Oracle Datenbank Mythen

Robert Baric
ITGAIN Consulting Gesellschaft für IT-Beratung mbH
Hamburg

Schlüsselworte
Oracle, Datenbank, Mythen, Index Rebuild, Clustering Factor, B-Tree, Datapump,, Deadlock, Memory_Max_Target, Index Block Split, AMM, Index, count(*), count(1), count(rowid), sqlplus

Einleitung
Fast alle Völker haben ihre sagenhaften Geschichten über böse Geister und Kobolde, Monster und Drachen. Damals wie heute erzählt man sich voller Spannung Geschichten über die alten Mythen aus längst vergangener Zeit.
Auch im Oracle-Reich ranken sich Erzählungen über seltsame Dinge, böse Geister und wie man sich vor Ihnen zu schützen vermag.

Begleiten Sie den Autor auf einem Streifzug ausgewählter alter und manchmal noch immer aktueller Mythen.

Inhaltsverzeichnis

Count(*) ist böse
Bekanntlich dient der Stern (*) bei Select Anweisungen als Platzhalten für *alle* Spalten einer Tabelle. Er stellt eine beliebte Kurzschreibweise dar.

Und eine solche Abfrage über alle Spalten weckt vielleicht die Befürchtung, dass ein Full-Table-Scan (FTS) statt eines Indexes genutzt wird und mehr Kosten verursacht.

Einige Alternativen sind daher die explizite Angabe einer indexierten Spalte oder die Verwendung der Rowid. Am häufigsten wird die Alternative count(1) verwendet. Mit der „1“ wird je nach Auffassung die erste (meist Primary Key) Spalte referenziert oder durch Angabe eines Literals ein Zugriff auf Tabellenwerte verzichtet.

Damit ergeben sich als Alternativen:

```
Select count(1) from mytable;
Select count(rowid) from mytable;
Select count(<index column>) from mytable;
```

Welche der vier Möglichkeiten sollte man am besten wählen?

Mythos: Bessere Performance ohne Count(*)

Ursprung
nicht bekannt

Wahrheit
Der Count(*) Ausdruck stellt eine eigene, spezielle Funktion dar und der Stern wird nicht durch alle Spalten ersetzt. Das Verhalten ist anders als count(<ausdruck>) wie später erklärt wird.

Der Cost Optimizer erstellt die gleichen Ausführungspläne, sofern

1. die Spalte eine Not Null Constraint besitzt
2. und ein Index für diese Spalte vorhanden ist.

Die Beispiele sind somit äquivalent, wie sich schnell durch einem Vergleich der Ausführungspläne zeigen lässt.

Ein häufiger Grund liegt im Count Transformationsprozess, der count(n) gegen count(*) austauscht, sofern die angegebenen Bedingungen erfüllt sind.[1] Diese Transformationsprozesse können durch Traces sichtbar gemacht werden.[2]

Der Count Befehl
Beachten Sie, wie Sie die Count Funktion verwenden möchten.

Die Funktionsweise von Count(<Ausdruck>) und Count(*) unterscheiden sich in einem wichtigen Detail:

> *Der Count(*) zählt Zeilen mit Null Werten während count(<Ausdruck>) Null Werte nicht zählt.*[3]

Ist ***kein*** Not Null Constraint vorhanden ändert sich beim Vergleich zwischen Count(*) und count(<Ausdruck>) vielleicht nicht nur das Resultat, sondern auch der Ausführungsplan!

---

[1] Troubleshooting Oracle Performance, Christian Antognini,2014, p.174
[2] Event 10053 oder alter session set events 'trace[RDBMS.SQL_Transform]’
[3] http://docs.oracle.com/cd/E11882_01/server.112/e26088/functions039.htm#SQLRF00624 Abschnitt „Purpose“

Beispiel

Erstellen Sie eine Tabelle TCOUNT mit Not Null Spalte oder Primary Key und befüllen sie diese. Mit Hilfe von sqlplus und der Autotrace Einstellung können sie die Ausführunspläne aller Schreibweisen einfach vergleichen.
Ein Befehl zum Aktivieren von Autotrace lautet „set autotrace on explain“.

Verweise

http://use-the-index-luke.com/de/blog/2013-08/its-not-about-the-star-stupid
https://asktom.oracle.com/pls/asktom/f?p=100:11:0::NO::P11_QUESTION_ID:1156159920245
https://asktom.oracle.com/pls/asktom/f?p=100:11:0:::P11_QUESTION_ID:1156151916789
http://www.oracledba.co.uk/tips/count_speed.htm

Deadlocks sind tödlich
In der weiten Welt der Oracle Funktionen gibt es ein Phänomen das den Ruf hat tödlich zu sein wie ein schwarzes Lock. Ich spreche von Deadlocks.

