Inhaltsverzeichnis

1. **Der Tuning Prozess**................................................................. 1
   Grundsätzliches

2. **SQL-Optimierung**................................................................. 3
   Grundregeln
   Aufgaben des Optimizers
   2.1 Optimizer Modes ........................................................................................... 4
       Rule Based Optimizer
       COST based Optimizer
   2.2 Zugriffsmethoden .......................................................................................... 7
       Full Table Scan
       Index Scan
   2.3 Indexierungs Strategie ................................................................................ 13
       Regeln
       Selektivität
       B*TREE Index Analyse
   2.4 Datenverteilung............................................................................................ 15
   2.5 Join Algorithmen ......................................................................................... 16
       Sort-Merge Join
       Hash-Join
       Nested Loop
       Star Joins
       Beurteilung
   2.6 Execution Plan ............................................................................................ 20
       AutoTrace in SQL*Plus
       TopSession im Enterprise Manager
       TKPROF
   2.7 Optimierung................................................................................................. 26
       Hints
   2.8 Regeln für gutes SQL.................................................................................. 28
       Allgemeines
       Outer Joins
       DISTINCT
       NOT IN
       Inline Functions

3. **Tuning ausgewählter SQL-Statements**............................................. 30
   3.1 COST- oder RULE-based Optimizer ? ........................................................ 30
       COST-based
       RULE-based
       Index-Zugriff
   3.2 Join-Methoden (Algorithmen) ...................................................................... 32
       Join-Methoden
   3.3 Ein oder mehrere Attribute in der Selektion ................................................ 36
       SELECT * ... 
   3.4 Subquery Optimierung ................................................................................ 38
       Variante 1
       Variante 2
       Variante 3
       Variante 4
   3.5 Zählen von Rows ...................................................................................... 40
       COUNT
   3.6 SQL-Funktionen die Indexzugriff verhindern............................................... 41
4. Datenbank Tuning .......................................................... 42

Übersicht

4.1 Oracle Performance Pack .......................................................... 42
  Overview Window
  Buffer Cache Hit %
  Library Cache Hit %
  Data Dictionary Cache Hit %
  Memory Sort Hit %
  Rollback Nowait %
  Rollback Segment Waits
  Rollback Segments Shrinks
  File I/O Rate
  System I/O Rate
  Throughput
  Active Users
  Circuit
  Dispatcher
  MTS
  File I/O Rate Details
  Free List Hit%
  Latch
  Library Cache Details
  Lock
  Memory Allocated
  Network Bytes Rate
  Network I/O Rate
  Session Events
  Parse Ratio
  Processes
  Queue
  Redo Alloc Hit%
  Redo Statistics
  Session
  Sort Rows Rate
  Sqlarea
  System Statistics
  Table Access
  Users logged on
  Users running
  Users waiting
  Users waiting for Locks
  Object Status

4.2 Detaillierte Analyse des Shared Pools .................................... 65
  Memory Fragmentierung
  Vorhandene Memory Segmente
  Grosse Objecte lokalisieren
  Grosse Anonymous PL/SQL Blocks
  Grösse des Shared Pools bestimmen
  Grösse des Reserved Pools bestimmen
  Überwachung des Shared Pool
  Output der Überwachung

5. Locking Verhalten .............................................................. 81

5.1 Row Locks ...................................................................... 81
  Row Share Table Locks (RS)
  Row Exclusive Table Locks (RX)

5.2 Table Locks ...................................................................... 82

5.3 Lock Manager Lock Types .............................................. 82
1. Der Tuning Prozess

Grundsätzliches

Es gibt wenig allgemein gültige Rezepte, da beispielsweise ein Tuning für ein Transaktionssystem (OLTP) anders vorgenommen werden muss als ein Tuning für ein DataWareHouse (DSS).

Ein «seriöses» Tuning muss sich über einen größeren Zeitraum erstrecken, Momentaufnahmen bilden die Mosaiksteine zum gesamten Tuning-Prozess.

Tuning beginnt grundsätzlich bereits beim Design und begleitet anschliessend die Applikation auf ihrem gesamten Lifecycle.

Ein Tuning beinhaltet immer mehrere Komponenten, die nur in ihrem Gesamtzusammenhang den angestrebten Nutzen bringen.

Datenmodell

- Keine oder übertriebene Normalisierung.
- Denormalisierung gezielt einsetzen.
- Gewährleistung der Integrität (Jede Tabelle hat einen Primary Key, referentielle Integritätskontrolle durch Foreign Keys, UNIQUE Attribute, CHECK Constraints, Datenbank-Trigger, etc).
- Nachführen von aufwendigen Berechnungen und Summenbildungen (vermeiden von SELECT COUNT(*) für Statistiken).

Korrekte Indexierung

- Indizes nur dort wo sie «wirklich» einen Nutzen bringen.
- Vermeidung von Locking-Problemen bei gleichzeitigem Mehrfachzugriff auf Master-Detail Tabellen.
- Bitmapped Indizes für wenig selektive Attribute verwenden.
- Wenn nötig Index-Histogramme erstellen.
Oracle Datenbank Tuning

Tools

- Möglichst moderne Tools einsetzen, die das Tuning unterstützen.
  Datenmodell: Designer/2000, S-Designer
  Physische Implementation: Designer/2000, PowerBuilder
  Client / Server Tools: PowerBuilder, Delphi, Visual C++, etc
  Netzwerk: Möglichst direkt über SQL*Net ohne Umweg über ODBC.
  Stored Procedures: SQL-Navigator

- Nicht am Tool «vorbei programmieren», das heisst, dass man die Eigenschaften des Tools nutzen sollte und nicht einen Lösungsweg anstrebt, für den das Tool nicht geeignet ist.

Physische Implementierung

- Tablespace Organisation, meist sind Index- und Tabellendaten getrennt.
- I/O-Tuning und Verteilung der Datenbank-Files auf mehrere Disks. Dabei ist Rücksicht zu nehmen auf RAID-Systeme. Grundsätzlich empfiehlt Oracle RAID 0 und RAID 1, aber nicht RAID 5.
- RAW-Devices einsetzen (möglichst nicht mit normalen Filesystem mischen).
- Speicherstrukturen wie SGA, Shared Pool optimieren (DB_BUFFER, SHARED_POOL_SIZE, SHARED_POOL_RESERVED_SIZE, SHARED_POOL_RESERVED_MIN_ALLOC).
- Inde x überwachen und gegebenfalls neu erstellen.

DB-Tuning

- INIT.ORA Parameter optimieren
- Behebung von Engpässen aufgrund gleichzeitiger Beanspruchung der Datenbank durch viele Prozesse (Free Lists, Rollback Segmente, Latches etc).
- Trennung von DSS und OLTP Datenbanken mittels Replikation.

SQL-Optimierung

- Richtiger Einsatz des Optimizers. Der COST-based Optimizer verlangt Statistiken, die aufgebaut werden müssen.
- Index-Scan oder Full-Table Zugriff ?
- Optimizer Hints einsetzen.
- Alternativen für ein bestimmtes SQL-Statement suchen, testen.
- Transaktionskontrolle ist Sache des Programmierers. Transaktionen so kurz wie möglich halten, Exceptions, Savepoints etc einbauen.
- Kein explizites Locking programmieren, SELECT FOR UPDATE nur wo dies wirklich begründbar ist. Möglichst nie ein exclusives Table Locking einsetzen. Grundsätzlich sollte das Lockingverhalten Oracle überlassen werden.

Betriebssystem Optimierung

- Oracle verlangt genügend Memory für die gesamte SGA. Die SGA MUSS unbedingt im Memory Platz finden. (Swapping, Paging der SGA muss unbedingt verhindert werden).
- Auslastung überwachen (TOP, SAR, etc).

2. SQL-Optimierung

Grundregeln

- Nur 1 - 2 % der SQL Befehle dürfen Performance Probleme aufweisen, ansonsten muss in einem anderen Bereich gesucht werden (Datenmodell, Hardware, physische Implementation).
- Restriktionen frühzeitig erkennen, dies erfolgt immer in der WHERE Klausel. Der EQUI-Join verhindert nur das Kartesische Produkt.
- Regeln einhalten, um entweder Fulltable-Scan oder Index-Zugriff zu ermöglichen.
- Die Antwortzeit ist direkt von der Anzahl Rows des Resultats und nicht von der Anzahl Rows in der Tabelle abhängig.
- Interaktive Abfragen sollten immer nur eine kleine Menge an Treffern haben. Dies ist besonders wichtig im Client / Server Bereich.
- Bei Batchverarbeitung ist die gesamte Antwortzeit entscheidend, die Zeit bis der erste Record gefunden wird ist unwichtig.

Aufgaben des Optimizers

Der Optimizer versucht über eine frühzeitige Restriktion die Resultats-Menge so klein wie möglich zu halten.

- Auf Tabellen Ebene geschieht eine Einschränkung nur dann, wenn nicht mehr die gesamte Tabelle gelesen werden muss. Dazu braucht es eine WHERE-Klausel, welche auf indexierte Attribute appliziert werden kann.
- Die Aufgabe des Optimizer besteht also darin, aufgrund der WHERE Klausel die kleinste Menge an Treffern zu finden, welche nicht für einen Join verwendet werden. Diese gefundene Menge wird nun mit der Menge an Treffern aus der anderen Tabelle gejoint.

