Datenbanktuning

am Beispiel von Oracle 10g

Karsten Förster, 05 IND

27.06.2008

Im Folgenden werden Aspekte zur Leistungssteigerung eines Datenbanksystems betrachtet. Es wird versucht einen möglichst nahen Praxis bezug herzustellen und eine Leitung zu geben um etwaige Leistungspotenziale zu nutzen.

Inhaltsverzeichnis

1. Ziele des Tunings

Beim Tuning eines Datenbanksystems, ist im eigentlichen von der Leistungssteigerung einer Anwendung die Rede. Eine Anwendung deren Betrieb eine Datenbank bedarf. So sollte die Leistungssteigerung auf die Anwendung auszurichten, die von dieser gestellten Anfragen an die Datenbank und das Nutzungsverhalten der Applikation.

So ergeben sich verschieden Ansätze der Leistungssteigerung. Zum einen ließe sich der Netzwerkverkehr minimieren, die Antwortzeit reduzieren, oder die Lastverteilung verbessern.

2. Überblick über das Datenbanksystem Oracle 10g

Um mehr Verständnis für das Tuningpotentials eines Datenbankmanagementsystems zu erhalten, ist es erforderlich die Grundlegende Anfrageverabeitung zu verstehen.

2.1. Anfragebearbeitung in Oracle 9

In Oracle 9 folgt die Anfragebearbeitung folgendem Schema.

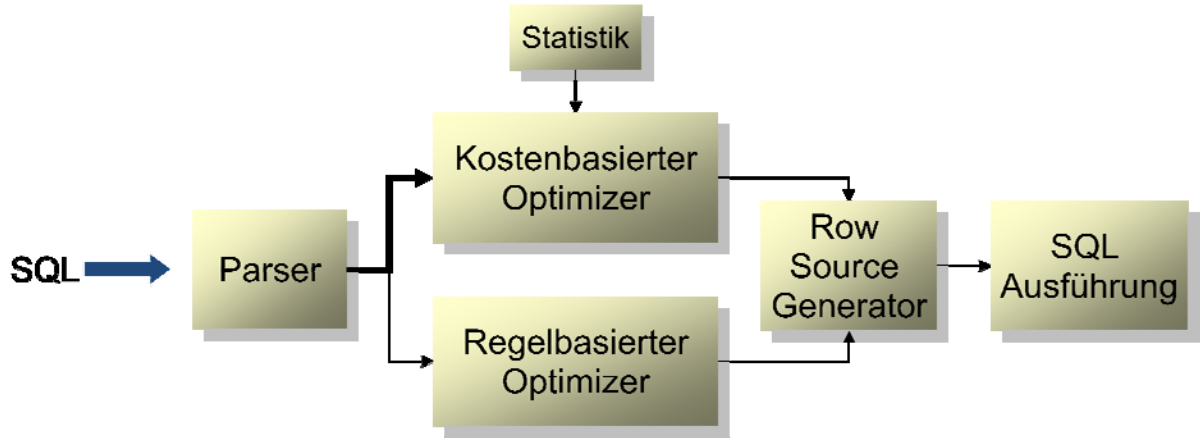


Abbildung 1: Anfragebearbeitung Oracle 9

Zur Erläuterung. Die Sql-Anfrage passiert den Parser, welcher die Sdl-Syntax validiert und in eine vom System ausführbare Form umwandelt. Entsprechend der Systemparameter wird nun entschieden welcher Optimierer zur Anwendung kommt. Im Anschluss erstellt der Row Source Generator den Zugriffsplan, welche letztlich ausgeführt wird.

2.2. Anfragebearbeitung in Oracle 10g

In der Version 10g des Datenbanksystem Oracle wurde der Regelbasierte Optimierer nicht mehr direkt unterstützt. So ergibt sich folgender Ablauf.