Ihnen wurde in vergangener Zeit nachgesagt, dass bei zwei Sessions die einen Deadlock aufweisen eine den Tod stirbt – sie soll einfach abgeschossen werden.

Ist Ihnen dies schon einmal bei einer Ihrer Sessions bei einem ORA-00060 passiert?

Beispiel

Session 1	Session 2	Kommentar

`Create table T1(`
`  C1 Number Primary Key,`
`  C2 Varchar2(10) );`		

`Insert Into T1 values (1,'1');`	`Insert Into T2 values (2,'2');`	
	`Insert Into T1 values (1,'3');`	Ausführung hängt
`Insert Into T2 values (2,'2');`		
	ORA-00060 deadlock detected	

Das Beispiel stammt von Charles Hooper [4].

Mythos: Deadlocks killen Sessions

Die Wahrheit
Tatsächlich wird weder Session 1 noch Session 2 terminiert und keine von beiden wird *vollständig* zurückgerollt.
Lediglich die letzte Deadlock Aktion wird nicht ausgeführt. Fragt man die Tabelle T1 in Session 2 nach dem Deadlock ab, erhält man folgendes Ergebnis:

Session 1	Session 2
	….
	`Insert Into T1 values (1,'3');`
`Insert Into T2 values (2,'2');`	
	ORA-00060 deadlock detected
	`SELECT * FROM T1;`
	`C1 C2`
	`-------- -------`
	`2 2`

Verweise
http://hoopercharles.wordpress.com/2012/01/04/faulty-quotes-7-deadlock-kills-sessions/
http://arup.blogspot.de/2013/04/application-design-is-only-reason-for.html
How To deal with INITTRANS/MAXTRANS ORA-00060 / Deadlocks (Doc ID 164661.1)

[4] http://hoopercharles.wordpress.com/2012/01/04/faulty-quotes-7-deadlock-kills-sessions/

Datenbank Memory auf Abwegen..
Wenn Sie viele Datenbanken betreiben, dann nutzen Sie vielleicht mehrere Datenbankinstanzen auf einem Serversystem. In diesem Fall werden sie die Ressourcen Ihres Systems aufteilen. Wie verteilen Sie den Memory solcher Oracle Datenbanken?

Wahrscheinlich werden Sie die entsprechenden Oracle Parameter setzen. Seit Version 11g ist die Chance groß, dass sie das Automatic Memory Management (AMM) einsetzen und den Memory_Max_Target und Memory_Target Parameter nutzen. Vielleicht setzen Sie ihre Memory Parameter manuell wie den Pga_Aggregate_Target Parameter.

Haben Sie sich schon mal gefragt, ob diese Grenzen sicher eingehalten werden?

Wahrscheinlich haben sie unter Verwendung von Memory_Max_Target Meldungen erhalten, dass Sie andere Parameter nur innerhalb dieser Grenzen setzen dürfen. Und bei der Verwendung der Datenbank haben Sie vielleicht sogar Out-Of-Memory Fehlermeldung gesehen. Damit scheint gesichert, des es funktionierende Begrenzungen gibt.

Warum nur verwendet man das Wort „Target“ in den Parameternamen?

Mythos: Memory_Max_Target stellt absolute Grenze für den DB Memory dar.

Ursprung
Memory Target Parameter werden als absolutes Limit angenommen.

Wahrheit
Memory_Max_Target und Pga_Aggregate_Target stellen kein absolutes Memory Limit dar.

Die PGA Größe kann beispielsweise sehr einfach durch den Einsatz von PL/SQL Collections überschritten werden:

> „There are certain areas of PGA that cannot be controlled by initialization parameters. Such areas include PL/SQL memory collections such as PL/SQL tables and varrays.“5

Beispiele
Erstellen Sie einen PL/SQL Code und Nutzen Sie zur Speicherung von Daten eine Collection.
Laden sie so lange Daten in die Collection und prüfen Sie den Memory Verbrauch.
Wiederholen Sie den Vorgang bis eine Fehlermeldung erscheint oder die Memory_Max_Target Grenze überschritten wird.
Vergrößern Sie die Collection, so dass die Größe des Shared Memory File System (/dev/shm) überschritten wird und Swap Space verwendet wird. Beachten Sie, dass der letzte Vorgang das System verlangsamt und ggf. in einen instabilen Zustand versetzt.

Führen sie das Beispiel nicht auf Produktivsystemen durch.