Um diese Aufgabe so effizient wie möglich zu erfüllen, erstellt der Optimizer einen Execution Plan, der kontrolliert und beeinflusst werden kann. Als Grundregel kann gesagt werden, dass selektive Abfragen mit einem kleinen Result-Set über einen Index-Scan gelesen werden sollten. Wird dagegen die gesamte Tabelle gelesen (keine WHERE-Klausel) so ist ein Fulltable-Scan deutlich effizienter. Wo die Grenze liegt, kann nicht genau spezifiziert werden, da auch die Selektivität der Daten eine wichtige Rolle spielt. Als ersten Ansatz kann von einer Annahme von 15 % ausgegangen werden.
2.1 Optimizer Modes

Oracle 7 bietet zwei verschiedene Optimizer an:

- Rule Based Optimizer (RBO)
- Cost Based Optimizer (CBO)

Rule Based Optimizer

Der Rule Based Optimizer entscheidet aufgrund des zur Verfügung stehenden Access Path und des zugehörigen Ranks nach einem fix vorgegebenen Schema.

Vorteile des Rule Based Optimizers

- Bei Migrationen älterer Applikationen (vorallem V6 -> V7)
- Immer dann, wenn ein Fulltable-Scan möglichst verhindert werden soll.
- Der Parse Overhead ist sehr gering.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Access Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Single Row by ROWID</td>
</tr>
<tr>
<td></td>
<td>Beispiel: empno ist der Primary Key</td>
</tr>
<tr>
<td></td>
<td>SELECT rowid INTO var_rowid</td>
</tr>
<tr>
<td></td>
<td>FROM emp</td>
</tr>
<tr>
<td></td>
<td>WHERE empno = 4711</td>
</tr>
<tr>
<td></td>
<td>FOR UPDATE;</td>
</tr>
<tr>
<td></td>
<td>UPDATE emp SET ename = 'MILLER'</td>
</tr>
<tr>
<td></td>
<td>WHERE rowid = var_rowid;</td>
</tr>
<tr>
<td>2</td>
<td>Single Row by Cluster Join</td>
</tr>
<tr>
<td>3</td>
<td>Single Row by Hash Cluster Key with Unique or Primary Key</td>
</tr>
<tr>
<td>4</td>
<td>Single Row by Unique or Primary Key</td>
</tr>
<tr>
<td></td>
<td>Beispiel: empno ist der Primary Key</td>
</tr>
<tr>
<td></td>
<td>SELECT *</td>
</tr>
<tr>
<td></td>
<td>FROM emp</td>
</tr>
<tr>
<td></td>
<td>WHERE empno = 7900</td>
</tr>
<tr>
<td>5</td>
<td>Clustered Join</td>
</tr>
<tr>
<td>6</td>
<td>Hash Cluster Key</td>
</tr>
<tr>
<td>7</td>
<td>Index Cluster Key</td>
</tr>
<tr>
<td>8</td>
<td>Composite Index</td>
</tr>
<tr>
<td></td>
<td>Beispiel: Composite Index auf (job, deptno)</td>
</tr>
<tr>
<td></td>
<td>SELECT *</td>
</tr>
<tr>
<td></td>
<td>FROM emp</td>
</tr>
<tr>
<td></td>
<td>WHERE job = 'CLERK'</td>
</tr>
<tr>
<td></td>
<td>AND deptno = 30;</td>
</tr>
<tr>
<td>9</td>
<td>Single Column Indexes</td>
</tr>
<tr>
<td></td>
<td>Beispiel: Index auf job</td>
</tr>
<tr>
<td></td>
<td>SELECT *</td>
</tr>
<tr>
<td></td>
<td>FROM emp</td>
</tr>
<tr>
<td></td>
<td>WHERE job = 'CLERK';</td>
</tr>
<tr>
<td>10</td>
<td>Bounded Range Search on Indexed Columns</td>
</tr>
<tr>
<td></td>
<td>Beispiel: column ist an 1. Stelle in einem Composite Index</td>
</tr>
<tr>
<td></td>
<td>WHERE column = expr</td>
</tr>
<tr>
<td></td>
<td>WHERE column &gt;= expr AND column &lt;= expr</td>
</tr>
<tr>
<td></td>
<td>WHERE column BETWEEN expr AND expr</td>
</tr>
<tr>
<td></td>
<td>WHERE column LIKE 'ABC%'</td>
</tr>
</tbody>
</table>
Der Rule Based Optimizer versucht also mit allen Mitteln einen Fulltable-Scan zu verhindern. Oft wird jedoch die gesamte Tabelle gelesen, wo ein Fulltable-Scan die effizienteste Methode darstellt.

**Beachte**

Bei Performance Problemen kann zuerst der andere Optimizer zugezogen werden, anschliessend werden die Execution Plans miteinander verglichen.

**Übersteuern des Optimizer Mode auf Session Ebene:**

```
ALTER SESSION SET OPTIMIZER_GOAL = RULE;
ALTER SESSION SET OPTIMIZER_GOAL = CHOOSE;
```

**Übersteuern des Optimizer Mode auf Befehls Ebene:**

```
SELECT /*+ RULE */ ename FROM .....  
SELECT /*+ CHOOSE */ ename FROM ....
```
**Der COST based Optimizer**

Der COST based Optimizer legt den Execution Plan aufgrund von Statistiken fest. Diese Statistiken müssen regelmässig erstellt werden (Vorteilhaft während der Nacht mittels ANALYZE ... Kommando).

**Vorteile des COST Based Optimizers**

- Der CBO erkennt unnötige Abfragen, wie sie oft von Endanwender Tools generiert werden.
- Umschaltung von «LIKE» auf «==» wenn Wildcard Zeichen fehlen.
- Verwendung von neuen Features wie Hash-Join, Anti-Join etc.
- Entscheidet sich nicht «stur» für einen Indexzugriff wie dies der RULE based Optimizer oft tut.

**Nachteile des COST Based Optimizers**

- Schätzt die Selektivität der Daten manchmal falsch ein, was dann fast immer zu einem Fulltable-Scan führt.
- Der CBO darf erst wirklich produktiv ab Version 7.3 eingesetzt werden.
- Der Parse Aufwand ist grösser als beim RULE based Optimizer.

**Entscheidungsgrundlagen**

Der COST based Optimizer benutzt folgende Entscheidungsgrundlagen:

- Statistiken werden im Data Dictionary abgelegt. Sie können eingesehen werden in den Views USER_TABLES und USER_TAB_COLUMNS.
- Berücksichtigung der Selektivität, wenn ein Histogramm zur Verfügung steht. Die Selektivität ist der Anteil an Rows in Prozent, die das Query selektiert. Queries, welche wenig Rows selektieren, haben eine gute Selektivität, sie eignen sich für einen Index-Scan.
- Der INIT.ORA Parameter DB_FILE_MULTIBLOCK_READ_COUNT wird berücksichtigt. Hohe Werte verleiten den COST Based Optimizer zu einem Fulltable-Scan.
### 2.2 Zugriffsmethoden

**Full Table Scan**

Wenn sich der Optimizer für einen Full Table Scan entscheidet, so wird bei dieser Zugriffsmethode die gesamte Tabelle sequentiell gelesen. Full Table Scans sind vorteilhaft, wenn eine große Datenmenge als Ergebnis gesucht wird.

Als **Richtwerte** können folgende Aussagen gemacht werden:

- Datenmengen > 15 % werden in der Regel effizienter mit einem Full Table Scan gelesen. Die genaue Grenze ist jedoch abhängig vom Clustering Factor (Datenverteilung), der Datendichte pro Block, den Anzahl Attributen etc.
- Oracle hat einen sehr effizienten Mechanismus für Full Table Scans. Es wird ein sogenannter Multi-Block-Read gelesen, welcher pro Disk Operation mehrere Blocks lesen kann.
- Der Durchsatz des Multi-Block-Reads wird mit dem INIT.ORA Parameter DB_FILE_MULTIBLOCK_READ_COUNT definiert.
- Einerseits sollten möglichst viele Blocks gelesen werden (I/O wird kleiner), andererseits sollten auch nicht zuviele Blocks angefordert werden, so dass während der Wartezeit bis die Blocks gefunden werden, die CPU arbeitslos ist.
- Einen zu hohen DB_FILE_MULTIBLOCK_READ_COUNT verleitet den COST based Optimizer zu Full Table Scans. Deshalb muss dieser Wert sorgfältig evaluiert werden.