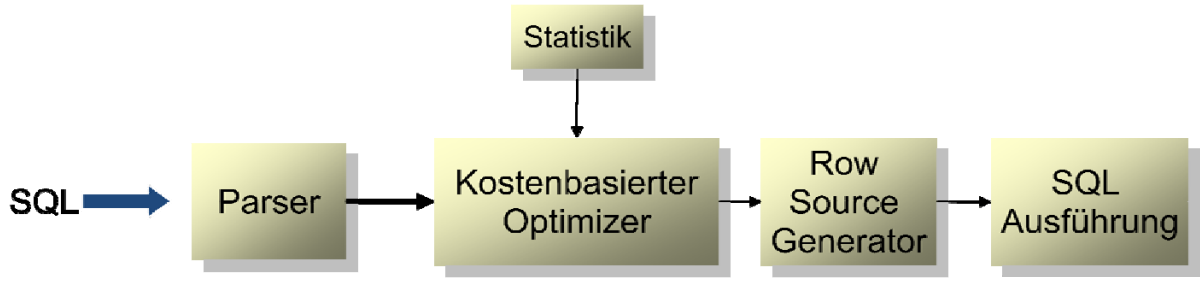


Abbildung 2: Anfragebearbeitung Oracle 9

Es ist ersichtlich das das System nun nicht mehr direkt die Wahl zwischen Regelbasiertem oder Kostenbasiertem Optimierer hat sondern ausschließlich der Kostenbasierte Optimierer zur Anwendung kommt.

2.3. Der Regelbasierte Optimierer

Der Regelbasierte Optimierer legt den „günstigsten“ Plan für die Sql-Ausführung anhand von 15 Regeln fest.

Regeln des Regelbasierten Optimierers:

RBO Path 1: Single Row by Rowid

RBO Path 2: Single Row by Cluster Join

RBO Path 3: Single Row by Hash Cluster Key with Unique or Primary Key

RBO Path 4: Single Row by Unique or Primary Key

RBO Path 5: Clustered Join

RBO Path 6: Hash Cluster Key

RBO Path 7: Indexed Cluster Key

RBO Path 8: Composite Index

RBO Path 9: Single-Column Indexes

RBO Path 10: Bounded Range Search on Indexed Columns

RBO Path 11: Unbounded Range Search on Indexed Columns

RBO Path 12: Sort Merge Join

RBO Path 13: MAX or MIN of Indexed Column

RBO Path 14: ORDER BY on Indexed Column

RBO Path 15: Full Table Scan

Oracle hat die Weiterentwicklung des Regelbasierten Optimierers und dessen Regeln eingestellt, da er zu unflexibel ist. So ist ein festes Regelwerk nahezu unbrauchbar wenn sich ein Datenbanksystem innerhalb eines Grids organisiert ist.

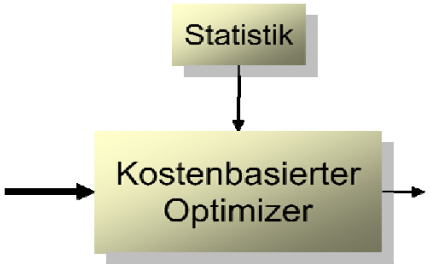
2.4. Der Kostenbasierte Optimierer

Der Vorgang der Anfrageoptimierung ist als Iteration über einige Permutationen der möglichen Ausführungspläne zu betrachten. Hier lege ich großen Wert auf „einige“, es ist nicht praktikabel den Optimierer wesentlich mehr als 50 Permutationen abzuschätzen, da dieser Prozess schließlich auch Zeit kostet und sonst wohlmöglich die gewonnen zeit innerhalb der Ausführung selbst auffrisst.

Leider garantiert uns der Optimierer nicht das „der beste“ Plan zur Ausführung kommt, da den möglicher weise nicht findet, aber er gewährleistet, das nicht „der schlechteste“ Plan ausgeführt wird.

Zur Skalierung der Pläne ist zu sagen, dass serielle Pläne mit geschätzten hohen Kosten, tatsächlich längere Ausführungszeit bedürfen als serielle Pläne mit geringen Kosten. Hingegen sich für parallele Pläne kein direkter Zusammenhang zum Ressourcenverbrauch herstellen lässt.

Wie das Schema zeigt:



Werden die vom Kostenbasierten Optimierer getroffenen Entscheidungen von Statistiken beeinflusst.

Für den effektiven Einsatz des Kostenbasierten Optimierers ist das geschickte setzen spezifischer Parameter vorzunehmen.