Anmerkungen
Ab Oracle Version 12c kann durch den Parameter PGA_Agregate_Limit der PGA Memorybereich mit Setzen einer oberen Grenze gesichert werden. Die minimale Default Einstellung für diesen Parameter liegt bei 2 GB und ist zudem Abhängig von der PGA_Aggregate_Target Größe.
Damit liegt der PGA_Aggregate_Limit praktisch immer weit über der PGA_Aggregate_Target Größe. Die Berechnung des maximal benötigen Hauptspeichers einer Datenbank fällt damit weit größer aus als der Max_Memory_Target Parameter vermuten lässt.

Oracle empfiehlt den PGA_Agregate_Limit Parameter nicht unterhalb der Default Einstellung zu setzen.[6]

---

[5] MEMORY_MAX_TARGET CAN BE EXCEEDED (Doc ID 1269898.1)

SGA_TARGET ist ein Parameter des Automatic Shared Memory Managements. Wenn gesetzt, werden die folgenden Memory Pools Buffer Cache, Shared Pool, Large Pool, Java Pool und Stream Pool automatisch nach Bedarf angepasst. [7]

SGA_MAX_SIZE setzt die maximal verfügbare Größe der SGA als absolute Größe. Beim Setzen von den Memory Target Parametern erhält dieser Wert entsprechend Regeln neue Default Größen.[8]

[6] Oracle Cooperation, Database Reference, PGA_AGGREGATE_LIMIT, http://docs.oracle.com/database/121/REFRN/refrn10328.htm#REFRN10328
[7] Oracle Cooperation, Database Reference, SGA_TARGET, http://docs.oracle.com/cd/E18283_01/server.112/e17110/initparams231.htm
[8] Oracle Cooperation, Database Reference, SGA_MAX_SIZE, http://docs.oracle.com/cd/E18283_01/server.112/e17110/initparams230.htm

Ein Null lässt sich nicht im Index fangen

„Einst hatte es mich verstört. Oracle Indexe sollen sich nicht so verhalten, wie es sich gehört. Denn bei anderen Datenbanken, verwendet der Index den Nullwert in seinen Ranken.

Gesagt, probiert und eine Nullable Spalte indexiert. Kurz darauf die Spalte mit „IS NULL" abgefragt und der Index kam nicht zum Ausführungsplan.

Was gibt es hierzu noch zu sagen? Indexe haben wohl nie Null in sich getragen."

Mythos: Indexe speichern keine Null Werte

Ursprung

Die Beobachtung, dass B-Tree Indexe mit einer Spalte keine Null-Werte enthalten wird auf zusammengesetzte Indexe mit mehreren Spalten übertragen.

Ein konkretes Beispiel:
Bei einer Tabelle „T1" mit 100 Zeilen enthalten 50 Zeilen bei der indexierten Spalte „Spalte1" den Wert Null.
Dann enthält ein Index der mit dem Befehl „create index Index1 on T1 (Spalte1)" angelegt wird nur 50 Werte.

Wahrheit

Bei B-Tree Indexen wird ein Datensatz nicht aufgenommen, wenn alle Spalten des Indexes Null sind.

Häufig wurden bei Beispielen nur Indexe mit einer Spalte betrachtet. Diese enthalten keine Nullwerte.

Zwei Optionen bei dem im Index Null Werte gespeichert werden:
1. Zusammengesetzter Index mit mindestens einer Not Null Spalte oder einer Spalte ohne Null Werte.
2. Zusammengesetztem Index mit einem Literal.

```
1. create index i1 on t2 (c2,c3);
2. create index i3 on t2 (c2,'1');
```

Beispiele

Durch das Betrachten eines Ausführungsplans lässt sich nachvollziehen, ob ein Index Null Werte enthält. Wird nur der Index beim Abfragen von Null Werten verwendet, so müssen folglich die Daten in diesem enthalten sein. Ansonsten wäre ein zusätzlicher Zugriff auf die Tabelle nötig.

Legt man einen Index mit den oben genannten Indexoptionen an und fragt die Tabelle mit „IS NULL" ab, ist der Nachweis erbracht. Wir nur auf den Index zugegriffen wird, muss folglich die Null im Index gespeichert sein.
Alternativ lässt sich durch einen Indexdump der Inhalt der Indexblöcke betrachten und dadurch direkt die Null Werte nachweisen[9,10]s

Anmerkungen

Ein Bitmap Index speichert Null-Werte auch wenn alle Indexspalten Null enthalten.

---

[9] Shailesh Mishra, Oracle Blog, https://blogs.oracle.com/sysdba/entry/how_to_dump_oracle_data_block
[10] Richard Foote, Richard Foote's Oracle Blog, https://richardfoote.wordpress.com/2010/02/08/index-block-dumps-and-treedumps-part-i-knock-on-wood/

Exit, Rollback?
Bei den ACID Eigenschaften eines Datenbanksystems seht der Buchstabe „A“ für Atomizität: Entweder wird eine Transaktion ganz oder gar nicht ausgeführt.

