Index-Rebuild - The Good, the Bad and the Ugly

Martin Hoermann
ORDIX AG
Paderborn

Schlüsselworte:

Index -Rebuild, blevel, branch blocks, leaf blocks, Index Unique Scans, Index Range Scans, Index Skip Scans, Index Full Scans, Index Fast Full Scans, DEL_LF_ROWS, COALESCE

Einleitung

Dieser Vortrag stellt die Indexzugriffsstrategien von Oracle vor und untersucht wesentliche Kennziffern, die auf diese Zugriffsstrategien Einfluss haben. Hierzu gehören beispielsweise die Höhe des Indexbaums (blevel), die Anzahl Blätter (leaf-nodes) und der Clustering-Faktor. Aber auch weniger beachtete Kennziffern, wie der Speicherparameter PCTFREE und die Anzahl gelöschter Blätter, finden Beachtung.

Anhand verschiedener Szenarien wird untersucht, welche Kosten ein Index-Rebuild nach sich zieht und wie durch einen Index-Rebuild die genannten Kennzahlen verändert werden. Die Auswirkungen sollen auf Unique-Indizes und Indizes mit unterschiedlichen Selektivitäten untersucht werden.

Mit den zuvor genannten Kennzahlen und anhand der beispielhaften Szenarien wird der Sinn und Unsinn von Index-Rebuilds analysiert und diskutiert.

Das Thema Index-Rebuild spaltet die Reihen der Oracle Administratoren. Von möglichst nie bis unbedingt regelmäßig lauten die Empfehlungen von Experten und solchen, die es für sich in Anspruch nehmen. Dieser Vortrag baut auf dem Beitrag „Thanks for the question regarding "Rebuilding Indexes", version 5“ von Tom Kyte unter http://asktom.oracle.com auf.

Sicherlich wird dieser Vortrag nicht die letzten Fragen und Details rund um die Reorganisation von Indizes beantworten. Das Ziel ist es aber, wichtige Denkanstöße zu liefern und wesentliche Fakten rund um das Thema zu präsentieren.

Keine Berücksichtigung finden in diesem Vortrag beispielsweise Bitmap Indexes, Hash Cluster, Domain Indexes, Index Joins und Bitmap Join Indexe.

Indexstruktur

An dieser Stelle wird lediglich ein kurzer Abriss über den Aufbau von B*Bäumen gegeben. B*Bäume bilden die Datenstruktur für herkömmliche Indizes unter Oracle. Eine detaillierte Betrachtung zu den Interna von Oracle Indizes findet sich auf der Homepage von Richard Foote unter http://richardfoote.wordpress.com/. Richard Foote kann sicherlich als die Autorität für das Thema Oracle Indizes genannt werden.

Den Einstiegspunkt in den Index stellt der sogenannte Root-Block dar. Von dort verzweigen zahlreiche Zeiger in die Branch-Blöcke. Abhängig von der Höhe des Indexbaums gelangt man

über verschiedene Ebenen von Branch-Blöcken zu den Leaf-Blöcken. In den Leaf-Blöcken finden sich Wert/Rowid-Kombinationen. Die Rowid zeigt auf die physikalische Adresse der Zeile in der Tabelle zu dem entsprechenden Wert.

Die Leaf-Blöcke sind in ihrer logischen Reihenfolge mit Zeigern verkettet. Dies ist hilfreich, wenn mehrere hintereinander liegende Werte gelesen werden müssen (z.B. BETWEEN-Klausel) und diese sich in unterschiedlichen Leaf-Blöcken befinden.

Die Verwaltung von Indizes ist effizient implementiert. Selbst Millionen von Indexeinträgen kommen oft mit einer Indexhöhe von 3 (blevel=2 + Leaf-Block) aus. Indizes mit einer Höhe von 5 sind selten und einer Höhe von 6 fast nie anzutreffen. Der Indexbaum ist stets ausgeglichen, d.h. alle Leaf-Blöcke werden mit derselben Anzahl an Schritten erreicht.

Indexzugriffsstrategien

In diesem Abschnitt werden kurz die Zugriffsstrategien via Indizes und die Berechnung der entsprechenden Kosten aufgeführt. Die Kosten werden in diesem Artikel nur über die Anzahl der Logical Reads definiert. Die tatsächliche Kostenberechnung ist weitaus komplizierter. Jonathan Lewis behandelt in seinem Buch „Cost Based Oracle“ dieses Thema sehr ausführlich.

