Eine Einführung in indexbasiertes SQL-Tuning

Sebastian Wittig
merlin.zwo InfoDesign GmbH & Co. KG
Karlsruhe

Schlüsselworte
SQL, Indizierung, Datenbank, Performance,

Einleitung

Unabhängig von der verwendeten Programmiersprache, dem Verwendungszweck und der genauen Architektur, muss sich eine Anwendung häufig an einer einfachen Anforderung messen: der Geschwindigkeit.
Um diese bei datenbankzentrierten Anwendungen zu optimieren, gibt es eine Reihe von nützlichen Werkzeugen die je nach Datenbankversion und Lizenzierung zur Verfügung stehen können, aber nicht müssen. Eines dieser Werkzeuge ist der Index, ein mächtiges Werkzeug dessen Einsatzmöglichkeiten im Folgenden vorgestellt werden.

Index, was ist das eigentlich?

Ein Index ist eine Datenstruktur, welche die Datenbank erzeugt, wenn es ihr über den CREATE INDEX Befehl mitgeteilt wird. Um diesen überhaupt absetzen zu dürfen, muss mindestens eine der folgenden Bedingungen erfüllt sein:

- Die zu indizierende Tabelle liegt im eigenen Schema
- Das Schema hat das INDEX Privileg für die Tabelle erteilt bekommen
- Das Schema hat das CREATE ANY INDEX Privileg

Ist eine dieser Voraussetzungen wahr, dann können wir mit folgendem Befehl einen Beispielindex erzeugen:

```
CREATE INDEX emp_idx
ON employee (last_name, first_name, phone_number)
```

Mit diesem Befehl erzeugen wir eine Datenstruktur, welche sortierte Informationen über den Nachnamen, Vornamen und die Rufnummer einer Person enthält. Ganz ähnlich der Informationen, die wir in einem Telefonbuch zu finden erwarten. Diese Informationen werden zusätzlich zu den Informationen aus der Tabelle abgelegt. Sie sind also redundant. Damit wir den Speicherplatz also nicht unnötig belegen, sollten wir uns Gedanken machen, welche Informationen im Index gespeichert werden sollten.

Diese sortierte Datenstruktur wird außerdem mit einem bestimmten Verwendungszweck erzeugt. Das ist ähnlich wie mit dem Telefonbuch, sucht man in diesem bspw. nicht nach der Größe einer Person. Sondern die Rufnummer und in selteneren Fällen die Anschrift. Mitunter aus diesem Grund, sind vor dem Erzeugen eines Index Überlegungen nötig, welche Informationen relevant sind.

Doch trotz dieser Überlegungen gibt es einige Fälle, in welchen man einen Index erzeugen sollte. Dazu allerdings später mehr.

Anders als unser Telefonbuch, das nach dem Druck nicht mehr verändert werden kann, kümmert sich die Datenbank darum, dass der Index immer aktuell gehalten wird. Fügen wir also zusätzlich zu den fiktiven Angestellten *Schmidt, Pascal* und *Schmitt, Anton* noch Herr *Schmied, Gerald* hinzu, so wird

dieser korrekt zwischen den beiden eingefügt. Der Index ist aufsteigend sortiert. Die ist implizit der Standard, wenn es um die Erzeugung des Index geht – ganz ähnlich wie bei ORDER BY.

Was findet innerhalb der Datenbank statt?
Es gibt für diese Datenstruktur einige Formen, zu den häufigsten zählen dabei:

- B*Tree Index: Der Standard, auf den wir vorrangig eingehen.
- Bitmap Index: Bei dem bestimmten Schlüsselwerte in Form einer Bitmap mit der ROWID gespeichert werden. Dieser kann sich bei XOR, OR und AND abfragen als nützlich erweisen. Er wird auch im OLAP-Bereich gerne verwendet.
- Funktionsbasierter Index: Dieser enthält keine Spalten, sondern die Ergebnisse einer deterministischen Funktion, welche für die jeweilige Zeile berechnet wird.
- Index Organized Table: Existiert für eine Tabelle ein Index in dem jede Spalte der Tabelle vorkommt, so verzichtet Oracle darauf, diese Tabelle noch vorzuhalten. Die Daten existieren nun innerhalb des Index und sind dort organisiert. Die Idee mag verführerisch sein, schränkt in der Praxis allerdings sehr ein.