**Kontrolle von DB_FILE_MULTIBLOCK_READ_COUNT**

```
SELECT NAME "File", PHYBLKWRIT WRITTEN, PHYBLKRD READ,
       ROUND(PHYBLKRD/DECODE(PHYRDS,0,1,PHYRDS),2) "Ratio"
FROM V$DBFILE A, V$FILESTAT B
WHERE A.FILE# = B.FILE#
ORDER BY SUBSTR(NAME,1,INSTR(NAME,'.'));
```

<table>
<thead>
<tr>
<th>File</th>
<th>WRITTEN</th>
<th>READ</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sipp/d1/ts_system/P</td>
<td>87378</td>
<td>86262</td>
<td>1,91</td>
</tr>
<tr>
<td>/dev/vg11/rsipp01</td>
<td>1500215</td>
<td>2651</td>
<td>1</td>
</tr>
<tr>
<td>/dev/vg11/rsipp02</td>
<td>2593390</td>
<td>1892911</td>
<td>4,65</td>
</tr>
<tr>
<td>/dev/vg11/rsipp08</td>
<td>6638830</td>
<td>6007879</td>
<td>1,57</td>
</tr>
<tr>
<td>/sipp/d1/ts_system/P</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/dev/vg11/rsipp03</td>
<td>3199930</td>
<td>16049439</td>
<td>6,42</td>
</tr>
<tr>
<td>/dev/vg11/rsipp04</td>
<td>189468</td>
<td>23607191</td>
<td>15,02</td>
</tr>
<tr>
<td>/dev/vg11/rsipp05</td>
<td>60560</td>
<td>8838645</td>
<td>8,37</td>
</tr>
<tr>
<td>/dev/vg11/rsipp06</td>
<td>504939</td>
<td>637395</td>
<td>1,03</td>
</tr>
<tr>
<td>/dev/vg11/rsipp07</td>
<td>3132762</td>
<td>1838936</td>
<td>1,25</td>
</tr>
<tr>
<td>/dev/vg11/rsipp09</td>
<td>27764</td>
<td>1210526</td>
<td>13,77</td>
</tr>
<tr>
<td>/dev/vg11/rsipp10</td>
<td>9641</td>
<td>420278</td>
<td>13,9</td>
</tr>
</tbody>
</table>
Zuordnung DB-File und Tablespace

<table>
<thead>
<tr>
<th>Name</th>
<th>Tablespace</th>
<th>Status</th>
<th>Size (M)</th>
<th>Used (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/vg11/rsipp01</td>
<td>RB</td>
<td>ONLINE</td>
<td>990.000</td>
<td>52.255</td>
</tr>
<tr>
<td>/dev/vg11/rsipp02</td>
<td>TEMP_PROC</td>
<td>ONLINE</td>
<td>990.000</td>
<td>368.123</td>
</tr>
<tr>
<td>/dev/vg11/rsipp03</td>
<td>TAB_SIPP</td>
<td>ONLINE</td>
<td>990.000</td>
<td>51.477</td>
</tr>
<tr>
<td>/dev/vg11/rsipp04</td>
<td>TAB_CDR</td>
<td>ONLINE</td>
<td>990.000</td>
<td>257.051</td>
</tr>
<tr>
<td>/dev/vg11/rsipp05</td>
<td>TAB_CREDIT</td>
<td>ONLINE</td>
<td>990.000</td>
<td>360.034</td>
</tr>
<tr>
<td>/dev/vg11/rsipp06</td>
<td>IDX_SIPP</td>
<td>ONLINE</td>
<td>990.000</td>
<td>254.031</td>
</tr>
<tr>
<td>/dev/vg11/rsipp07</td>
<td>IDX_BIG</td>
<td>ONLINE</td>
<td>990.000</td>
<td>181.482</td>
</tr>
<tr>
<td>/dev/vg11/rsipp08</td>
<td>TEMP_CLINT</td>
<td>ONLINE</td>
<td>990.000</td>
<td>381.523</td>
</tr>
<tr>
<td>/dev/vg11/rsipp09</td>
<td>TAB_CDR</td>
<td>ONLINE</td>
<td>990.000</td>
<td>139.958</td>
</tr>
<tr>
<td>/dev/vg11/rsipp10</td>
<td>TAB_CDR</td>
<td>ONLINE</td>
<td>990.000</td>
<td>165.556</td>
</tr>
<tr>
<td>/sipp/d1As_system/F16M7S_sys1.dbf</td>
<td>SYSTEM</td>
<td>SYSTEM</td>
<td>80.000</td>
<td>15.451</td>
</tr>
<tr>
<td>/sipp/d1As_system/F16M7S_users1.dbf</td>
<td>USERS</td>
<td>ONLINE</td>
<td>10.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>


Index Scan

Bei einem Index Scan werden die Daten aufgrund der WHERE Klausel zuerst im Index lokalisiert, dort wird die ROWID des Treffers bestimmt und dann der entsprechende Datenblock direkt gelesen. Die Adresse des Datenblocks ist in der ROWID enthalten.

Jede Row besitzt eine eindeutige physikalische Adresse, welche durch ROWID repräsentiert wird. ROWID ist aus drei Teilen aufgebaut:

Aufbau ROWID

<table>
<thead>
<tr>
<th>Nr des Blocks im File</th>
<th>Nr der Row im Block</th>
<th>Nr des Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT rowid, ename FROM emp;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ROWID ENAME
--------------------------
000005DF.0000.0004 SMITH
000005DF.0001.0004 ALLEN
000005DF.0002.0004 WARD
000005DF.0003.0004 JONES
000005DF.0004.0004 MARTIN
000005DF.0005.0004 BLAKE
000005DF.0006.0004 CLARK
000005DF.0007.0004 SCOTT
000005DF.0008.0004 KING
000005DF.0009.0004 TURNER
000005DF.000A.0004 ADAMS

Alle Rows sind im gleichen File
Nummer der Row im Block
Alle Rows sind im gleichen Block, kein «Block Chaining»

Oracle kennt zwei Arten von Indexen, den B*TREE Index und den Bitmap-Index.
**B*TREE Index**

B*TREE Indexe sind nach dem Datenwert sortierte baumförmige Indizes. Man spricht auch von einem sorted Index. B-Trees bestehen aus Index- und Sequenzblöcken. In den Sequenzblöcken werden die ROWID’s der Records gespeichert:

**Unique und Non-unique Indexes**

Wird eine Column mit einem Unique Index versehen, dann muß jeder Wert der Column eindeutig sein. Dies trifft bei Primary Keys zu, auf diese wird immer ein Unique Index angelegt, was unter Oracle7 automatisch erfolgt. Doch auch nicht eindeutige Columns können indexiert werden, indem beim Kreieren des Index das Schlüsselwort «unique» angegeben wird. Das Anlegen von Indexes auf non-unique Columns ist jedoch nur dann sinnvoll, wenn der Wertebereich groß ist. Eine Column, die z.B. nur die Werte «männlich», «weiblich» beinhalten kann, ist denkbar ungeeignet zum indexieren!

**Concatenated Index**

Aus mehreren Columns bestehende Indextre nach dem Datenwert sortierte baumförmige Indizes. Man spricht auch von einem sorted Index. B-Trees bestehen aus Index- und Sequenzblöcken. In den Sequenzblöcken werden die ROWID’s der Records gespeichert:

**Unique und Non-unique Indexes**

Wird eine Column mit einem Unique Index versehen, dann muß jeder Wert der Column eindeutig sein. Dies trifft bei Primary Keys zu, auf diese wird immer ein Unique Index angelegt, was unter Oracle7 automatisch erfolgt. Doch auch nicht eindeutige Columns können indexiert werden, indem beim Kreieren des Index das Schlüsselwort «unique» angegeben wird. Das Anlegen von Indexes auf non-unique Columns ist jedoch nur dann sinnvoll, wenn der Wertebereich groß ist. Eine Column, die z.B. nur die Werte «männlich», «weiblich» beinhalten kann, ist denkbar ungeeignet zum indexieren!

**Concatenated Index**

Aus mehreren Columns bestehende Indexe werden «Concatenated Index» genannt. Sie sind sehr ideal, wenn Eindeutigkeit über mehrere Columns gefordert ist.

<table>
<thead>
<tr>
<th>Vorwahl</th>
<th>Telefon</th>
<th>PLZ</th>
<th>Ort</th>
<th>Straße</th>
</tr>
</thead>
<tbody>
<tr>
<td>033</td>
<td>345 02 40</td>
<td>3628</td>
<td>Seftigen</td>
<td>Sonnenrain 5</td>
</tr>
<tr>
<td>031</td>
<td>385 30 11</td>
<td>3000</td>
<td>Bern</td>
<td>Zieglerstr. 34</td>
</tr>
</tbody>
</table>

Die Columns «Tel.-Vorwahl» und «Telefon» sind zusammen unique und deshalb für einen concatenated Index geeignet.
Verwendung des Index

B*TREE Indexes erlauben eine effiziente Suche, wenn die indexierten Daten selektiv sind. Die selektivste Indexierung ist die Sonderform eines Unique Index, weil es dort genau 1 einzige Row pro Datenwert gibt.

Ein B*TREE Index kann in den folgenden Fällen nicht verwendet werden:

- NULL Werte sind nicht im Index gespeichert und können nur in der Tabelle mittels Full Table Scan gefunden werden.