Wird eine Transaktion nicht abgeschlossen, so führt die Datenbank ein Rollback durch. Dieses Verhalten sollten Sie immer beobachten können.

Vielleicht tun Sie dies manchmal auch nicht...

Beispiel
Führen Sie das folgende Skript in sqlplus aus.

```
SQL> show autocommit
autocommit OFF
SQL> create table TEXIT (a number);
SQL> Insert into TEXIT values (1) ;
SQL> Exit;

#> SQLPLUS <User/Password>
SQL> select * from texit;

A
--
1
```

Ergebnis: Die Insert Opertion wurde trotz abgeschaltetem autocommit festgeschrieben.

Mythos: Beenden einer Transaktion ohne Commit führt immer zum Rollback

Ursprung
Annahme, das Verhalten bei Session Abbruch ist gleichzusetzen mit ordentlichem Beenden eines Tools oder Utilities.

Wahrheit
Tools wie SqlPlus führen bei Exit als Standardverhalten ein Commit aus:

„A user exits normally from most Oracle Database utilities and tools, causing the current transaction to be implicitly committed.“[11]

In Sqlplus ist dieses Verhalten unabhängig von der Autocommit Einstellung, die nach einer festzulegenden Anzahl von Kommandos automatisch ein ‚commit’ ausführt.[12]

Verweise
Uwe Küchler, Oraculix, http://oraculix.wordpress.com/2011/03/30/sqlplus-autocommit-und-exit/

[11] Oracle Cooperation, Database Concepts, Abschnitt „End of a Transaction“ http://docs.oracle.com/cd/E29505_01/server.1111/e25789/transact.htm
[12] Oracle Cooperation, SQLPLUS User’s Guide and Reference, http://docs.oracle.com/cd/B19306_01/server.102/b14357/ch4.htm#sthref896

Der einstige Lügner: Expdp
Exporte mit DataPump haben viele Vorteile.

Sie können schnell und handlich einen Teil ihrer Datenbank auch zwischen unterschiedlichen Versionen und Betriebssystemen austauschen.
Sie können nur die Struktur oder auch die Daten exportieren.

Sie können sicher sein, dass Ihre Daten immer konsistent exportiert werden… oder nicht?

Hand aufs Herz: Haben Sie das schon einmal selbst überprüft?

Beispiel

1. Starten Sie einen simplen Export eines Schemas ohne besondere Parameter (directory, logfile, dumpfile, schemas)
2. Löschen Sie Tabellendaten während des Exports, die noch nicht exportiert worden sind und schließen Sie mit commit ab.
3. Vergleichen sie die Anzahl der Exportierten Daten mit Ihrem erwarteten Ergebnis.

Mythos: Data Pump erzeugt immer konsistente Exporte

Wahrheit

Für konsistente Daten wird entweder ein Flashback oder der CONSISTENT Parameter benötigt.
Die Meldung „FLASHBACK automatically enabled“ in Version 10g garantiert keine Konsistenz der Exportdaten.

Ursprung

Falsche Meldung in 10g. Annahme, Konsistenz wird immer gewährleistet:

1. Simple Annahme von konsistenten Exporten ohne Flashback_Time
2. Falsche Ausgabe im Logfile oder auf der Konsole (Version 10.1 und 10.2)
   „FLASHBACK automatically enabled to preserve database integrity"
3. Falsche Aussagen in der Oracle Dokumentation von 10.2[13] und 11.1[14]:
   "A parameter comparable to CONSISTENT is not needed."

Verweise

Oracle Corporation, Expdp Message "FLASHBACK automatically enabled" Does Not Guarantee Export Consistency (Doc ID 377218.1)

Oracle Corporation, Oracle Database Utilities 11.2, Using Original Export Parameters with Data Pump
http://docs.oracle.com/cd/E11882_01/server.112/e22490/dp_legacy.htm#SUTIL961

[13] Oracle Cooperation, Oracle Database Utilities, How Datapump Parameters Map to Those oft he Orignial Export Utility, Version 10.2
http://docs.oracle.com/cd/B19306_01/server.102/b14215/dp_export.htm#sthref181
[14]Oracle Cooperation, Oracle Database Utilities, How Datapump Parameters Map to Those oft he Orignial Export Utility , Version 11.1
http://docs.oracle.com/cd/B28359_01/server.111/b28319/dp_export.htm#i1005864

Mehr Einschränkungen für schnelleres SQL
Performancetuning kann einfach sein. Manchmal reicht ein Blick und der Problemkandidat ist ausgemacht. Eine Select-Abfrage ohne Where Bedingung zählt sicher zu den simpelsten Beispielen.