3. Optimierungsmöglichkeiten

3.1. Statistiken

Die Entscheidungen des Kostenbasierten Optimierers beruhen auf Daten der Statistik über:

- Tabellen
  - Anzahl Blöcke
  - Tupel
- Spalten
  - Nullwerte
  - Anzahl unterschiedlicher Werte

- Index
  - Anzahl Blattknoten
  - Höhe des Baumes

So ist die Statistik das Argumentationswerkzeug des Optimierers. Somit sollte die Statistik so aktuell und korrekt wie möglich sein. So ergibt sich auch die eine Tuningmöglichkeit. Ändert sich über längere Zeit das Lastprofil nicht, so ist davon auszugehen das die Statistiken sich über die Zeit nicht verändern. Es biete sich an die Statistiken einzufrieren. Das bietet den Vorteil, das die Zeit für das aktualisieren der Statistiken nicht aufgebracht werden muss und Ressourcen für andere Tätigkeiten zur Verfügung stehen.

Das Einfrieren muss letztlich dann wieder aufgehoben werden, wenn sich Änderungen ergeben, dies erfordert ein Mehr an Administrationsbedarf.

3.2. Parameter des Kostenbasierten Optimierers

3.2.1. OPTMIZER_DYNAMIC_SAMPLING

Der Parameter OPTMIZER_DYNAMIC_SAMPLING gibt an wie das Datenbankmanagementsystem reagieren soll wenn keine Statistiken für die betreffenden Objekte vorliegen. So legt dieser fest wie ausführlich eine ad hoc Erhebung statistischer Daten ausfallen soll.

Die möglichen Werte erstrecken sich über 11 unterschiedliche Ebenen. Default Wert ist 2.

Auszug der zur Verfügung stehenden Werte

Level 0: kein dynamisches Sampling

Level 1: die Anzahl der dynamisch gesampleten Blöcke entspricht dem Standardwert (32)

Level 2: führe dynamisches Sampling für alle nicht analysierten Tabellen aus, Die Anzahl des gesampleten Blöcke entspricht 2*Standardwert, 2*32=64

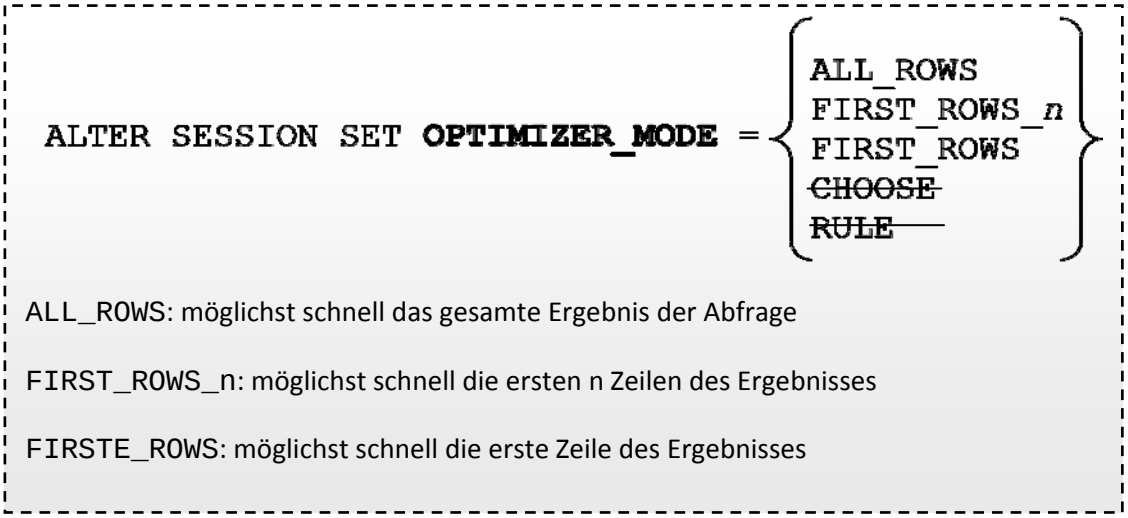
…

Level 10: führe dynamisches Sampling für all Tabellen aus, für die Level 9 zutrifft, verwende für das Sampling alle Blöcke der Tabelle.