  ... WHERE column IS NULL;  (Index)
  ... WHERE column IS NOT NULL;  (Index)

- Alle Funktionen direkt auf der Column verhindern den Indexzugriff

  ... WHERE SUBSTR(ename, 1, 7) = 'CAPITAL';  (Index)
  ... WHERE ename LIKE 'CAPITAL';  (Index OK)
  ... WHERE TRUNC(edeate) = TRUNC(sysdate)  (Index)
  ... WHERE edate BETWEEN TRUNC(sysdate) 
       AND TRUNC(sysdate) + .99999;  (Index OK)

- Ungleich Operator verhindert Indexzugriff

  ... WHERE amount != Wildcard;  (Index)
  ... WHERE amount > 0;  (Index OK)

- Berechnungen verhindern den Indexzugriff

  ... WHERE amount + 3000 < 5000;  (Index)
  ... WHERE amount < 2000;  (Index OK)

- Wildcard % auf erstem Zeichen verhindert Indexzugriff

  ... WHERE ename LIKE '%ueller';  (Index)

- Gleiche Columns auf beiden Seiten verhindern Indexzugriff

  ... WHERE ename = NVL(var, ename);  (Index)
  ... WHERE ename LIKE NVL(var, '%');  (Index OK)

- String Concatenierung verhindert den Indexzugriff

  ... WHERE ename || vname = 'XYZ';  (Index)
  ... WHERE ename = 'XY' AND vname = 'Z';  (Index OK)

- Mehr als ein Index (auf ename und deptno) pro Tabelle kann nur verwendet werden, wenn alles Equality-Searches mit NON-UNIQUE Indexes sind. Hier führt Oracle einen Index-Merge durch.

  ... WHERE ename LIKE 'XY%' AND deptno = 10;  (ein Index)
  ... WHERE ename = 'XY' AND deptno = 10;  (beide Indexe)
Oracle erlaubt einen Index über mehrere Attribute in der gleichen Tabelle zu erstellen. Ein solcher concatenated Index eröffnet dem Optimizer ideale Zugriffe, die von zwei einzelnen Indexes nicht zu lösen sind. Im folgenden Beispiel ist ein Index über \((deptno, ename)\) vorhanden.

```
SELECT ename, sal, deptno
FROM emp
WHERE ename LIKE 'KIN%'
AND deptno = 10;
```
(Index OK)

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno = 10;
```
(Index OK)

```
SELECT ename, sal, deptno
FROM emp
WHERE ename LIKE 'KIN%';
```
(Index)

Im dritten Beispiel kann der Index nicht benutzt werden, da das Attribut \(ename\) im Index an zweiter Stelle steht.

- **NOT** verhindert den Indexzugriff

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno NOT = 0
```
(Index)

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno > 0
```
(Index OK)

- Manchmal will man einen Indexzugriff explizit verhindern. Dies wird mit einer Addition von 0 oder einer Concatenierung eines Leerstrings erreicht.

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno + 0 = 10
```
(Index)

```
SELECT ename, sal, deptno
FROM emp
WHERE ename || '' = 'MU'
```
(Index)
Bitmapped Index

Ab Oracle Version 7.3 stehen auch Bitmapped Indexe zur Verfügung. Sie eignen sich hervorragend, um unselektive Attribute zu indexieren, das heisst es gibt nur wenige DISTINCT Werte.

Eigenschaften von Bitmapped Indexen

- Nur der neue Optimizer CBO kann Bitmapped Indexe verwenden.
- Verknüpfte Bitmapped Indexes sind möglich.
- NULL Werte können ebenfalls im Bitmapped Index gefunden werden. Das heisst also, das Abfragen auf NULL und IS NULL möglich sind.
- Bitmapped Indexte können auch != Abfragen lösen.
- Bitmapped Indexe eignen sich für SELECT COUNT(*) Abfragen.
- Das Nachführen von Bitmapped Indexen ist aufwendig.
- Das Locking eines Bit ist nicht möglich, deshalb lockt Oracle einen gesamten Index-Block, was zu Locking Problemen führen kann. Bitmapped Indexe sollten also in OLTP Datenbanken mit gleichzeitigen DML Befehlen auf der gleichen Tabelle nicht eingesetzt werden. In DataWareHouse Anwendungen bieten sie sehr grosse Vorteile.

Beispiel: Abfragen auf männlich, weiblich sind sehr unselektiv

<table>
<thead>
<tr>
<th>ROWID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Einsatz von Bitmapped Indexen

Bitmapped Indexe eignen sich gut für nicht selektive Attribute. Man muss also zuerst die Selektivität feststellen.

```
SELECT status, count(*) FROM credit
GROUP BY status;
```

<table>
<thead>
<tr>
<th>STATUS</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>176858</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>373</td>
</tr>
</tbody>
</table>

In diesem Fall haben wir lediglich 3 verschiedene Werte, ein Bitmapped Index könnte in Betracht gezogen werden.

```
SELECT lname, count(*) FROM customer
GROUP BY lname;
```

<table>
<thead>
<tr>
<th>LNAME</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST Communication</td>
<td>3</td>
</tr>
<tr>
<td>Abbet</td>
<td>1</td>
</tr>
<tr>
<td>Abbondioli</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

In diesem Fall kommt ein Bitmapped Index nicht in Frage.
2.3 Indexierungs Strategie

Regeln

- Grundsätzlich werden alle Primary Key und UNIQUE Keys mittels Constraint indexiert.
- Fremdschlüssel werden meistens indexiert wenn eine Referential Integrity implementiert ist. Fremdschlüssel von Definitionstabellen (Lookup-Tables) müssen nicht indexiert werden, da auf diesen Tabellen keine DDL-Kommandos (UPDATE, INSERT, DELETE) durchgeführt werden und somit kein Locking eintreten kann.
- Intersection Tabellen werden meistens mit 2 Indexen in beiden Richtungen indexiert.
- Alle Suchfelder (WHERE Klausel), welche eine gute Selektivität aufweisen, werden indexiert. Suchfelder müssen bekannt sein (Oracle Expert, Entwickler Interviews etc).
- Suchfelder, welche eine gute Selektivität aufweisen, jedoch unregelmäßig verteilte Datenwerte haben, werden zusätzlich mit einem Histogramm versehen.
- Auf jeder Tabelle muss kontrolliert werden, dass nicht zwei Indexe mit dem gleichen Attribut beginnen.
- Bestehende Indexe müssen analysiert werden (ANALYZE INDEX).

Selektivität

Die Selektivität kann wie folgt festgestellt werden (Beispiel):

```
SELECT SUBSTR(TO_CHAR(date_cdr1,'DDMMYYYY'),1,8) "Wert", count(*) "Anzahl"
FROM acm
GROUP BY TO_CHAR(date_cdr1,'DDMMYYYY')
ORDER BY 2 desc;
```

<table>
<thead>
<tr>
<th>Wert</th>
<th>Anzahl</th>
</tr>
</thead>
<tbody>
<tr>
<td>23051997</td>
<td>649</td>
</tr>
<tr>
<td>22051997</td>
<td>604</td>
</tr>
<tr>
<td>24051997</td>
<td>578</td>
</tr>
<tr>
<td>02051997</td>
<td>573</td>
</tr>
<tr>
<td>21051997</td>
<td>536</td>
</tr>
<tr>
<td>17051997</td>
<td>535</td>
</tr>
<tr>
<td>09051997</td>
<td>533</td>
</tr>
<tr>
<td>03051997</td>
<td>527</td>
</tr>
<tr>
<td>16051997</td>
<td>515</td>
</tr>
<tr>
<td>01051997</td>
<td>502</td>
</tr>
<tr>
<td>30041997</td>
<td>496</td>
</tr>
</tbody>
</table>

Pro Tag (date_cdr1) werden zwischen 400 und 650 Reloads durchgeführt. Die Selektivität ist nicht hervorragend aber für einen Index-Scan auf einen bestimmten Tag durchaus noch akzeptabel.
B*TREE Index Analyse

Um ganz sicher zu sein, ob ein Index sinnvoll ist oder nicht, kann eine Index Analyse durchgeführt werden. Dazu wird der Index analysiert und anschliessend wird das erstellte Index-Histogramm kontrolliert.

```
ANALYZE INDEX cdr_imei_btidx VALIDATE STRUCTURE;

SELECT repeat_count "R-Count",
       keys_with_repeat_count "Keys"
FROM index_histogram;
```

<table>
<thead>
<tr>
<th>R-Count</th>
<th>Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>63163</td>
</tr>
<tr>
<td>512</td>
<td>593</td>
</tr>
<tr>
<td>1024</td>
<td>115</td>
</tr>
<tr>
<td>1536</td>
<td>26</td>
</tr>
<tr>
<td>2048</td>
<td>9</td>
</tr>
<tr>
<td>2560</td>
<td>5</td>
</tr>
<tr>
<td>3072</td>
<td>1</td>
</tr>
<tr>
<td>3584</td>
<td>1</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
</tr>
<tr>
<td>4608</td>
<td>0</td>
</tr>
<tr>
<td>5120</td>
<td>0</td>
</tr>
<tr>
<td>5632</td>
<td>0</td>
</tr>
<tr>
<td>6144</td>
<td>0</td>
</tr>
<tr>
<td>6656</td>
<td>0</td>
</tr>
<tr>
<td>7168</td>
<td>0</td>
</tr>
<tr>
<td>7680</td>
<td>0</td>
</tr>
</tbody>
</table>

Auswertung

Es gibt 63'163 IMEI-Werte, welche genau ein einziges Mal vorkommen in der Tabelle CDR. Für diese 63'163 IMEI's ist ein Indexzugriff sehr effizient, da die Selektivität sehr gut ist. Wir haben jedoch auch einen IMEI-Wert, der 3072 mal vorkommt, wird also nach diesem IMEI-Wert selektiert, dann ist ein Indexzugriff nicht vorteilhaft. Welche IMEI-Werte kommt denn so oft vor ?

```
SELECT imei
FROM CDR
GROUP BY imei
HAVING COUNT(*) > 3070;
```

<table>
<thead>
<tr>
<th>IMEI</th>
</tr>
</thead>
<tbody>
<tr>
<td>490137201144390</td>
</tr>
<tr>
<td>490137203883610</td>
</tr>
<tr>
<td>490137203905860</td>
</tr>
</tbody>
</table>
2.4 Datenverteilung

Der COST Based Optimizer geht normalerweise von einer gleichmäßigen Datenverteilung aus, er kennt also nur die Anzahl Rows und die Anzahl unterschiedlicher Datenwerte (distinct Keys). Diese Werte werden ihm mittels ANALYZE jede Nacht zur Verfügung gestellt.