Haben Sie schon einmal von dem umgekehrten Fall gehört? Dem Mythos, dass eine Abfrage durch zu viele Where-Bedingungen weniger performant wird?

Beispiel

```
create index tab_idx on tb1 (a, date_column);

Select date_column, count(*)
From tb1
Where a=123
Group by date_column;
```

Die Abfrage liefert 100 aus 1.000.000 Zeilen

```
Select date_column, count(*)
From   tb1
Where  a = 123
And    b = 42
Group by date_column;
```

Die Abfrage liefert 10 aus 1.000.000 Zeilen

Wie wirken sich die Kosten/Performance durch Hinzufügen der zweiten Where Bedingung aus?

A) Bleiben sie ca. gleich (+/- 10%)
B) Die Abfrage läuft langsamer (<10%)
C) Die Abfrage läuft schneller (>10%)
D) Es hängt von den Daten ab

Das Beispiel stammt von Marcus Winand.[15]

Mythos: Schnelleres SQL durch einschränkendere Where Klauseln

Wahrheit

Durch weitere Einschränkungen können die Kosten drastisch höher ausfallen.
Im Beispiel muss durch die zusätzliche Einschränkung zusätzlich zum Index auf die Tabelle zugegriffen werden. Der Aufwand durch den Tabellenzugriff steigt erheblich, wie sich durch einen Vergleich der beiden Ausführungspläne einfach zeigen lässt.

```
select date_column,count(*)
from tb1
where a=123
group by date_column;

-------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |   100 |  2200 |     3   (0)|
|   1 |  SORT GROUP BY NOSORT|       |   100 |  2200 |     3   (0)|
|*  2 |   INDEX RANGE SCAN   |TAB_IDX|   100 |  2200 |     3   (0)|
-------------------------------------------------------------------
```

[15] Marcus Winand,Luke Blog, https://use-the-index-luke.com/3-minute-test

```
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("A"=123)

select date_column,count(*)
from tb1
where a=123
and b=42
group by date_column;

-------------------------------------------------------------------------------
| Id  | Operation                    | Name  | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |   126 |  4410 |    28   (0)|
|   1 |  SORT GROUP BY NOSORT        |       |   126 |  4410 |    28   (0)|
|*  2 |   TABLE ACCESS BY INDEX ROWID|TB1    |   126 |  4410 |    28   (0)|
|*  3 |    INDEX RANGE SCAN          |TAB_IDX|    30 |       |     3   (0)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("B"=42)
   3 - access("A"=123)
```

Indexe ohne Balance gehören neu gemacht
Vor vielen Jahren war in alter Oracle Literatur und Oracle Support Dokumente beschrieben, das ein Index Baum durch zu viele ungleich verteilte Insert's seine Blätter (Leaf Blöcke) unterschiedliche Ebenen (Level) aufweisen können.

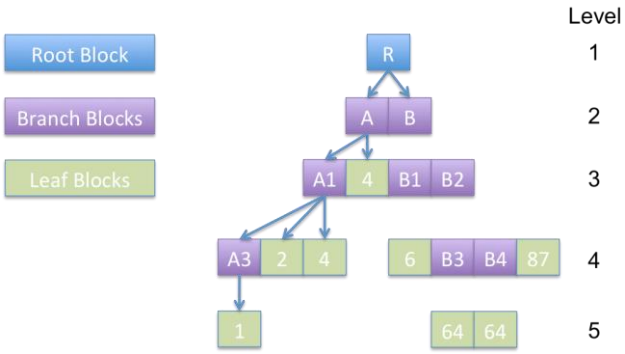


Abbildung 1 - Baum mit Leaf Blocks auf unterschiedlichen Ebenen. Ein solcher Baum kommt bei Oracle B-Tree Indexen nie vor.

Eine tiefere Struktur mit mehr Ebenen führt zu längeren Zugriffspfaden. Längere Zugriffspfade führen zu mehr IO und damit zu mehr Kosten verursacht.

Um diesem entgegenzuwirken, wurde ein Rebuild angeraten, sobald ein Index eine Tiefe von 4 oder mehr aufwies.

Damit sollte die Tiefe des Indexbaumes verringert und auf ein Level gebracht werden.

Mythos: B-Tree Indexe können unbalanciert werden

Ursprung
Aussagen alter Support Dokumente und Literatur
„Hence, an Oracle index may have four levels, but only in those areas of the index tree where the massive inserts have occurred."[16,17]

Beispiel
Wir Erstellen einen Index mit gleichmäßig verteilten Einträgen und erweitern die Einträge, so dass ein Überhang an einer Stelle des Indexbaumes entsteht.
Vor und nach dem Erweitern können wir die Index Struktur überprüfen auf welchem Level sich die jeweiligen Indexblöcke befinden.