Die Datenbank – das einmal unabhängig von konkreten Herstellern – reserviert also Speicherplatz an dem sie in Zukunft Informationen über den LAST_NAME, FIRST_NAME sowie die PHONE_NUMBER, vorhält. Diese Informationen werden bei jedem INSERT, UPDATE und DELETE synchronisiert. Die Indizierung sorgt somit bei jeder Transaktion erst einmal für Mehraufwand. Dieser Mehraufwand lohnt sich allerdings, da wir damit in zahlreichen Anwendungsfällen ein deutliches Performanceplus herausholen können.

Der B*Tree Index
Der Standardindex innerhalb von Oracle ist der B*Tree. Hierbei handelt es sich nicht um einen binären Baum, wie er häufig in Informatikliteratur als Einstiegskonstrukt verwendet wird, sondern um einen balanced Tree, also einen ausbalancierter Baum.

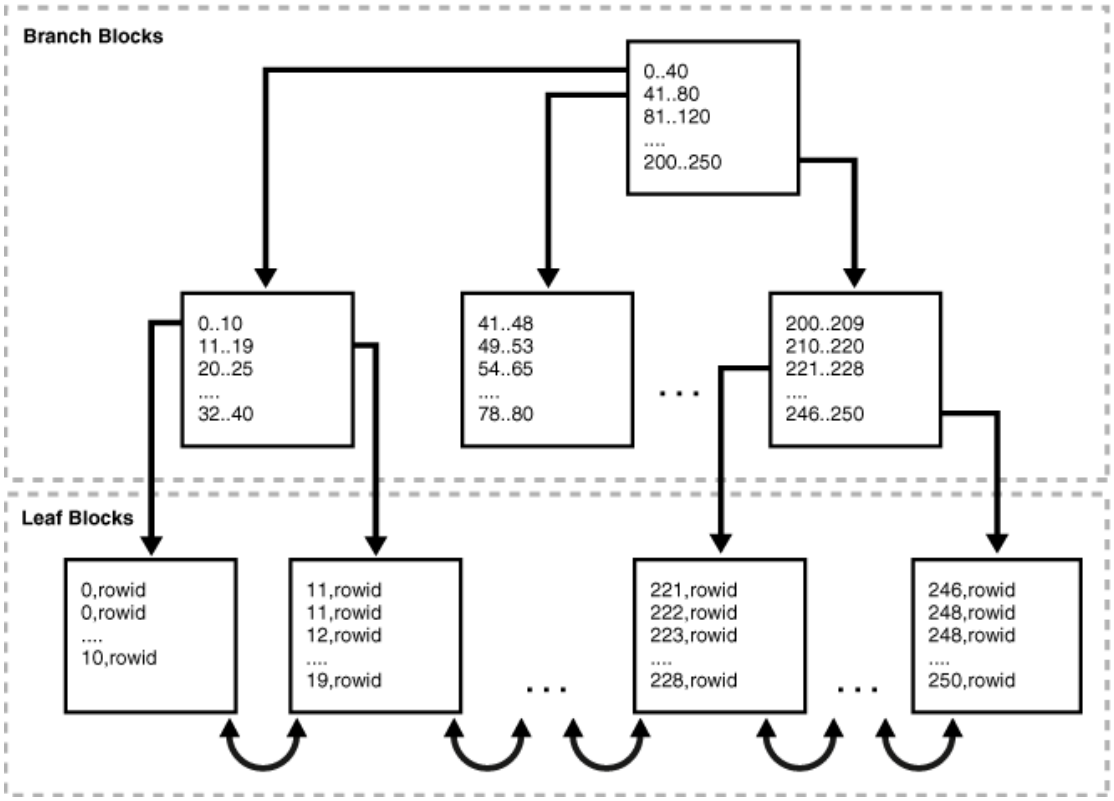

*Abb. 1: Beispiel eines B*Tree Index*

Jede Ebene des Baums enthält dabei mehrere Einträge, die wiederum Verweise für die darunterliegenden Ebenen enthalten. In Abbildung eins enthält der Wurzelknoten einen Eintrag mit dem Wert 0-40, der auf den äußerst linken Wert der darunterliegenden Ebene enthält. Auf dieser darunterliegenden Ebene, dem ersten Zweigknoten, wird der grob zusammengefasste Wertebereich feiner granuliert in Wertebereichen aufgezeigt. Diese feineren Wertebereiche zeigen wiederum auf eine darunterliegende Ebene, welche ihrerseits feiner granulierte Verweise auf die darunterliegende Ebene enthält.
Dieser Vorgang wiederholt sich, bis wir bei den so genannten Blattknoten ankommen. Diese enthalten nur noch den indizierten Wert und die ROWID, also die Adresse der Zeile innerhalb der Datenbank und ermöglichen somit den Zugriff auf die restlichen, nicht indizierten Daten.

Der B*Tree ist ein interessantes Konzept, da jede Ebene mehrere hundert Einträge enthalten kann. Ein Suchbaum gewinnt also mit jeder Ebene exponentiell an Einträgen. Dies ist anderen Suchkonstrukten, wie beispielsweise Listen, deutlich überlegen. Leider ist er allerdings nicht für jede Spaltenkombination gleich gut geeignet. Um die Eignung einer Spalte einschätzen zu können bedienen wir uns dem Begriff der Kardinalität.