Bei der folgenden Abfrage ist damit zu rechnen, dass nicht die gleiche Selektivität vorhanden ist:

```sql
SELECT ename, sal FROM emp
WHERE ename = 'MEIER';

SELECT ename, sal FROM emp
WHERE ename = 'VAN NIEVERGELT';
```

Typischerweise findet man oft folgenden Sachverhalt:

- Die häufigsten Datenwerte werden auch am häufigsten abgefragt
- Die häufigsten Datenwerte werden am wenigsten abgefragt.

In beiden Fällen besteht das Dilemma, dass für einige Abfragen ein Index vorteilhaft, für andere Abfragen ein Full-Table Scan besser wäre. Genau dieses Problem lösen ab Oracle 7.3 Histogramme.

Histogramme zeigen die Verteilung und die Häufigkeit der Datenwerte auf. Damit kann der CBO entscheiden, ob er einen Index-Scan oder einen Fulltable-Scan anwenden soll.

Histogramme werden mittels ANALYZE ... FOR ... erstellt. Sie können mit der View USER_HISTOGRAMS dargestellt werden.

<table>
<thead>
<tr>
<th>Bucket</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endwert</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
</tbody>
</table>

Gleichmäßige Datenverteilung: 10 % der Daten sind in Bucket 10

<table>
<thead>
<tr>
<th>Bucket</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endwert</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>100</td>
</tr>
</tbody>
</table>

Ungleichmäßige Datenverteilung: 30 % der Daten müssen sich Bucket 10 teilen
2.5 Join Algorithmen


Sort-Merge Join


Nachteil des Sort-Merge: Sortiervorgang, obwohl dies durch Oracle sehr effizient gemacht wird.

Beispiel

```sql
SELECT e.ename, e.sal, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

Execution Plan

```sql
SELECT STATEMENT Optimizer=CHOOSE
MERGE JOIN
SORT (JOIN)
TABLE ACCESS (FULL) OF 'EMP'
SORT (JOIN)
TABLE ACCESS (BY ROWID) OF 'DEPT'
```

Darstellung des Execution Plans

![Diagramm des Execution Plans]

Sort-Merge Algorithmus

1. **Driving Table** (Hier wird begonnen)
2. **Sort**
3. **Sort (JOIN)**
4. **TABLE ACCESS (FULL) OF 'EMP'**
5. **SORT (JOIN)**
6. **TABLE ACCESS (BY ROWID) OF 'DEPT'**
7. **Merge-Join**
Hash-Join


Beispiel

SELECT e.ename, e.sal, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;

Execution Plan
-----------------------------------------------------
SELECT STATEMENT Optimizer=ALL_ROWS
   HASH JOIN
      TABLE ACCESS (FULL) OF 'EMP'
      TABLE ACCESS (FULL) OF 'DEPT'

Darstellung des Execution Plans

Der Hash-Join wird ab Version 7.3 dem Sort-Merge Join bevorzugt eingesetzt. Um den Hash-Join zu aktivieren, müssen die entsprechenden INIT.ORA Parameter gesetzt sein, dies sind insbesondere die Parameter HASH_JOIN_ENABLED und HASH_AREA_SIZE.
Nested Loop


- Für kleinere Datenmengen.
- Ideal wenn WHERE Klausel Einschränkung enthält.
- Zu verarbeitende Datenmenge ist vom äußeren Loop abhängig.

Algorithmus

Algorithmus des Nested Loop

```
BEGIN
  FOR outer_loop IN (SELECT deptno FROM dept)
  LOOP
    FOR inner_loop IN (SELECT empno FROM emp
                         WHERE deptno = outer_loop.deptno)
    LOOP
      RETURN;
    END LOOP inner_loop;
  END LOOP outer_loop;
END;
```

Beispiel

SELECT e.ename, e.sal, d.dname
FROM emp e, dept d
WHERE d.deptno = e.deptno
  AND d.dname = 'SALES';

Execution Plan

```
SELECT STATEMENT Optimizer=CHOOSE
  NESTED LOOPS
    TABLE ACCESS (FULL) OF 'DEPT'
    TABLE ACCESS (BY ROWID) OF 'EMP'
    INDEX (RANGE SCAN) OF 'EMP_DETNO'
```

Darstellung des Execution Plans

![Diagram of Execution Plan](image-url)
Star Joins

Normalerweise versucht der Optimizer um jeden Preis ein Kartesisches Produkt zu vermeiden und joint nur Tabellen, welche aufgrund der WHERE Klausel auch wirklich joinbar sind. Bei sehr grossen Tabellen, welche vor allem in Decision Support Systemen (DSS) oder auch DataWareHouses vorkommen, bietet Oracle 7.3 eine spezielle Join-Methode an, den Star Join. Bei einem Star-Join wird eine sehr grosse zentrale Tabelle (Fact Table) geschaffen, welche mit mindestens 2 weiteren kleinen Tabellen einen Join bilden.

Beurteilung

- Sort-Merge Join und Hash-Join eignen sich für grosse Datenmengen, wo der Gesamtdurchsatz eine wichtige Rolle spielt.
- Der neuere Hash-Join ist dem älteren Sort-Merge Join fast immer überlegen und sollte ab Oracle 7.3 bevorzugt eingesetzt werden.
- Nested Loops sind sehr effizient bei kleineren Datenmengen sowie einer WHERE Klausel im äusseren Loop.
2.6 Execution Plan

Da der COST Based Optimizer seine Entscheidungen aufgrund vieler komplexer Abhängigkeiten fällt, ist es praktisch unmöglich, eine Vorhersage seiner Zugriffe zu machen. Oracle stellt deshalb einige Tools zur Verfügung, welche es erlauben, den Execution Plan zu visualisieren und zu beeinflussen (Hints).

Folgende Tools stehen zur Verfügung:

**AutoTrace in SQL*Plus**


**Installation von AutoTrace**

Nur notwendig, wenn dies nicht bereits gemacht wurde.


2. Die `PLAN_TABLE` muss an alle User die AutoTrace nutzen dürfen verfügbar gemacht werden.

   ```sql
   GRANT ALL ON SYS.PLAN_TABLE TO ...;
   CREATE PUBLIC SYNONYM PLAN_TABLE FOR SYS.PLAN_TABLE;
   ```


4. Die Rolle «PLUSTRACE» an alle User die AutoTrace nutzen dürfen granten.

   ```sql
   GRANT PLUSTRACE TO ...;
   ```

**Nutzen von AutoTrace**

Innerhalb von SQL*PLUS können folgende Optionen gesetzt werden:

```sql
SET AUTOTRACE {OFF | ON | TRACEONLY} {EXPLAIN} {STATISTICS}
```

- ON, OFF, TRACEONLY: Ein-, Ausschalten der ganzen Option oder nur Execution Plan anzeigen.
- EXPLAIN
  - Execution Plan anzeigen.
- STATISTICS
  - Statistiken wie Netzwerkverkehr anzeigen.

**Beispiele**

```sql
SQL> set autotrace on explain
SQL> select * from dept;
DEPTNO  DNAME          LOC
--------- -------------- --------------
     10 ACCOUNTING     NEW YORK
     20 RESEARCH       DALLAS

Execution Plan
SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=4 Bytes=140)
TABLE ACCESS (FULL) OF 'DEPT' (Cost=1 Card=4 Bytes=140)
```
Oracle Datenbank Tuning

SQL> set autotrace on explain statistics

SQL> select * from dept;

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
<td>DALLAS</td>
</tr>
</tbody>
</table>

Execution Plan

```
SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=4 Bytes=140)
   TABLE ACCESS (FULL) OF 'DEPT' (Cost=1 Card=4 Bytes=140)
```

Statistics

```
0  recursive calls
2  db block gets
1  consistent gets
0  physical reads
0  redo size
386  bytes sent via SQL*Net to client
288  bytes received via SQL*Net from client
3  SQL*Net roundtrips to/from client
0  sorts (memory)
0  sorts (disk)
4  rows processed
```

SQL> set autotrace traceonly explain
SQL> select * from dept;

Execution Plan

