sowie IO Last.

Indizes
Oft hilft es, Indizes zu definieren, mit denen schneller auf die Datensätze zugegriffen werden kann. Manchmal hilft es aber auch, Indizes durch Manipulation am Statement unbenutzbar zu machen, um das IO Verhalten zu verbessern. Bei Indizes muss man allerdings beachten, dass so ein Index sich

nicht immer nur positiv auswirkt. Es kann sein, dass dadurch eine Abfrage besser abgearbeitet wird, während andere Abfragen schlechter werden.

Rewrites
Statements werden in den Ausführungsplan umgewandelt. Dabei werden verschiedene Zugriffsmöglichkeiten geprüft, aber eben nicht alle. So ist es möglich, für andere Schreibweisen eines Statements andere Ausführungspläne zu erhalten, die unter Umständen besser funktionieren. Diese Möglichkeit des Optimierens ist sehr mächtig und hat den Vorteil, dass man außer dem Statement selbst ja nichts ändert. Allerdings gibt es schon bei einfachen Statements eine Menge verschiedener Schreibweisen, weshalb man hier manuell nicht sehr weit kommen wird, sondern auf Tools angewiesen ist.

Optimizer Hints
Es ist möglich, dem Query Optimizer direktiven mit auf den Weg zu geben. So kann man etwa festlegen, dass für eine bestimmte Tabelle ein Full Table Scan oder ein bestimmter Index verwendet werden soll, oder forcieren, dass die Tabellen in der Reihnenfolge verknüpft werden sollen, wie in der ORDER Klausel definiert ist. Dadurch kann den Ausführungsplan gezielt verändern, der Nachteil ist, dass man dann die eigentlichen Vorteile des Query Optimizers nicht mehr nutzen kann. Ändert sich die Verteilung der Daten und damit die Statistiken, kann der Ausführungsplan nicht entsprechend angepasst werden.

Outlines, Baselines
Rewrites und Optimizer Hints setzen immer voraus, dass man das Statement selbst ändern kann. Dazu gibt es dann noch die Möglichkeit, einen festen anderen Ausführungsplan für ein Statement zu definieren. Das passiert für die Anwendung transparent, sie setzt ihr Statement ab und bekommt ihre Daten wieder. Unter der Haube wird das Statement dann anders abgearbeitet. Dadurch hat man ein mächtiges Optimierungswerkzeug für Anwendungen, auf die man nicht direkt Einfluss nehmen kann, aber auch wieder das Problem, dass sich der Plan nicht anpassen kann, wenn sie die Datenverteilung ändert. Da man die geänderten Ausführungspläne nicht direkt sieht, läuft man immer Gefahr sie zu vergessen. Deshalb sollten Outlines oder Baselines nur verwendet werden, wenn die anderen Tuningmöglichkeiten ausgeschöpft sind.

Fazit

Automatisiert zu klären, wie Statements sinnvoll abgearbeitet werden, ist es sehr komplexes Problem und es gehören eine Menge Schritte dazu, an die gewünschten Daten zu kommen. Deshalb ist der Query Optimizer so gebaut, dass er nicht den absolut besten Weg findet an die Daten zu kommen, sondern in einer begrenzten Zeit einen möglichst guten Weg. Es gibt also in fast allen Fällen noch Optimierungsmöglichkeiten.
Was es nicht gibt, ist ein Standardvorgehen. Tuning bedeutet immer, jedes Problem für sich zu betrachten und Arbeit zu investieren. Das heißt wiederum, dass man nicht jedes Statement optimieren sollte, sondern eben die, die Probleme machen. Sehr hilfreich sind dann Tools wie das Oracle Tuning Pack, oder der Quest SQL Optimizer, die einfach sehr schnell sehr viele verschiedene Varianten testen können und so in einer vertretbaren Zeit bessere Alternativen finden, als man es von Hand könnte.
Auch wenn das ganze Tuning Thema nicht gerade einfach ist, lohnt es, sich damit zu beschäftigen. Es gibt viele Möglichkeiten, die Performance deutlich zu verbessern.

Kontaktadresse:

Thomas Klughardt
Quest Software
Im Mediapark, 4e
D-50670 Köln

Telefon: +49 (0) 221-5777 4114
Fax: +49 (0) 221-5777 40
E-Mail thomas.klughardt@quest.com
Internet: www.questsoftware.de