

DB-Tuning

Wege zur Performanceoptimierung

Treten bei einer Datenbank Performanceprobleme auf, gibt es mehrere Stellen, an denen Engpässe auftreten können und deshalb Optimierungsmaßnahmen gesetzt werden sollen.

- Konfiguration des DB-Buffers (Cache)
- Optimizer
- Analysedaten
- Suche nach kostenintensiven SQL-Befehlen
- Optimierung der SQL-Befehle mittels Explain Plan
- Hints

DB Buffer Cache

- Schreib- und Lese-Cache für die Nutzdaten der Datenbank.
- Blöcke entsprechen den Blöcken der Files
- Daten werden aus Cache gelesen, sind diese nicht vorhanden, werden die Daten vorher gelesen und im Cache abgelegt
- Trefferquote (Hit Rate) gibt den Prozentsatz an Block-Zugriffen an, die direkt aus dem Cache gelesen werden können (verglichen mit den Gesamtzugriffen)
- Hit Rate sollte im Idealfall mind. 95% sein, daher muss die Buffergröße entsprechend groß gewählt werden. Werte < 90% weisen auf Performanceprobleme hin!

DB Buffer Cache

- Die Größe des Caches wird durch den Initialisierungsparameter `DB_BLOCK_BUFFERS` in Anzahl Blöcken angegeben. Die tatsächliche Größe ist also `DB_BLOCK_BUFFERS * DB_BLOCK_SIZE`. Alternativ wird die Größe über `DB_CACHE_SIZE`-Parameter gesetzt bzw. bei der XE-Edition beispielsweise über `SGA_TARGET`.
- `DB_CACHE_ADVICE`
Bietet eine Schätzung an, wie sich eine Größenänderung des Buffer-Caches auf die Anzahl physischer Zugriffe auswirken würde.

CMD: Parameter lesen/setzen

```
SQL> show parameter db_cache
```

NAME	TYPE	VALUE
db_cache_advice	string	ON
db_cache_size	big integer	52M

```
SQL> show parameter db_block
```

NAME	TYPE	VALUE
db_block_buffers	integer	0
db_block_checking	string	FALSE
db_block_checksum	string	TYPICAL
db_block_size	integer	8192

```
SQL> alter system set db_cache_advice=ON;
```

```
SQL> alter system set db_cache_advice=OFF;
```

```
SQL> alter system set DB_CACHE_SIZE=52M;
```

Cache Hit Ratio ermitteln

- In der View **v\$sysstat** sind diverse statistische Werte abrufbar. Unter anderem auch folgende:
 - **'physical reads'**
Anzahl von Datenblöcken, die seit dem letzten Start der Instanz vom File in den Cache gelesen wurden
 - **'db block gets'**
Anzahl aller Anforderungen für Daten, die direkt aus den Segmenten (ausgenommen Rollback- und Undo-Segmente) bedient wurden
 - **'db consistent gets'**
Anzahl aller Anforderungen für Daten, die aus Gründen der Lesekonsistenz aus Rollback- bzw. Undo-Segmenten bedient wurden

CMD: Cache Hit Ratio ermitteln

```
SQL> select name, value from v$sysstat

SQL> select round(100*(1-(p.value / (b.value + c.value))),2) "Hit Ratio"
  2  from v$sysstat p, v$sysstat b, v$sysstat c
  3  where p.name = 'physical reads'
  4  and b.name = 'db block gets'
  5  and c.name = 'consistent gets';

Hit Ratio
-----
    97,47
```

CMD: Cache Advice nutzen

SQL> select size_for_estimate "BC_Size", size_factor,
 estd_physical_read_factor "PHYS_Factor",
 estd_physical_reads "PHYS_Reads"
 from v\$db_cache_advice;

BC_Size	SIZE_FACTOR	PHYS_Factor	PHYS_Reads
-----	-----	-----	-----
4	,0769	1,5648	43586
8	,1538	1,342	37380
12	,2308	1,2182	33932
16	,3077	1,161	32339
20	,3846	1,1304	31487
24	,4615	1,0974	30566
28	,5385	1,0745	29928
32	,6154	1,052	29303
36	,6923	1,0356	28845
40	,7692	1,0243	28532
44	,8462	1,0148	28265
48	,9231	1,0081	28080
52	1	1	27854
56	1,0769	,9942	27692
60	1,1538	,9875	27506
64	1,2308	,9861	27466
68	1,3077	,9861	27466
72	1,3846	,9651	26881
76	1,4615	,9286	25866
80	1,5385	,8099	22558

Optimizer

Versuchen, einen optimalen Ausführungsplan für das Statement zu ermitteln.