```
SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=4 Bytes=140)
   TABLE ACCESS (FULL) OF 'DEPT' (Cost=1 Card=4 Bytes=140)
```

AutoTrace ist erst ab Oracle 7.3 verfügbar. Es eignet sich hervorragend für rasche ad hoc Kontrollen. Es empfiehlt sich deshalb sehr, die eigenen SQL-Statements auf diese Weise rasch zu kontrollieren und allenfalls mittels HINT's zu tunen. Um auch eine Zeitmessung in SQL*PLUS anzuzeigen, kann die Option SET TIMING ON zusätzlich eingeschaltet werden.

Angaben im Execution Plan

Optimizer: Angabe welcher Optimizer gewählt wurde.

Für den CBO werden zusätzlich noch die folgenden Angaben gemacht:

Cost: «Kostenberechnung» des CBO


Bytes: Schätzung der zu verarbeitenden Bytes.
Das aktuelle SQL-Statement einer beliebigen Session kann sehr rasch und auf einfache Weise analysiert werden.

**TopSession im Enterprise Manager**

Session auswählen

Details der Session

Execution Plan der Session
**TKPROF**

Genaue Analyse des SQL-Statements anhand eines generierten Tracefiles. Dazu muss das SQL-Tracing eingeschaltet werden und anschließend wird das erzeugte Tracefile analysiert.

SQL-Tracing einschalten

SQL ausführen

SQL-Tracing ausschalten

TKPROF

Trace-File

Output

**Installation von TKPROF**

Nur notwendig, wenn dies nicht bereits gemacht wurde.

1. Stored Procedure *trace* als User SYS installieren. Damit wird der Name des erzeugten Trace-Files angezeigt.

```sql
CREATE OR REPLACE PROCEDURE trace (pOnOff IN BOOLEAN) IS
  vDirectory  v$parameter.value%TYPE;
  vFile       utl_file.file_type;
  vFileName   VARCHAR2(9) := 'trace.log';
  vMode       VARCHAR2(1) := 'a'; -- append
  vGlobal_name global_name.global_name%TYPE;
  vSPid       v$process.spid%TYPE;
  vServer     v$session.server%TYPE;
  vDate       VARCHAR2(21) := TO_CHAR(SYSDATE, 'DD.MM.YYYY HH24:MI.SS');

  /*
  * Internal function to find the first valid directory for utl_file.
  * There can be more than one.
  * If there is none, then the fopen will fail..., so make sure that
  * there is one...
  */
  FUNCTION utl_file_dir RETURN VARCHAR2 IS
    BEGIN
      SELECT value INTO vDirectory
      FROM v$parameter
      WHERE name = 'utl_file_dir';
      EXCEPTION
        WHEN NO_DATA_FOUND THEN
          vDirectory := NULL;
      END;
      IF (INSTR(vDirectory,'') > 0) THEN
        vDirectory := SUBSTR(vDirectory,1,INSTR(vDirectory,'')-1);
      END IF;
      RETURN(LTRIM(RTRIM(vDirectory))); END utl_file_dir;
```

SQL-Tracing einschalten

SQL ausführen

TKPROF

Trace-File

Output
BEGIN
  IF (pOnOff) THEN
    BEGIN
    SELECT g.global_name, p.spid, s.server
    INTO vGlobal_name, vSPid, vServer
    FROM v$session s, v$process p, global_name g
    WHERE s.paddr = p.addr
    AND s.audsid = USERENV('sessionid');
    END;

    BEGIN
    vFile := utl_File.FOpen(utl_file_dir, vFilename, vMode);
    EXCEPTION
      WHEN utl_File.invalid_operation THEN
        /* The file does not yet exist, so create it...*/
        vFile := utl_File.FOpen(utl_file_dir, vFilename, 'w');
    END;

    utl_File.puts(vFile, 'At %s user %s@%s opens file %s with %s server 

    utl_File.close(vFile);
    dbms_output.put_line('You have server '||vSPid||' ('||vServer|')');
  END IF;
DBMS_Session.set_sql_trace(pOnOFF);
END;
/

2. Identifikation des Trace Directory, definiert im Parameter
 USER_DUMP_DEST im File INIT.ORA. Fragen Sie am besten den
 DB-Verantwortlichen nach dem Ort und den Berechtigungen dieses
 Directory. Für das PBB Projekt lautet dieses Directory:

    GD3I9K: /sipp/oracle/admin/P18M7S/udmp
GD3IQB: /sipp/oracle/admin/T18M7S/udmp

Nutzten von TKPROF

Innerhalb von SQL*PLUS wird Tracing ein- und ausgeschaltet:

SQL> set serveroutput on
SQL> execute trace(true);
You have server 24539 (DEDICATED)
SQL> SELECT e.ename, e.sal, d.dname
FROM emp e, dept d
WHERE d.deptno = e.deptno
AND d.dname = 'SALES';
SQL> execute trace(false);

Nun in das Trace Directory wechseln

$ cd /sipp/oracle/admin/T18M7S/udmp

Nun wird das Outputfile trace.lis generiert

$ tkprof ora_24539.trc trace.lis explain=user/pwd sort=exeela sys=no

explain=user/pwd Connect to ORACLE and issue EXPLAIN PLAIN.
sys=no TKPROF does not list SQL statements run as user SYS.
exeela elapsed time executing
Das Outputfile von TKPORF hat den folgenden Inhalt

TKPROF: Release 7.3.2.2.0 – Production on Sat Jun 7 15:12:15 1997
Copyright (c) Oracle Corporation 1979, 1994. All rights reserved.
Trace file: ora_26996.trc
Sort options: execela

******************************************************************************
count = number of times OCI procedure was executed
cpu   = cpu time in seconds executing
elapsed = elapsed time in seconds executing
disk  = number of physical reads of buffers from disk
query = number of buffers gotten for consistent read
current = number of buffers gotten in current mode (usually for update)
rows  = number of rows processed by the fetch or execute call
******************************************************************************
SELECT e.ename, e.sal, d.dname
FROM emp e, dept d
WHERE d.deptno = e.deptno
AND d.dname = 'SALES'
call count cpu elapsed disk query current rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse  1  0.01  0.01     0   0     0     0   0
Execute 1  0.00  0.00     0   0     0     0   0
Fetch  1  0.01  0.02     2   3     4     6  6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total  3  0.02  0.03     2   3     4     6  6

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 28  (SCOTT)

Auswertung

• Von grosser Wichtigkeit sind die Anzahl Rows, welche auf jeder Stufe verarbeitet werden. Hier ist darauf zu achten, dass eine Restriktion der Datenmenge aufgrund der WHERE Klausel so früh als möglich eintritt.
• Sehr hohe Zahlen bei «Query» deuten auf eine Repetition der gleichen Arbeit hin. Insbesondere bei ungünstigen Nested Loops ist «Query» oftmals sehr gross und Einsparungen sind möglich.
• «CPU» und «ELAPSED» sollten möglichst identisch sein, dann liegt kein Locking Problem vor und das System ist nicht überlastet.
• TKPROF zeigt grundsätzlich die Arbeit an, welche wirklich gemacht wurde. Mit einem Vergleich zu AutoTrace, wo nur eine Schätzung vorhanden ist, kann sehr schnell gesehen werden, wo der CBO falsche Annahmen getroffen hat, welche zu schlechter Performance führen.
2.7 Optimierung


Anbringen eines Hints

Ein Hint wird dem SQL-Befehl als Kommentar und einem «+++» Zeichen mitgegeben. Er ist für den Optimizer nicht zwingend.