Das Verständnis der Zugriffsstrategien und deren Kosten sind als Basis für einen Rebuild unabdingbar. Nur mit diesem Wissen kann über den Erfolg oder Misserfolg eines Rebuild entschieden werden. Für eine vollständige Bewertung sind sehr viele Faktoren wie z.B. CPU-Kosten, Kosten für Single Block vs. Multiblock Reads zu berücksichtigen. Bezogen auf den Index-Rebuild vereinfachen wir hier die Kosten für den Zugriff auf die Anzahl der logischen Leseoperationen. Die Kosten für den Index-Rebuild werden später besprochen.

Index Unique Scans: Bei einem Zugriff über einen Unique Index beträgt die Anzahl zu lesender Blöcke blevel + 1.

Index Range Scans (Ascending/Descending): Bei einem Index Range Scan wird zuerst der erste Treffer ermittelt. Der Zugriff kostet genauso viel, wie der für den Index Range Scan. Anschließend wird der Index entlang der Leaf-Blöcke durlaufen, bis der erste Nicht-Treffer gefunden wird. Die Kosten hierfür sind abhängig von der Ergebnismenge. Sie variieren von den Kosten für einen Unique Scan für einen oder null Treffer und gehen maximal bis zu den Kosten eines Full Scan. Indizes können aufsteigend und absteigend durchlaufen werden.

Index Skip Scans: Bei einem Skip Scan werden alle Werte für das erste Attribut geprüft (Probe) und für jeden dieser Werte wird ein Range Scan durchgeführt. Die Kosten sind abhängig von der Anzahl unterschiedlicher Werte für das erste Attribut und der entsprechenden Treffer in jeder Probe.

Full Scans: Beim Full Scan wird der erste Wert des Index gelesen. Anschließend werden alle übrigen Leaf-Blöcke in der logischen Reihenfolge durchlaufen. Die Kosten addieren sich daher auf (blevel + 1) + (Anzahl Leaf-Blöcke - 1) = blevel + Anzahl Leaf-Blöcke.

Fast Full Index Scans: Beim Fast Full Index Scan werden alle Blöcke eines Index gelesen. Der Index wird mit Multiblock Read gelesen. Die hier nicht diskutierten I/O-Kosten sind dadurch

geringer als beim Full Scan. Im Gegensatz zum Full Scan liefert der Fast Full Index Scan keine sortierte Ergebnismenge.

Zugriffe über funktionsbasierte Indizes, Index Joins und Bitmap Indexes werden in dieser Ausarbeitung nicht berücksichtigt.

Zu allen Kosten muss noch der Zugriff auf die Tabelle multipliziert mit der Anzahl der Treffer bezüglich der Indexspalten addiert werden. Der für die Kostenberechnung des Tabellenzugriffs wesentliche Cluster-Faktor findet in diesem Beitrag keine Berücksichtigung, da dieser durch Index-Rebuilds nicht verändert werden kann. Sind alle Spalten aus der SELECT- und der WHERE-Klausel im Index enthalten so erübrigt sich der Zugriff auf die Tabelle. Dies wird häufig als Index-Only-Zugriff bezeichnet.

Positive Effekte eines Index-Rebuild

Wird ein Datensatz in eine Tabelle eingefügt, so müssen alle Indizes für diese Tabelle gefüllt werden, soweit die Inhalte der indizierten Spalten nicht alle null sind. Hierzu wird die Wert/Rowid-Kombination an die passende Stelle im Index eingefügt. Indexblöcke haben einen Füllgrad zwischen 50 und 100 Prozent, wenn ausschließlich Datensätze in eine Tabelle eingefügt werden. Ist ein Indexblock zu 100 Prozent gefüllt und ein weiterer Wert wird in den Bereich eingefügt, so wird der Block geteilt (leaf node split). Anschließend sind beide Blöcke zu etwa 50 Prozent gefüllt.

Werden Daten monoton aufsteigend in den Index eingefügt und der „rechte“ Block des Index wird geteilt, so entstehen ein fast voller und ein fast leerer Block (leaf node 90-10 split). Dies ist typischerweise dann der Fall, wenn ein Index über eine Sequenz gefüllt wird. „Fast voll“ bedeutet in der Regel nahezu 100 Prozent. Der Begriff „leaf node 90-10 split“ ist dahingehend irreführend, da die vermuteten 10 Prozent nicht frei bleiben.

Durch einen Index-Rebuild wird der Index neu aufgebaut. Alle Indexblöcke bekommen einen Füllgrad von ca. 90 Prozent, da beim Rebuild der Parameter PCTFREE (Default 10 Prozent) wirksam wird. Wird der Füllgrad durch die Reorganisation höher, reduziert sich dadurch die Anzahl der Leaf-Blöcke. Eine Veränderung der Höhe des Indexbaums ist eher selten der Fall. Diese Veränderung führt zu einer entsprechenden Veränderung der Kosten der verschiedenen Zugriffe. Die Komprimierung der Daten wir meist als Grund für einen Rebuild genannt.