Es ist zu empfehlen das genügend statistische Informationen vorliegen, so dass der Wert des Parameters und dessen mit sich ziehendes dynamisches Sampling nicht zum tragen kommen. Sollte es doch zum Sampling kommen ist der Wert so zu wählen das nicht zu viele und nicht zu wenige Teile der Objekte herangezogen werden. So ist der Wert des Parameters stark an den Daten gekoppelt und schwer vorab festzulegen.

3.2.2. OPTIMIZER_MODE

Der Parameter OPTIMIZER_MODE legt das Ziel der Optimierung fest. Die Werte CHOOSE und RULE stehen in Oracle 10g nicht mehr zur Verfügung, da dem Optimierer weder die Wahl steht, noch der Regelbasierte Optimierer gewählt werden kann.



ALL_ROWS: möglichst schnell das gesamte Ergebnis der Abfrage

FIRST_ROWS_n: möglichst schnell die ersten n Zeilen des Ergebnisses

FIRSTE_ROWS: möglichst schnell die erste Zeile des Ergebnisses

3.2.3. OPTIMIZER_FEATURES_ENABLE

Der Parameter OPTIMIZER_FEATURES_ENABLE beeinflust das Optimiere Verhalten um Verhalten Vorgängerversionen des Datenbanksystems zu erzwingen. Oracle rät von der aktiven Verwendung ab, da es nicht nötig ist altes Verhalten zu simulieren. Jedoch kann es zu Testzwecken durchaus interessant sein.

Die zugelassenen Werte entsprechen den Versionsnummern des Oracle Datenbanksystems.

3.2.4. DB_FILE_MULTIBLOCK_READ_COUNT

Gibt die Anzahl gelesener Blöcke innerhalb einer I/O Operation an. Mit steigendem Wert sinken die Kosten für Full Table -/ Index Full Scans. Defaultwert = 8

Dieser Parameter beeinflusst die Bewertung des Optimierers direkt, somit kann direkt Einfluss auf dessen Entscheidung genommen werden.

3.2.5. OPTIMIZER_INDEX_CACHING

Entspricht dem prozentualen Erwartungswert, der im Puffer aufzufindende Indexblöcke.

Begünstigt damit Indexanfragen und Nested Loops.

3.2.6. OPTIMIZER_INDEX_COST_ADJ

Gibt die „normal“ zu erwartenden Zugriffskosten auf den Index an. Es können Werte zwischen 1 und 10 000 angenommen werden. Defaultwert = 100

Ein wert von 10 gibt an das $\frac{1}{10}$ der „normalen“ Zugriffskosten zu erwarten sind.

3.2.7. PGA_AGGREGATE_TARGET

Gibt die Menge des Arbeitsspeichers an, der für Sortier-/Hashoperationen zu verwenden ist.

Somit hat dieser Parameter Einfluss auf das Verhalten bei Joins der entsprechenden Sorte, oder auch Sortierungen.

Minmum: 375 MB

Das Command:

```
SELECT round(PGA_TARGET_FOR_ESTIMATE/1024/1024) target_mb,
        ESTD_PGA_CACHE_HIT_PERCENTAGE cache_hit_perc,
        ESTD_OVERALLOC_COUNT
FROM V$PGA_TARGET_ADVICE;
```

target_mb	Cache_hit_perc	ESTD_OVERALLOC_COUNT
…	…	…
…	…	…

Erzeugt eine Ergebnismenge, von der die ersten beiden Spalten von gesteigertem Interesse sind. Trägt man die Werte (hier einem konkreten Beispiel entnommen) auf ergibt sich folgendes Bild:

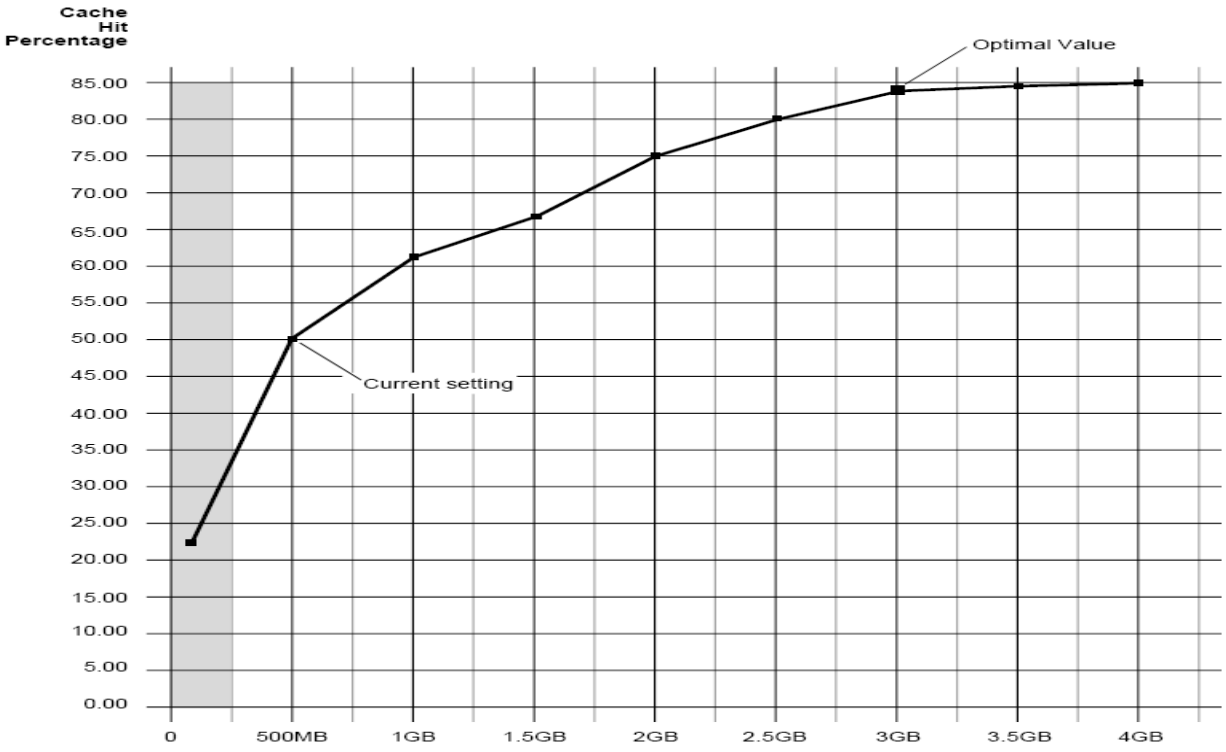


Abbildung 3: Cachhits

Diese Grafik zeigt das Cacheverhalten des Datenbanksystems und den Nutzen eines groß dimensionierten Datenbankcaches. Zunächst ist festzustellen das sich die Kurve asymptotisch den 85% nähert, dies ist darin begründet, das das Datenbankmanagementsystem nicht sofort alles aus dem Cache zugunsten anderer Daten verdrängt, sondern einen ausgewogenen Datenbestand anhand Statistiken im Cache zu halten und der Cache nie vollständig gefüllt sein soll. Des weitern sieht man, das eine Prameteränderung von 3 GB auf 4 BG nicht lohnt, da die Menge der aufgefundenen Daten um nur 2% steigt, somit ist die Aufwendung von einem GB mehr unangemessen.

3.3. Hints

Hints sind vom Anfragenformulierer gegebene Hinweise an den Optimierer, wie er seine Plan anpassen könnte. Wie im Leben so sind auch hier die Hinweise zu verstehen, der Optimierer muss den Hinweis nicht beachten. Somit ergibt sich ein großer Mangel der Hints, es gibt keine Garantie für deren Beachtung und damit kann man sich nicht auf sie verlassen.

3.3.1. Syntax

Hints folgen stets den Schlüsselwörtern: SELECT, INSERT, UPDATE, MERGE oder DELETE.

Sie folgen diesen Schlüsselwörtern als Kommentar mit angeführtem `+`

{DELETE|INSERT|MERGE|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */

Es ist darauf zu achten, das das `+` direkt, ohne Leerzeichen dem `*` folgt.