```
SELECT /*+ RULE */ e.ename, e.sal, d.dname
FROM emp e, dept d
WHERE d.deptno = e.deptno
  AND d.dname = 'SALES';
```

Strategie zur Optimierung

Um rasch und gezielt eine Performance Verbesserung zu erreichen, empfiehlt sich das folgende Vorgehen.


   - COST Based Optimizer: Hint = CHOOSE
   - RULE Based Optimizer: Hint = RULE

2. Soll möglichst rasch die erste Row angezeigt werden oder ist die gesamte Verarbeitung (Batch) von Wichtigkeit?
   - Möglichst rasch 1 Row anzeigen: Hint = First_Rows
   - Gesamtdurchsatz optimieren: Hint = All_Rows

3. Kann der Zugriff mit einem Hint geändert werden?

4. Kann ein Index verwendet werden?

5. Kann das Statement umformuliert werden?

6. Haben Sie die Regeln für gutes SQL eingehalten?
### Hints

<table>
<thead>
<tr>
<th>Hint</th>
<th>Erläuterung</th>
</tr>
</thead>
<tbody>
<tr>
<td>RULE</td>
<td>Verwendung des Rule Based Optimizer</td>
</tr>
<tr>
<td>CHOOSE</td>
<td>Verwendung des Cost Based Optimizer</td>
</tr>
<tr>
<td>FIRST_ROWS</td>
<td>CBO Optimierung der Zeit zur Anzeige der ersten Row. Index-Scans und Nested Loops werden vom Optimizer bevorzugt eingesetzt.</td>
</tr>
<tr>
<td>ALL_ROWS</td>
<td>CBO Optimierung des Gesamtdurchsatzes. Fulltable-Scans werden vom Optimizer bevorzugt eingesetzt.</td>
</tr>
<tr>
<td>FULL(Table)</td>
<td>Fulltable-Scan auf angegebener Tabelle durchführen</td>
</tr>
<tr>
<td>ORDERED</td>
<td>Join der Tabellen erfolgt in der gleichen Reihenfolge wie sie in der FROM Klausel erscheinen. Dieser Hint steht selten allein, sondern wird vielfach in Kombination mit einem anderen Hint angebracht.</td>
</tr>
<tr>
<td>USE MERGE(Table)</td>
<td>Joine die angegebene Tabelle zum Resultat der vorangegangenen Row-Source mittels einem Sort-Merge Join.</td>
</tr>
<tr>
<td>USE HASH(Table)</td>
<td>Joine die angegebene Tabelle zum Resultat der vorangegangenen Row-Source mittels einem Hash Join.</td>
</tr>
<tr>
<td>USE NL(Table)</td>
<td>Joine die angegebene Tabelle zum Resultat der vorangegangenen Row-Source mittels einem Nested Loop. Die angegebene Tabelle wird im inneren Loop verarbeitet.</td>
</tr>
<tr>
<td>INDEX(Table Index)</td>
<td>Verwendung des Index einer bestimmten Tabelle</td>
</tr>
<tr>
<td>AND_EQUAL(Table Index Index Index ...)</td>
<td>Verwendung eines Index Merge auf bestimmten einzelnen Indizes der angegebenen Tabelle.</td>
</tr>
<tr>
<td>SELECT ename, sal</td>
<td>FROM emp</td>
</tr>
<tr>
<td></td>
<td>WHERE deptno = 10</td>
</tr>
<tr>
<td></td>
<td>AND ename = 'KING';</td>
</tr>
<tr>
<td>HASH_AJ</td>
<td>CBO: Anti Join als Hash auflösen.</td>
</tr>
<tr>
<td></td>
<td>SELECT ...</td>
</tr>
<tr>
<td></td>
<td>WHERE deptno NOT IN (SELECT ...);</td>
</tr>
<tr>
<td></td>
<td>NOT IN Abfragen möglichst vermeiden !</td>
</tr>
<tr>
<td></td>
<td>SELECT ...</td>
</tr>
<tr>
<td></td>
<td>WHERE deptno NOT IN (SELECT ...);</td>
</tr>
<tr>
<td></td>
<td>NOT IN Abfragen möglichst vermeiden !</td>
</tr>
<tr>
<td>CACHE</td>
<td>Full Table Scans lassen die Blocks solange wie möglich in der SGA stehen. Ideal für kleine Lookup Tabellen.</td>
</tr>
<tr>
<td>PARALLEL</td>
<td>Im Zusammenhang mit der Parallel Query Option einsetzbar. Angabe der concurrent Query Server zur Verarbeitung angeben.</td>
</tr>
<tr>
<td>ROWID(Table)</td>
<td>Table Scan über die ROWID der Tabelle.</td>
</tr>
<tr>
<td>CLUSTER(Table)</td>
<td>Verwendung des Index Clusters der Tabelle.</td>
</tr>
<tr>
<td>HASH (Table)</td>
<td>Verwendung des Hash Clusters der Tabelle.</td>
</tr>
</tbody>
</table>

Die zur Verfügung stehenden Hints sind nicht für alle Oracle Versionen verfügbar. Man beachte also nochmals, dass der Optimizer einen unbekannten Hint ignoriert und keine Fehlermeldung generiert.
2.8 Regeln für gutes SQL

Beachten Sie auch die Regeln für den Indexzugriff (siehe Verwendung des Index auf Seite 10).

**Allgemeines**

- Die SELECT Liste steht zur freien Wahl. Alle Funktionen (auch Inline Funktionen) und Möglichkeiten stehen offen.
- Die FROM Klausel soll die Tabellen in der Reihenfolge auflisten, wie der Optimizer sie absuchen soll.
- In der WHERE Klausel sollten keine indexierten Attribute durch Funktionen / Berechnungen verändert werden, da dies den Indexzugriff verhindert.
- Die WHERE Klausel führt zuerst die Join-Attribute, danach die Attribute gemäss ihrer Selektivität auf.
- ORDER BY Klauseln können nur auf unveränderte, indexierte NOT NULL Attribute optimiert werden.
- Joins sind einfacher optimierbar als Subqueries.
- NOT IN sollte grundsätzlich vermieden werden. Im Zusammenhang mit dem RULE Based Optimizer ist die Verwendung verboten!
- Komplexe Views in Joins oder Views auf Views sollten immer mit TKPROF kontrolliert werden.
- Tabellen Aliasnamen in Joins bei allen Columns angeben, auch wenn dies für die Eindeutigkeit nicht notwendig ist.

**Outer Joins**

Bei Outer Joins kann sich der Optimizer allen verfügbaren Join-Methoden bedienen. Die Driving Table ist einfach bestimmbar, es ist die Gegenseite zum (+), da diese Tabelle ja immer ganz gelesen werden soll.

```sql
SELECT d.dname, e.ename
FROM emp e, dept d
WHERE d.deptno = e.deptno (+);
```

**Execution Plan**

```
SELECT STATEMENT Optimizer=CHOOSE
   (Cost=5 Card=14 Bytes=756)
   NESTED LOOPS (OUTER)
      TABLE ACCESS (FULL) OF 'DEPT'
      TABLE ACCESS (FULL) OF 'EMP'
```

**DISTINCT**

DISTINCT im Zusammenhang mit Joins sind «Performance Killer». Das Statement kann praktisch immer umgeschrieben werden:

Schlecht ist:

```sql
SELECT DISTINCT d.deptno, d.dname
FROM dept d, emp e
WHERE d.deptno = e.deptno;
```

besser ist:

```sql
SELECT d.deptno, d.dname
FROM dept d
WHERE EXISTS (SELECT 'X'
               FROM emp e
               WHERE e.deptno = d.deptno);
```
NOT IN

Vermeiden Sie wenn immer möglich den Operator NOT IN

Schlecht ist:

```sql
SELECT *
FROM emp
WHERE deptno NOT IN (SELECT deptno
                      FROM dept
                      WHERE loc = 'BOSTON');
```

besser ist:

```sql
SELECT *
FROM emp e
WHERE NOT EXISTS (SELECT 'X'
                  FROM dept d
                  WHERE d.deptno = e.deptno
                  AND d.loc = 'BOSTON');
```

Inline Functions

Inline Funktionen sind eigene PL/SQL Funktionen, die sich exakt gleich verhalten wie die bekannten Oracle Funktionen AVG, COUNT, SUM, etc. Sie können direkt in SQL-Statements verwendet werden und erweitern so den Funktionsumfang der SQL Sprache mit eigenen Funktionen. Solche Funktionen (Stored Functions) sind bereits vorkompiliert und bezüglich Performance sehr ideal!

Beispiel:

Definieren der Funktion:

```sql
CREATE OR REPLACE FUNCTION sal_sum (p_deptno IN NUMBER)
RETURN NUMBER IS
  l_sum  NUMBER;
BEGIN
  SELECT SUM(sal) INTO l_sum
  FROM emp
  WHERE deptno = p_deptno;
  RETURN (l_sum);
END;
```

Verwenden der Funktion in normalem SQL-Statement:

```sql
SELECT deptno, dname, sal_sum(deptno) FROM dept;
```
3. Tuning ausgewählter SQL-Statements

3.1 COST- oder RULE-based Optimizer?

COST-based

Für den COST-based Optimizer ist der Entscheid zwischen Fulltable-Scan oder Index-Scan schwierig zu fällen, wenn das Resultset gross ist.

```sql
SELECT ORIGIN_ID,
       TO_CHAR(CREATE_DATE, 'YYYYMMDDHH24MISS'),
       TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'),
       MS_ID,
       RELOAD_ID
FROM CREDIT
WHERE (STATUS = 0 AND ACCOUNT_DATE < (SYSDATE - (1 / 24)));
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>9792</td>
<td>19.88</td>
<td>20.43</td>
<td>6</td>
<td>12117</td>
<td>2</td>
<td>146876</td>
</tr>
<tr>
<td>total</td>
<td>9794</td>
<td>19.88</td>
<td>20.43</td>
<td>6</td>
<td>12117</td>
<td>2</td>
<td>146876</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: **CHOOSE**
Parsing user id: 10 (SIPP)
Rows Execution Plan