Ein Begleiteffekt der Reduzierung der Indexgrößen ist ein besseres Caching der Index- und Datenblöcke. Dies führt ggf. zu einer Reduzierung der physikalischen Lesevorgänge.

Kosten des Index-Rebuild

Die Kosten eines Rebuild setzen sich aus den direkten und den indirekten Kosten zusammen. Die direkten Kosten bestehen aus dem Verbrauch von CPU-Zeit, Erzeugung von Redo-Volumen, Nutzung temporären Speicherplatzes, Belastung durch logischen und physikalischen I/O und Zugriffskonflikten aufgrund von Sperren, um die Wichtigsten zu nennen.

Die indirekten Kosten ergeben sich durch vermehrte Block-Splits wenn Daten nach einer Reorganisation in einen Index eingefügt werden. Liegt der durchschnittliche Blockfüllgrad vor

der Reorganisation bei 75 Prozent, das ist das Mittel zwischen 50 und 100 Prozent, so liegt er nach der Reorganisation bei 90 Prozent. Dadurch häuft sich die Anzahl der Block-Splits nach der Reorganisation. Die Kosten eines Block-Split liegen im CPU-Verbrauch, in der Erzeugung von Redo-Volumen und in der Serialisierung durch das längere Halten von Latches.

Neben den genannten technischen Kosten für einen Rebuild kommen die nicht unerheblichen organisatorischen Kosten hinzu. Hierzu gehören eine sorgfältige Planung, die Erstellung der Skripts und die Kontrolle der Skript-Läufe.

Bewertung Primary Key Index

Recht einfach ist die Bewertung der Reorganisation eine Primary Key Index, der über Sequenznummern gefüllt wird. Da bei streng monoton aufsteigenden Werten ein leaf node 90-10 split durchgeführt wird, der zu einer nahezu hundertprozentigen Füllung führt verschlechtert sich die Kennzahl leaf blocks bei einer Reorganisation mit dem Default von 10 Prozent für PCTFREE. Mit einem Wert für PCTFREE von 0 Prozent bleibt Kennzahl für leaf blocks konstant. Es bleibt den interessierten Leser überlassen etwaige Unterschiede zu einer RAC-Installation mit der empfohlenen Sequenzeinstellung CACHE und NOORDER zu prüfen.

Bewertung Index mit geringer Selektivität

Kommen wir nun zu einem interessanten Fall, bei dem ein Index-Rebuild augenscheinlich zu guten Ergebnissen führt. Wenn eine Spalte eine geringe Selektivität hat, es also zu sehr vielen Wiederholungen des Indexwertes kommt, so kommt es bei dem Index zu den typischen 50-50 Block-Splits. Interessanterweise füllen sich viele der Indexblöcke aber nicht mehr weiter. Dies liegt daran, dass bei identischen Werten die Rowid die Position zum Einfügen bestimmt. Wird die Tabelle nun permanent gefüllt, so sind die Rowids innerhalb eines Extents aufsteigend. Für hintereinander liegende Blöcke wachsen die Datenwerte also immer nur in eine Richtung.

Wird nun ein Index-Rebuild durchgeführt, so verdichten sich die Leaf-Blöcke, ohne dass es im Nachhinein zu extrem häufigen Block-Splits kommt, da die Datenwerte ja nicht wahllos an bestimmte Positionen eingefügt werden, sondern immer in Gruppen wachsen. Der Index komprimiert gut und behält diese Komprimierung auch bei. Bei dieser Form des Indexes sollte durchaus geprüft werden, ob ein PCTFREE von unter 10 Prozent den positiven Effekt verstärkt. Werden Daten aus der Tabelle gelöscht und einzelne Blöcke kommen in die Freispeicherliste, so kann sich dieses Verhalten natürlich stark ändern.

Bezüglich des Caching kann dieser Rebuild sehr positive Effekte haben. Auf der anderen Seite muss natürlich gut überlegt werden, ob ein Index auf Werte mit geringer Selektivität sinnvoll genutzt werden kann, z.B. durch Fast-Full-Scan-Index-Only-Zugriffe. Bei einem Range Scan mit anschließendem Tabellenzugriff dürfte der Komprimierungseffekt nur sehr geringe Auswirkungen haben.

Bewertung Index mit normalverteilter Selektivität

In einem weiteren Test wurde ein Index mit einer Normalverteilung, besser bekannt als Gaußverteilung oder Gaußglocke, gefüllt. Hierzu wurden Werte mit der Funktion `normal` aus dem Paket `dbms_random` generiert. In einem ersten Versuch wurden ca. 1.000 und in einem

zweiten Durchlauf ca. 70.000 unterschiedliche Werte generiert. Das gesamte Volumen lag jeweils bei 1.000.000 Datensätzen. Dann wurde der Index auf der Spalte reorganisiert. Anschließend wurden 100.000, 900.000 und 4.000.000 Daten hinzugefügt und diverse Statistiken betrachtet. Anschließend wurde der gesamte Vorgang ohne Index-Rebuild wiederholt.