Die Baumstruktur eines beliebigen Indexes kann mit einem Treedump betrachtet werden:
alter session set events 'immediate trace name treedump level <object_id>'18

Das entstandene Tracefile zeigt die Baumstruktur eines gewählten Indexes.
Bei jedem B-Tree Index werden die Blätter auf der gleichen Ebene stehen.

Wahrheit
B-Tree steht für Balanced Tree und ist immer balanciert[19]. Dies bedeutet nichts anderes, als das die Blätter eines Indexes (Leaf Blöcke) sich immer auf der gleichen Ebene befinden.

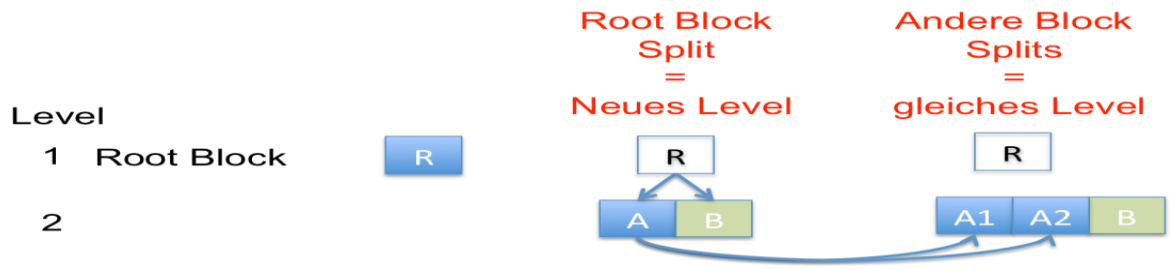
---

[16] http://www.dba-oracle.com/art_index1.htm mit Verweis auf Support Note 77574.1 (derzeit nicht verfügbar)
[17] https://blogs.oracle.com/sysdba/entry/when_to_rebuild_index, Abschnitt Index height
[18] Jonathan Lewis, http://jonathanlewis.wordpress.com/2009/08/17/treedump/
https://richardfoote.wordpress.com/2010/02/08/index-block-dumps-and-treedumps-part-i-knock-on-wood/

Mythos: Rebuilds für B-Tree Indexe mit einer Baumtiefe > 3,4,..

Ursprung
Aussagen alter Support Dokumente und Oracle Literatur.
Häufiger fand sich die Aussage, dass zu viele Level im Index auf ein Strukturproblem hindeutet, welche durch einen Rebuild korrigiert wird.

Wahrheit
Neue Indexblöcke entstehende durch Block Splits, wenn in einem Index Block kein freier Platz mehr zu Verfügung steht.
Leaf und normale Branch Block Splits finden auf dem gleichen Level statt.
Nur ein Split des obersten Root Blocks bewirkt bei B-Tree Indexen eine Vertiefung der Struktur.



Dieser Split tritt wiederum nur bei entsprechend großer Menge an Branch Blocks und Schlussendlich durch das gesamte Anwachsen des Indexes statt.

Die Tiefe hängt damit direkt von der maximal verwendeten Datenmenge im Index ab. Ein Rebuild wird die Indextiefe in der Regel nicht senken.

Beispiel
Statt mit einem synthetisch erstellen Indexe zu argumentieren, lade ich Sie ein:
Betrachten Sie die Index Tiefe bei anstehenden Index Rebuilds, deren Volumen sich nicht wesentlich verändert hat, davor und danach.

Verweise

Oracle Corporation, Support Doc ID 122008.1

[19] Oracle Corporation, B-Tree Indexes, http://docs.oracle.com/cd/E11882_01/server.112/e25789/indexiot.htm#CNCPT1170

Mehr Ordnung für die Index Ordnung

Der Clustering Faktor (CF) ist eine wesentliche Kennzahl für die Entscheidung des Optimizers, ob ein Indexzugriff oder besser ein Full-Table-Scan angewendet werden sollte. Der CF ist ein wesentlicher Faktor für die Kosten beim Indexzugriff.

Indexe sind in der Lage die Ausführungszeiten von Abfragen von Stunden auf Sekunden zu senken. Es ist daher nicht abwegig bei einem schlechten bzw. hohen CF eine Optimierung durchzuführen. Kommt Ihnen an dieser Stelle eine Optimierung des Indexes in den Sinn?

Und tatsächlich wurde vor langer Zeit für die Optimierung des Clustering Faktor ein Rebuild des Indexes vorgeschlagen.