Durch die Angabe in Kommentarzeichen, liegt der Einbindung der Hints liegt keine spezielle Syntax zugrunde. Das ist negativ zu werten und man könnte vermuten, das diese nie konkret vorgesehen waren. Leider sind Hints aus gleich drei Gründen unsicher:

- Ist ein Hint syntaktisch inkorrekt wird er nicht beachtet
- Ist ein Hint syntaktisch korrekt muss er nicht beachtet werden
- Wird eine Abfrage mehrfach ausgeführt, um einen Hint zu testen, ist liegen die Daten meist schon im Cache und somit sind kaum Aussagen über Wirksamkeit möglich

3.3.2. Unterscheidung

Hints unterschieden sich ihrer Ausrichtung nach

- Zielorientierte
- Ausführungsplan
- Abfragetransformation
- Join-Reihenfolge
- Join-Operation
- Parallelausführung

3.3.2.1. Beispiel Zielorientiert

```
SELECT /*+ ALL_ROWS */ empno, ename, sal, job

FROM emp

WHERE empno = 7566;
```

Gibt dem Optimierer den Hinweis, möglichst schnell das gesamte Ergebnis zu liefern.

3.3.2.2. Beispiel Ausführungsplan

```
SELECT /*+ FULL(A) don't use the index on accno */

        accno, bal

FROM accounts a

WHERE accno = 7086854;
```

Gibt dem Optimierer den Hinweis statt des existierenden Indexes, einen Full Table Scan für die Relation a, also die Tabelle accounts durchzuführen.

3.3.2.3. Beispiel Kombinierter Hint

```
SELECT /*+ LEADING(e2 e1) USE_NL(e1) INDEX(e1 emp_emp_id_pk)
USE_MERGE(j) FULL(j) */
        e1.first_name, e1.last_name, j.job_id,
        sum(e2.salary) total_sal
FROM employees e1, employees e2, job_history j
WHERE e1.employee_id = e2.manager_id
AND e1.employee_id = j.employee_id
AND e1.hire_date = j.start_date
GROUP BY e1.first_name, e1.last_name, j.job_id
ORDER BY total_sal;
```

Der Hint **LEADING(e2 e1)**, weist den Optimierer darauf hin die Relation e2 als fürhende zu verwenden und bei Joinoperationen die Relation e1 an ihr vorbei zufürhen.

USE_NL(e1) weist an, das bei einem Join mit der Relation e1 ein Nested Loop durchzuführen ist.

INDEX(e1 emp_emp_id_pk) weist, das für den Zugriff auf die Relation e1 der Index emp_emp_is_pk zu verwenden ist.

USE_MERGE(j) weist an, das für die Relation ein Mergejoin auszuführen ist.

FULL(j) weist an, das ein Full Table Scan für die Realtion j auszuführen ist.

3.3.3. Einschätzung

Hints sind kritisch zu bewerten, neben der fehlender Garantie ihrer Beachtung bringen sie auch administrativen Aufwand mit sich. Der Datenbankadministrator und der Anwendungsentwickler sind in der Regel nicht ein und die selbe Person. Somit entsteht eine Anwendung ohne genaue Kenntnisse über gebotene Strukturen und umgekehrt kennt der Datenbankadministrator nicht die exakten Anforderungen der Anwendung an die Datenbank. Somit ist hier Kommunikationsbedarf erforderlich.

Des Weiteren ist die Verwendung von Hints ein Indiz für einen mangelhaften physischen Entwurf, fehlende Zugriffsstrukturen oder schlechte Statistiken.

3.4. SQL Optimierung

3.4.1. Vermeiden von SELECT *

SELECT * ist zu vermeiden, da man die Möglichkeit der Selektivität ausräumt.

3.4.2. Vermeiden von ORDER BY

Häufig sind Sortierungen unnötig, wenn man auf sie verzichten kann, sollte man das auch tun. Ein ganz besonders unnötiger Fall einer Sortierung liegt vor, wenn in einer Subquery solch eine erfolgt.

Oftmals lassen sich Sortierungen auf dem Client ausführen. Da Sortierungen ohnehin ein bestandteil der Sicht auf die Daten ist, ist der Client auch genau der richtige Ort um eine Sortierung auszuführen.