Während bei der geringen Selektivität der Rebuild dauerhaft zu weniger Leaf-Blöcken führte und auch die Anzahl der Block-Splits gering war, stieg die Anzahl der Block-Splits und die Zunahme von Leaf-Blöcken im letzten Lauf mit der hohen Selektivität massiv an. Interessanterweise waren, selbst bei der hohen Selektivität, die CPU-Belastung in etwa gleichwertig und die Anzahl Leaf-Blöcke am Ende trotz des Anstiegs in Summe sogar geringer also ohne Rebuild.

Bewertet man die prozentuale Verbesserung für verschiedene Indexzugriffe so ergibt sich eine nennenswerte Verbesserung ausschließlich bei Index-Only-Zugriffen.

Bewertung DEL_LF_ROWS

Die Frage, ob nach dem Löschen von Daten ein Index-Rebuild notwendig ist, kann nur abhängig vom Kontext beantwortet werden. Wenn ein Löschvorgang eine hinreichend große Menge an Daten löscht und der Platz in absehbarer Zeit nicht wiederverwendet wird, so kann eine Reorganisation sehr sinnvoll sein. Hierzu müssen aber die Begriffe „hinreichend“ und „absehbar“ quantifiziert werden. Demgegenüber müssen die positiven Effekte überwiegen. Wenn eine Indexpartition in einen de-facto oder quasi read-only-Zustand geht, ist dies meist ein guter Zeitpunkt eine Bewertung vorzunehmen.

Die häufig referenzierte Maßzahl DEL_LF_ROWS als Kennziffer für einen Index-Rebuild zu verwenden ist fehleranfällig. Eine ausführliche Darstellung zu diesem Thema findet sich wiederum auf der Homepage von Richard Foote unter der Überschrift „DEL_LF_ROWS Index Rebuild Criteria? (Codex)“.

Fazit

Der Vortrag hat gezeigt, welche Kriterien zur Bewertung eines Index-Rebuild herangezogen werden können. Interessanterweise hat bei vielen Szenarien der Rebuild eine Reduktion der Anzahl der Leaf-Blöcke gebracht, die auch beim Einfügen von weiteren Daten Bestand hatte. Besteht der Zugriff aus Index plus Tabelle, so löst sich der Vorteil sehr schnell auf.

Ein typisches Dilemma wurde in diesem Beitrag nur sehr kurz angerissen. Jede Form von Verdichtung führt zu einem besseren Caching und damit zu weniger physikalischen I/O. Dies kann ein ganz wesentlicher positiver Effekt einer Indexreorganisation sein. Auf der anderen Seite der Medaille steht bei der Verdichtung aber immer das Problem der Serialisierung. D.h. je mehr Daten durch eine Sperre oder einen Latch geschützt werden, desto weniger skaliert die Anwendung in Bezug auf die Anzahl der Sessions.

Ein weiteres Thema, welches in diesem Beitrag keine Berücksichtigung gefunden hat, ist die Art und Weise eines Index-Rebuild. Beispielsweise kann durch einen parallelen Neuaufbau viel Zeit „gewonnen“ werden. Dieses Vorgehen bedarf aber einer genauen Untersuchung, da es auch „interessante“ Seiteneffekte hervorrufen kann. Neben der klassischen Rebuild-Syntax

steht mit dem COALESCE ein Mechanismus zur Verfügung, der sehr effizient mit gelöschten Indexeinträgen umgeht. Dieses Verfahren ist aber einen eigenen Beitrag wert.

Ob ein Index-Rebuild hilfreich ist, lässt sich auf der abstrakten Ebene sehr leicht beantworten: Dies ist immer dann erfüllt, wenn die positiven Effekte die Kosten übersteigen. In der Praxis ist diese Frage ausgesprochen schwierig zu beantworten. Es gibt viele klare Fälle, in denen ein Rebuild strukturell nichts bringt und es gibt einige Fälle, in denen die Abwägung relativ einfach ist. Die überwiegende Anzahl von Fällen ist aber hinreichend kompliziert zu beurteilen. Ein sehr wirtschaftliches Vorgehen wird durch die Method-R von Cary Millsap beschrieben.

Kontaktadresse:

Martin Hoermann
ORDIX AG
Westernmauer 12-16
D-33098 Paderborn

Telefon: +49 (0) 5251 / 1063-0
Fax: +49 (0) 1801 / 67439 - 0
E-Mail info@ordix.de
Internet: www.ordix.de