Mythos: Der Clustering Faktor kann durch einen Index Rebuild verbessert werden

Ursprung

In ältern Support Dokumenten und alter Literatur finden sich entsprechende Aussagen.[20]
Die Grundlage dieses Mythos dürfte in der Verwechslung der Abhängigkeitsbeziehung zwischen Tabelle und Index liegen.

Wahrheit

Ein Rebuild eines Indexes ändert seinen Clustering Faktor niemals.

Der Grund liegt darin, dass der Index immer geordnet vorliegt.
Der Clustering Faktor beschreibt die Ordnung der Indexeinträge zu den Einträgen der zugehörigen Tabellenblöcke.
Diese Ordnung ändert sich durch einen Rebuild des Indexes nicht.

Beispiel

Man konstruiert einen Kandidaten für einen Index Rebuild.
Dazu erstellt man einen großen Index, der aus monoton steigenden Werten aufgebaut ist.
Danach entfernt man große Teile aus dem Index ohne freie Blöcke zu erzeugen, beispielsweise in dem nur jeder 10ten Eintrag im Index belassen wird.

Analysiert man vor und nach dem Rebuild den Index, lässt sich feststellen, dass sich der Clustering Faktor nicht verändert.

[20] Oracle Corporation, Support Doc ID 122008.1

Gelöschte Indexeinträge sind tote Einträge
Jeder tote Eintrag belastet den Index doppelt. Er verschwendet Platz und senkt die Performance. Besonders, wenn der gesamte Index gelesen werden muss.

Daher ist es logisch und sinnvoll, wenn der Index möglichst regelmäßig neu aufgebaut wird, um toten Einträgen entgegenzuwirken. Oder nicht?

Mythos: Gelöschte Indexeinträge werden nicht wieder verwendet

Mythos: Ein Index Rebuild ist nach 20% gelöschter Einträge sinnvoll

Ursprung
Der Ursprung könnte von dem Zitat stammen:
„Remember that Oracle leaves "dead" index nodes in the index when rows are deleted“ [21] [22]

Die Aussage lädt dazu ein, falsch Interpretiert zu werden.
Man könnte annehmen, dass ein Eintrag niemals wiederverwendet wird.

Es ist richtig, dass Index Einträge nicht vollständig gelöscht, sondern *zuerst* nur als gelöscht markiert werden. Nachfolgende Operationen können die als gelöscht markierten Einträge jedoch verwenden.

Wahrheit
Gelöschte Einträge innerhalb eines Blocks und vollständig entleerte Blöcke werden wiederverwendet.

Vollständig entleerte Blöcke gelten als freie Blöcke und werden bei neuen Indexsplits verwendet.
Gelöschte Einträge werden wiederverwendet, wenn der neue Eintrag zu dem Block mit den gelöschten Daten passt und genügend Platz für den Eintrag im Block vorhanden ist.

Da gelöschte Einträge wiederverwendet werden können, ist ein genereller Rebuild ab 20% gelöschter Einträge nicht nachvollziehbar!

Beispiele
Wiederverwendung von gelöschten Daten

1. Man erstelle einen Index mit monoton aufsteigenden Zahlen in 99er Schritten und 100.000 Einträgen.
2. Man Entferne jeden 10

Wiederverwendung von gelöschten Indexblöcken

1. Löschen von Einträgen und betrachten der gelöschten Blöcke.
2. Erneutes Einfügen von Daten

Wurde kein Block gelöscht und keiner hinzugefügt, muss der alte Platz wiederverwendet worden sein.

3. Erzeugen eines Indexes, welcher alle Extentblöcke nutzt.

Man konstruiere einen Index, der alle Blöcke seiner Extents nutzt. Würde ein weiterer Block benötigt, so würde der Index um einen weiteren Extent erweitert werden.

Es lassen sich Tabelledaten löschen, so dass ganze Blöcke beim Index frei werden.
Fügt man nach dem Löschen neue Daten hinzu, die von Ihrer Ordnung einen neuen Block benötigen, gibt es zwei Möglichkeiten:

---

[21] https://blogs.oracle.com/sysdba/entry/when_to_rebuild_index
[22] http://www.dba-oracle.com/art_index1.htm

1. Der Index wird um einen neuen Extent erweitert, um die neuen Daten aufzunehmen.
2. Oder es wird kein neuer Extent benötigt. Dann müssen die alten Blöcke in den Extents wiederverwendet worden sein.