Kardinalität und Selektivität
Die Kardinalität einer Menge ist die Anzahl der Elemente in dieser Menge. Im Datenbankkontext wird damit beschrieben, wie viele unterschiedliche Werteausprägungen die einzelne Spalte hat. Man spricht von geringer Kardinalität, wenn die Spalte wenige Ausprägungen hat. Ein mögliches Beispiel dafür ist eine Spalte GESCHLECHT, die Informationen über das Geschlecht der Person enthält. Eine Spalte weißt eine hohe Kardinalität auf, wenn diese ähnlich viele Werte enthält, wie die Tabelle Zeilen hat. Ein mögliches Beispiel wären Nutzerkennungen oder Mitarbeiternummern.

Eng verwandt mit dem Begriff der Kardinaltiät ist die Selektivität. Hierbei handelt es sich um ein Maß, wie gut die WHERE Klausel dafür geeignet ist, die Ergebnismenge einzuschränken. Ein Umstand den Optimizer dazu nutzen den Ausführungsplan zu erstellen.
Eine Abfrage ist sehr selektiv, wenn wir nach dem Nutzernamen einer Person suchen. Dort kommt in der Regel genau eine einzige Zeile zurück, da wir ein Attribut mit hoher Kardinalität gesucht haben. Suchen wir eine Liste aller männlichen Mitarbeiter, so werden wir ca. 50% der Beschäftigten zurückhalten. In diesem Fall ist die Selektivität niedrig, da wir ein Filterattribut mit geringer Kardinalität verwendet haben.

Da der Index für jede passende Zeile einen Tabellenzugriff per ROWID durchführen muss, ist er für sehr selektive Spalten besser geeignet, als für wenig selektive.

Das große Ganze:
Wie wir diese Informationen nun dazu nutzen unsere Abfragen zu optimieren und welche Rolle der Ausführungsplan dabei spielt, das werden Sie in meinem Vortrag am 15.11.16. erfahren.

Kontakt
Sebastian Wittig
Merlin.zwo InfoDesign GmbH & Co. KG
Elsa-Brändström-Straße 14
D-76228 Karlsruhe

Telefon: +49 (0) 721-132 096-44
Fax: +49 (0) 721-132 096-99
E-Mail sebastian.wittig@merlin-zwo.de