Es können unterschiedliche Ziele für den Optimizer gesetzt werden:

- **'RULE'**
Regelbasiert; festgelegte Prioritätenvergabe
- **'ALL_ROWS'**
Kostenbasiert, mit dem Ziel den besten Durchsatz für das Gesamtergebnis zu erzielen
- **'FIRST_ROWS_n'**
Kostenbasiert, mit dem Ziel schnelle Antwortzeit für die ersten n-Datensätze zu erreichen
(n=1,10,100,1000)

Optimizer

Verschiedene Möglichkeiten, den Optimizer zu setzen:

- Für die Instanz in der init<SID>.ora:
OPTIMIZER_MODE=<mode>
- Für die Session:
ALTER SESSION SET OPTIMIZER_MODE=<mode>;
- Für ein Statement mittels Hints (siehe später):
SELECT /*+<mode>/ ...
wobei mode=FIRST_ROWS(n), ALL_ROWS

Statistics

Damit der kostenbasierte Optimizer gute Ergebnisse bringen kann, braucht er statistische Daten zu den DB-Objekten. Diese werden im Data-Dictionary abgespeichert:

Tabellen	Spalten	Index	System
Zeilenanzahl	Anzahl unterschiedl. Werte	Anzahl Blätter	I/O-Performance
Blockanzahl	Anzahl Nullwerte	Levels	CPU-Performance
Durchschn. Zeilenlänge	Werteverteilung (Histogramm)	Clustering-Faktor	
	Extended Statistics		

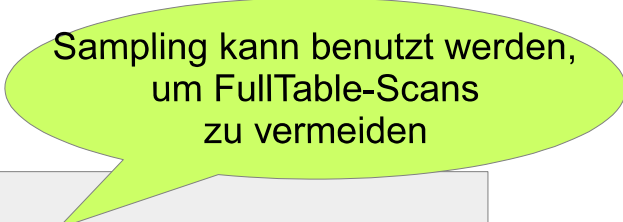
DataDictionary-Tabellen:

DBA_TAB_STATISTICS, DBA_TAB_COL_STATISTICS,
DBA_TAB_HISTOGRAMS, DBA_IND_STATISTICS, ...

Statistics

Die Statistiken werden im Normalfall automatisch von Oracle erstellt und in Wartungsfenstern aktualisiert. Im Bedarfsfall können die Statistiken aber auch auf Anforderung aktualisiert werden. Dazu stehen im Package DBMS_STATS einige Prozeduren zur Verfügung:

```
GATHER_INDEX_STATS,  
GATHER_TABLE_STATS,  
GATHER_SCHEMA_STATS,  
GATHER_DICTIONARY_STATS,  
GATHER_DATABASE_STATS
```



Sampling kann benutzt werden,
um FullTable-Scans
zu vermeiden

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('OE',  
                                         DBMS_STATS.AUTO_SAMPLE_SIZE);  
  
select num_rows, blocks, avg_space, sample_size  
from user_tab_statistics  
where table_name='TEST1';
```

Statistics - Histogramme

Histogramme können auch auf Anforderung neu erstellt werden und die Daten ausgelesen werden:

```
DBMS_STATS.GATHER_table_STATS (OWNNAME => 'OE',
                                TABNAME => 'INVENTORIES',
                                METHOD_OPT => 'FOR COLUMNS SIZE 20 warehouse_id');

SELECT column_name, num_distinct, num_buckets, histogram
FROM   USER_TAB_COL_STATISTICS
WHERE  table_name = 'INVENTORIES'
AND    column_name = 'WAREHOUSE_ID';

SELECT  endpoint_number, endpoint_value
FROM    USER_HISTOGRAMS
WHERE   table_name = 'INVENTORIES'
AND     column_name = 'WAREHOUSE_ID'
ORDER BY endpoint_number;
```

NDV>Buckets ? Height-Balanced : Frequency Histogram

Statistics – Generieren / Löschen

Prüfen auf bestehende Statistik-Daten:

```
SELECT owner, table_name, last_analyzed
FROM    dba_tables                                -- dba_indexes
WHERE   table_name='PRODUCTS'
AND     owner='HTL';
```

Index rebuilden mit Statistikdaten:

```
ALTER INDEX htl.idx_products
REBUILD PARALLEL NOLOGGING COMPUTE STATISTICS;
```

Statistikdaten für alle Indizes einer Tabelle generieren:

```
Execute DBMS_STATS.GATHER_TABLE_STATS (ownname => 'HTL',
    TABNAME           => 'PRODUCTS',
    CASCADE           => TRUE,
    METHOD_OPT         => 'FOR ALL INDEXED COLUMNS',
    ESTIMATE_PERCENT  => DBMS_STATS.AUTO_SAMPLE_SIZE);
```

Statistikdaten löschen:

```
EXECUTE DBMS_STATS.DELETE_TABLE_STATS (
    ownname => 'HTL',
    Tabname => 'PRODUCTS');
```

SQL-Cache

Über die View v\$sql können Details zu gecachten SQL-Statements abgerufen werden. Dazu gehören zB Anzahl der Aufrufe, Anzahl Blockzugriffe, Trefferquote im DB-Buffer-Cache. Somit lassen sich relativ rasch optimierungsbedürftige SQL-Statements rausfinden.

```
select to_char(executions, '999G999G990') "executions",
to_char(buffer_gets, '999G999G990') "gets",
to_char(buffer_gets/greatest(nvl(executions,1),1),
        '999G999G990') "gets je exec",
to_char (round(100*(1-(disk_reads/greatest(nvl(buffer_gets,1),1))),2),
        '990D00') Trefferquote,
sql_text
from v$sql
where buffer_gets > 1000
order by buffer_gets desc;
```

SQL-Tuning / Explain Plan

Der Optimizer erstellt für jede Abfrage einen Ausführungsplan, dessen Generierung von verschiedenen Faktoren wie Optimizer-Modus, Statistikdaten, Hints, etc beeinflusst wird.
Abruf entweder grafisch im SQL-Developer oder via SQL-Statement:

```
delete plan_table;

explain plan
set statement_id = 'MySelect'
for
Select ...; /* Hier das Select Statement angeben */

select lpad(' ',2*level) || OPERATION || ' ' ||
       OPTIONS || ' ' || OBJECT_NAME QUERY_PLAN
from PLAN_TABLE WHERE STATEMENT_ID = 'MySelect'
connect by prior id = PARENT_ID AND STATEMENT_ID = 'MySelect'
start with id = 1;
```


Indizes

Durch den Einsatz von Indizes können kostenintensive Full Table Scans meist vermieden werden.

In Oracle existieren verschiedene Arten von Indizes:

- **B-Tree Index** (Standard):
- **Bitmap Index**
- **Function-Based Index**
- **Partitioned Index**

Da ein Index selbst kostenintensiv im Aufbau bzw. bei der Aktualisierung ist, muss ein Einsatz gut überlegt werden und Kosten/Nutzen in Relation gestellt werden.

B-Tree Index

Ist der Standard-Index.

Für Primärschlüssel und hochselektive Indizes gut geeignet. Auch sehr gut um Daten nach den Indexspalten sortiert zu liefern.

```
Select *
From employees
Where lastname='King'
And firstname='Robert';

CREATE INDEX emp_name_Idx ON Employees(lastname, firstname);
```

Bitmap Index

Sinnvoll bei Daten mit geringer Kardinalität, wobei die Spalten performant mit AND bzw. OR verknüpft werden können. Auch für COUNT() sehr performant.

In Oracle XE leider nicht verfügbar!

```
CREATE BITMAP INDEX cust_bmx ON Customers (gender, status);
```

ROWID	male	female	active	inactive
12345	1	0	1	0
12346	1	0	0	1
12347	0	1	1	0
12348	0	1	0	1

Function Based Index

Es wird ein B-Tree-Index mit den Ergebniswerten einer Funktion aufgebaut.

Beispielsweise sinnvoll, wenn eine Namenssuche nicht case-sensitive sein soll.

```
Select *
From employees
Where upper(lastname)='KING'
And upper(firstname)='ROBERT';

CREATE INDEX emp_uppername_idx ON
  Employees(upper(lastname), upper(firstname));
```

Wann wird ein Index verwendet?

```
Select *  
From Employees  
Where substr(lastname,1,3)  
      ='Kin';
```

```
Select *  
From Employees  
Where lastname like 'Kin%';
```

```
Select *  
From Products  
Where categoryid != 0;
```

```
Select *  
From Products  
Where categoryid > 0;
```

```
Select *  
From Employees  
Where lastname||firstname =  
      'KingRobert';
```

```
Select *  
From Employees  
Where lastname='King'  
And   firstname='Robert';
```

Wann wird ein Index verwendet?