Anmerkung:
Es ist für viele einfacher und anschaulicher für Vergleiche ganze Zahlen für die Indexwerte zu verwenden. Beim Typ Number variieren auch größere Zahlen vom Platzbedarf teils erheblich. Der unterschiedliche Platzbedarf erschwert es, allein durch eine bestimmte Anzahl von Zahlen ein Block oder Extent zu füllen.

Bei diesem Beispiel reicht es allerdings zu zeigen, das mindestens ein freigestellter Block wiederverwendet wurde.
Um die Wiederverwendbarkeit von Indexblöcken zu zeigen, kann man Daten von 10 Indexblöcken löschen. Fügt man mindestens Daten in der Größe eines neuen hinzu, kann man die Größe danach betrachten. Ist sie gleichgroß, so muss ein Block wiederverwendet worden sein. Sonst müsste ein neuer Block in einem neuen Extent verwendet worden sein und der Index wäre um einen Extent gewachsen.

Neu ist immer besser
Ein Index Rebuild ist Zeit und Ressourcenaufwändig.
Im Gegenzug erhält man einen neu aufgebauten Index, der keine „Alterserscheinungen“ aufweist.
Warum also keine Index Rebuilds, wenn man Zeit und Ressourcen zur Verfügung hat?

Es schadet doch keinem, nicht wahr?

Mythos: Ein Index Rebuild ist immer gut

Mythos: Indexe sollten regelmäßig neu aufgebaut werden

Ursprung
Annahme, nach einem Rebuild ist immer ein optimaler Zustand erreicht.

Wahrheit
Der Indexzustand kann schlechter sein als zuvor:

- Zusätzliche Aufwände können auch nach einem Rebuild entstehen.
- Der Platzbedarf kann schlechter ausfallen.

Der Platzbedarf kann durch die Standardeinstellung mit PCTFREE 10 schlechter ausfallen, da hierbei 10% des Blocks für spätere Daten freigehalten wird.

Umgekehrt kann zusätzlicher Aufwand nach dem Rebuild entstehen, wenn für viele weitere Daten kein Platz besteht. Dann entstehen während des Betriebes Kosten für Index Splits und zusätzliches Redo. Damit verbunden ergeben sich längere Laufzeiten.

Beispiel
Erstellen Sie einen Index und befüllen Sie Ihn mit monoton aufsteigenden Werten durch Insert Operationen. Ermitteln sie darauf die verwendeten Leaf Blöcke des Indexes.
Führen sie anschließend einen Standard Rebuild ohne weitere Parameterangaben durch.
Abschließend ermitteln Sie erneut die benötigten Index Blöcke.

Das Ergebnis: Mit den Standardeinstellungen beim Indexrebuild werden nach dem Rebuild mehr Leaf Blöcke benötigt.

Wird für den Index ein eigener sich selbst erweiternder Tablespace angelegt, so lässt sich beobachten, dass während eines Offline Rebuild der Platzbedarf bis auf die doppelten Größe des Indexes ansteigt.
Das Ergebnis: Der Tablespace ist angewachsen.

Erstellen Sie eine Tabelle mit einem Index. Befüllen sie diese durch Insert Operationen mit Zufallswerten. Messen sie das Redo aufkommen. Befüllen sie die Tabelle erneut mit Zufallswerten und messen Sie die Zeit. Danach betrachten sie erneut die Menge an Redo.

Wiederhole Sie das Beispiel mit einer neuen Tabelle und führen Sie nach dem ersten Befüllen einen Index Rebuild durch. Vergleichen Sie das Redo und die benötigte Zeit.

Das Ergebnis: Weitere Random Insert Operationen führen zu mehr Index Splits mit vermehrtem Redo und entsprechenden Verzögerungen.

Zusammengefasst: Ein Rebuild kann dazu führen, dass

- mehr Redo benötigt wird
- die nachfolgende Ausführungzeit sich verlängert
- und mehr Speicherplatz allokiert wird.

Resümee

Dies war nur eine Auswahl von vielen Oracle Mythen.
Kannten Sie alle der vorgestellten Mythen? Oder waren einige von Ihnen für Sie neu?
Alle Mythen sind im Internet dokumentiert.
Keine war wirklich neu. Und doch sind sie Vielen von uns unbekannt.
Wie kann das sein?

War es einmal war, was geschrieben stand?
Oder hat jemand etwas falsch verstanden?

Seien sie Neugierig.
Lesen Sie. Stellen Sie in Frage. Probieren Sie es aus!

Kontaktadresse

Robert Baric
ITGAIN Consulting Gesellschaft für IT-Beratung mbH
Essener Straße 1
D-30173 Hannover

Telefon: +49 (0) 511 51513 - 700
Fax: +49 (0) 511 51513 - 800
E-Mail robert.baric(at)itgain.de
Internet: www.itgain.de