3.4.3. UNION ALL ist UNION vorzuziehen

Diese spezielle Form der Vereinigung des UNION Operators erzwingt ein DISTINCT, so dass es nur einzigartige Datensätze in der Menge enthalten sind.

Kann man auf diese eigenschaft, der Duplikatfreiheit verzichten, sollte man den UNION ALL Operator verwenden, den dieser erzwingt kein DISTINC und lässt damit Duplikate zu und schont Ressourcen.

3.4.4. Vermeiden von Subqueries

Subqueries mit IN oder EXIST Operator sind generell zu vermeiden. Sollte man doch von Ihnen Gebrauch machen, beachte man folgendes:

EXIST ist dann zu verwenden wenn die äußere Menge selektiver ist.

IN ist dann zu verwenden wenn die innerer Menge selektiver ist.

3.4.5. Vermeiden von langen IN Listen

IN Listen entarten OR verknüpften Bedingungen und schon allein dadurch zu meiden, da OR´s eine Menge nur vergrößern können. Des weiteren sind IN Listen ein Indiz für eine fehlende Relation, so sollte man, wenn diese Liste ständig angeführt wird, eine entsprechende Realtion anlegen.

4. Indexstrukturen

Indexstrukturen im Allgemeinen sollten jedem bekannt sein. Es stellt sich die wo lässt sich tunen. Zum einen existieren zusammengesetzte Indizes, die würden erforder, das das Datenbankmanagementsystem jede vor stehenden hierarchie durchlaufen werden muss um zugriff auf die Daten / Verweise zu nehmen. Um dies zu minimieren kann man das Datenbankmanagementsystem dazu anweisen strukturen zu überspringen, sogenannter Skiped Index.

Eine weitere Möglichkeit großen Nutzen aus Indexstrukturen zuziehen sind indexorganisierte Tabellen. So liegen alle Indizierten Daten und nicht nur Verweise im Cache. Daraus ergibt sich, das es möglich ist Operationen Komplett mit Indexzugriffen zu gewährleisten.

5. Materialisierte Sichten

Materialisierte Sichten bieten enormen Leistungsschub für ein Datenbankmanagementsystem.

Den genauen Nutzen muss man wie folgt abwiegen:

5.1. Einordnung: Es ist keine absolute Aktualität der Sicht auf die Daten nötig

Hier biete Materialisierte Sichten die Möglichkeit, komplexe Verknüpfungs und kalkulations Operationen voraus zu berechnen und zu materialisieren. Jedoch sollte man für die Aktualisierung dennoch ein genügend engen Zeitraum wählen. Wenn möglich ist diese Zeit zu wählen wenn das System eine geringe Auslastung erfährt, Predeistiniert sind dafür Nächte, sofern das System zu Nacht nicht aktiv ist, Mittagspausen, oder Wochenenden. Häufig sind Datenbanksysteme nicht diesem allgemeinen Rhythmus unterworfen, so muss ein abgestimmter Aktualisierungsrhythmus erstellt werden.

Der Gewinn liegt darin, das die Antwortzeit aller Fragen verbessert wird, da zum einen die Zeit fürs Joins und zum Anderen für Berechnungen gespart wird und somit die gesamte Last sinkt und mehr Ressourcen zur Verfügung stehen.

5.2. Einordnung:Stets aktuelle Sicht auf die Daten

Materialisierte Sichten sind in diesem Fall ungeeignet, da die Aktualität mit hohen Kosten verbunden ist. So möge man keine Materialisierten Sichten verwenden und bei weiterhin ordinären Anfragen absetzen.

6. Quellverzeichnis

Oracle Programmierung, Datenbankprogrammierung und –admnistration, Henz-Ger Raymans ADDISON-WESLEY

Oracle Database, Performance Tuning Guide, 10g Realease 2 (10.2), Immanuel Chen, Oracle, 2005

ORACLE DATABASE 10g Performance Tuning, Tips & Techinques, Richard J. Niemiec, McGrawHill/OraclePress

Oracle Database 11g DBA Handbook, Bob Bryla, Kevin Loney, McGrawHill/OraclePress