```
Select *  
From Products  
Where unitprice*1.2 > 120;
```

```
Select *  
From Products  
Where unitprice > 100;
```

```
Index1 (Feld1)  
Index2 (Feld2)
```

```
Index (Feld1, Feld2)
```

Hints

Hints erlauben dem Anwender Entscheidungen zu treffen, die normalerweise automatisiert vom Optimizer bestimmt werden.

Syntax:

```
<Statement> /*+ <hint> [text] [<hint>[text]] ... */  
<Statement> --+ <hint> [text] [<hint>[text]] ...
```

Wobei hint das jeweilige Hint-Schlüsselwort ist, text können Parameter für den jeweiligen Hint sein.

Hints: Optimizer

Durch folgende Hints kann der Modus des Optimizers gewählt werden (siehe Folie 9):

- ALL_ROWS
- FIRST_ROWS(n)
- RULE

```
SELECT /*+ ALL_ROWS */ kundenr, nachname
FROM kunde
WHERE kundenr=7566;

SELECT /*+ RULE */ kundenr, nachname
FROM kunde
WHERE kundenr=7566;
```


Hints: Access Path

FULL		
CLUSTER	HASH	} Nur bei Cluster-Objekten
INDEX	NO_INDEX	
INDEX_ASC	INDEX_DESC	
INDEX_COMBINE	INDEX_JOIN	
INDEX_FFS	NO_INDEX_FFS	} FFS=Fast Full Index Scan
INDEX_SS	NO_INDEX_SS	
INDEX_SS_ASC	INDEX_SS_DESC	} SS=Index Skip Scan

Liste und Beschreibungen dieser und vieler weiterer Hints unter:
https://docs.oracle.com/cd/B12037_01/server.101/b10752/hintsref.htm

```
SELECT /*+ NO_INDEX (kunde kunde_nachname_idx) */ *
FROM kunde
WHERE nachname='Gehrke';
```

Hints: Join-Variante

USE_NL	<i>Nested Loop-Join</i> Bei Online-Transaktionssystemen, wo schnellstmöglich Ergebniszeilen geliefert werden sollen. Ergebnismenge sollte kleiner als 10% der Zeilen sein.
USE_HASH	<i>Hash Join</i> Es wird zuerst das Gesamtergebnis ermittelt, dann erst Ergebniszeilen an den Benutzer geliefert.
USE_MERGE	<i>Sort Merge Join</i> Performant, wenn alle Spalten der Join-Klausel durch einen Index vorsortiert sind/werden.

Tuning Tipps für die Datenbank

- Blockgröße mind. (8 KB)
- Auf ausreichend großen Datencache achten (Trefferquote beobachten)
- REDO-LOG-Dateien nicht zu klein wählen (je nach Datenaufkommen ca. 10 bis 200 MB)
- Anzahl Rollback-Segmente ausreichend groß wählen: Anzahl **gleichzeitiger** Transaktionen = Anzahl Rollback-Segmente
- Überflüssige Checkpoints vermeiden:
`Parameter log_checkpoint_interval=1000000`
`Parameter log_checkpoint_timeout=0`
- Indizes auf Tabellen, in denen häufig gelöscht wird, regelmäßig reorganisieren: `alter index <index> rebuild` (möglichst, wenn keine Anwender arbeiten; auf ausreichend TEMP-Platz achten)
- Durch geeignete Storage Parameter die Anzahl der Extents nicht zu groß werden lassen (maximal einige Hundert)

Tuning Tipps für SQL-Statements

- Kostenbasierten Optimizer mit aktuellen Analysedaten einsetzen
- Erstellen von Histogrammen
- bei Verwendung von RULE: driving table berücksichtigen (letzte Tabelle in der from-Klausel)
- bei Verwendung von "or" ggf. RULE einschalten oder "UNION" verwenden
- an häufig benutzten kleinen Tabellen den Parameter "cache" setzen
=> wird bei full table scans nicht aus cache entfernt
(`alter table <tabelle> cache;`)
- auf "vergessene" Indizes achten
- wenig selektive Indizes vermeiden
- bei Performance Messungen caching-Effekt berücksichtigen! Bei der ersten Ausführung eines Befehls werden evtl. Daten über IO geladen. Ab dem zweiten Ausführen des selben Befehls befinden sich diese evtl. schon im Hauptspeicher).

Tuning Tipps für SQL-Statements

- keine Verwendung von Indizes bei:
 - is not null
 - <>
 - like '%abc...'
- besser "exists"-Subquery als "in"-Subquery verwenden

(Quelle: www.datenbank-tuning.de/tipps.htm)