---

RULE-based

Wie verhält sich der RULE-based Optimizer?

```sql
SELECT /*+ RULE */ ORIGIN_ID,
       TO_CHAR(CREATE_DATE, 'YYYYMMDDHH24MISS'),
       TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'),
       MS_ID,
       RELOAD_ID
FROM CREDIT
WHERE (STATUS = 0 AND ACCOUNT_DATE < (SYSDATE - (1 / 24)));
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>9806</td>
<td>20.55</td>
<td>22.50</td>
<td>2251</td>
<td>12142</td>
<td>2</td>
<td>147075</td>
</tr>
<tr>
<td>total</td>
<td>9808</td>
<td>20.56</td>
<td>22.50</td>
<td>2251</td>
<td>12142</td>
<td>2</td>
<td>147075</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 1
Optimizer goal: **RULE**
Parsing user id: 10 (SIPP)
Rows Execution Plan

---

Der RULE-based Optimizer versucht den Zugriff wenn möglich über einen Index herzustellen. Da hier jedoch kein Index vorhanden ist, wählt auch er einen Full-Table Scan. Der Bitmapped Index auf dem Attribut STATUS kann nur vom COST-based Optimizer genutzt werden.
Nun ermöglichen wir einen Indexzugriff:
B-Tree Index auf ACCOUNT_DATE, Bitmapped Index auf STATUS:

CREATE INDEX CREDIT_ACCOUNT_DATE_IDX ON CREDIT (ACCOUNT_DATE)
TABLESPACE IDX_SIPP STORAGE (INITIAL 1M NEXT 1M
MAXEXTENTS UNLIMITED PCTINCREASE 0);

CREATE BITMAP INDEX CREDIT_STATUS_BIDX ON CREDIT (STATUS)
TABLESPACE IDX_SIPP STORAGE (INITIAL 1M NEXT 1M
MAXEXTENTS UNLIMITED PCTINCREASE 0);

SELECT ORIGIN_ID, TO_CHAR(CREATE_DATE,
'YYYYMMDDHH24MISS'),
TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'), MS_ID, RELOAD_ID
FROM CREDIT
WHERE (STATUS = 0 AND ACCOUNT_DATE < (SYSDATE - (1 / 24)))

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>9792</td>
<td>28.10</td>
<td>30.12</td>
<td>798</td>
<td>294688</td>
<td>0</td>
<td>146878</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 10  (SIPP)
Rows   Execution Plan

0 SELECT STATEMENT  GOAL: CHOOSE
147329  TABLE ACCESS  GOAL: ANALYZED (BY ROWID) OF 'CREDIT'
147330  INDEX (RANGE SCAN) OF 'CREDIT_ACCOUNT_DATE_IDX' (NON-UNIQUE)

Da sehr viele Rows zurückkommnen ist der Full-Table Scan deutlich schneller. Index auf ACCOUNT_DATE wieder droppen!
3.2 Join-Methoden (Algorithmen)

Join-Methoden

In diesem Beispiel wird die Column MSISDN selektiert, das Result-Set besteht aus genau einer Row.

```
SELECT MSISDN
FROM CUSTOMER, MSISDN, BINDING, SIM
WHERE CUSTOMER.CUS_ID = MSISDN.CUS_ID
AND MSISDN.MS_ID = BINDING.MS_ID
AND BINDING.IMSI = SIM.IMSI
AND BINDING.STATUS = 0
AND SRN BETWEEN '8941019622417348629' AND '89410196224173486299'
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>2.71</td>
<td>3.41</td>
<td>7</td>
<td>3238</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td>4</td>
<td>2.71</td>
<td>3.41</td>
<td>7</td>
<td>3238</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 10  (SIPP)

```
Rows Execution Plan
0 SELECT STATEMENT    | GOAL: CHOOSE
1 NESTED LOOPS           | 
2 HASH JOIN
101995 TABLE ACCESS    | GOAL: ANALYZED (FULL) OF 'MSISDN'
1 NESTED LOOPS
101995 TABLE ACCESS    | GOAL: ANALYZED (FULL) OF 'SIM'
1 TABLE ACCESS GOAL: ANALYZED (BY ROWID) OF 'BINDING'
2 INDEX GOAL: ANALYZED (RANGE SCAN) OF 'BINDING_IDX1' (UNIQUE)
1 INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_CUSTOMER' (UNIQUE)
```

---

Seite 32 von 82
Der Join zwischen der Tabelle SIM und BINDING erfolgt in einem Nested Loop. Dazu wird die gesamte Tabelle SIM gelesen und der innere Loop wird 101'995 mal ausgeführt um genau eine Row zurückzubringen. Der Join zwischen diesem Ergebnis (SIM, BINDING) und der Tabelle MSISDN wird in einem HASH-Join vorgenommen, die Tabelle MSISDN wird ganz gelesen, der Join bringt 2 Rows zurück. Der Join zwischen diesem Ergebnis und der Tabelle CUSTOMER wird wieder in einem Nested Loop durchgeführt, der innere Loop wird nur noch 2 mal ausgeführt, über den Primary Key der Tabelle CUSTOMER wird das Result-Set von einer Row schliesslich zurückgegeben. Der erste NL ist hier schlecht, da er 101'995 mal ausgeführt werden muss.
Da wir wissen, dass die Tabelle SIM einen UNIQUE Index auf dem Attribut SRN hat, geben wir dies dem Optimizer in einem Hint bekannt. Da nur eine einzige Row zurückerholt wird, ist es von Vorteil alle Joins über einen Nested Loop Algorithmus zu joinen, dazu dient der Hint FIRST_ROWS.

```
SELECT /*+ INDEX(SIM SIM_IDX1) FIRST_ROWS */ MSISDN
FROM CUSTOMER, MSISDN, BINDING, SIM
WHERE CUSTOMER.CUS_ID = MSISDN.CUS_ID
  AND MSISDN.MS_ID = BINDING.MS_ID
  AND BINDING.IMSI = SIM.IMSI
  AND BINDING.STATUS = 0
  AND SRN BETWEEN '8941019622417348629' AND '89410196224173486299' ;
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: FIRST_ROWS
Parsing user id: 10  (SIPP)

**Rows Execution Plan**

<table>
<thead>
<tr>
<th>Rows</th>
<th>Execution Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT GOAL: HINT: FIRST_ROWS</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS GOAL: ANALYZED (BY ROWID) OF 'SIM'</td>
</tr>
<tr>
<td>2</td>
<td>INDEX GOAL: ANALYZED (RANGE SCAN) OF 'SIM_IDX1' (UNIQUE)</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS GOAL: ANALYZED (BY ROWID) OF 'BINDING'</td>
</tr>
<tr>
<td>2</td>
<td>INDEX GOAL: ANALYZED (RANGE SCAN) OF 'BINDING_IDX1' (UNIQUE)</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS GOAL: ANALYZED (RANGE SCAN) OF 'MSISDN'</td>
</tr>
<tr>
<td>1</td>
<td>INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_MSISDN' (UNIQUE)</td>
</tr>
<tr>
<td>1</td>
<td>INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_CUSTOMER' (UNIQUE)</td>
</tr>
</tbody>
</table>

******************************************************************************

**Execution Plan grafisch dargestellt:**

Auf jeder Stufe wird nur noch das Notwendigste verarbeitet, die äußeren Loops werden nie mehr als 2 mal durchlaufen, dementsprechend schnell wird die gesuchte MSISDN gefunden.
Wie verhält sich in diesem Fall der RULE based Optimizer?

```
SELECT /*+ RULE */ MSISDN
FROM CUSTOMER, MSISDN, BINDING, SIM
WHERE CUSTOMER.CUS_ID = MSISDN.CUS_ID
AND MSISDN.MS_ID = BINDING.MS_ID
AND BINDING.IMSI = SIM.IMSI
AND BINDING.STATUS = 0
AND SRN BETWEEN '8941019622417348629' AND '89410196224173486299'
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: RULE
Parsing user id: 10 (SIPP)

Der RULE based Optimizer wählt in diesem Fall den optimalen Execution Plan, da er wenn möglich Indexe verwendet und Fulltable Scans vermeidet. Die Lösung besteht hier also den RULE based Optimizer zu verwenden, da dadurch der Hint einfacher wird.
3.3 Ein oder mehrere Attribute in der Selektion

SELECT * ...

Es stellt sich nun die Frage, ob sich der Execution Plan ändert, wenn alle Attribute in die Selektion aufgenommen werden.

SELECT *
FROM CUSTOMER, MSISDN, BINDING, SIM
WHERE CUSTOMER.CUS_ID = MSISDN.CUS_ID
AND MSISDN.MS_ID = BINDING.MS_ID
AND BINDING.IMSI = SIM.IMSI
AND BINDING.STATUS = 0
AND SRN BETWEEN '8941019622417348629' AND '89410196224173486299'

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>2</td>
<td>0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>9.80</td>
<td>14.62</td>
<td>5</td>
<td>4075</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td>4</td>
<td>9.81</td>
<td>14.63</td>
<td>5</td>
<td>4075</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 10  (SIPP)

Rows  Execution Plan
-------  ---------------------------------------------------
0  SELECT STATEMENT  GOAL: CHOOSE
1  HASH JOIN
101995  TABLE ACCESS  GOAL: ANALYZED (FULL) OF 'SIM'
203888  HASH JOIN
1940  TABLE ACCESS  GOAL: ANALYZED (FULL) OF 'CUSTOMER'
181250  HASH JOIN
101998  TABLE ACCESS  GOAL: ANALYZED (FULL) OF 'BINDING'
101995  TABLE ACCESS  GOAL: ANALYZED (FULL) OF 'MSISDN'

In diesem Fall tritt das typische Problem des COST based Optimizers zutage, er schätzt das zu erwartende Result Set falsch ein. Er wählt für jeden Join eine Hash-Algorithmus, der für ein grosses Result Set geeignet ist. In diesem Fall wird aber nur eine Row selektiert, dementsprechend ist der gewählte Execution Plan in diesem Fall sehr ungeeignet. Wie verhält sich der RULE Based Optimizer?
Wie verhält sich in diesem Fall der RULE based Optimizer?

SELECT /*+ RULE */ *
FROM CUSTOMER, MSISDN, BINDING, SIM
WHERE CUSTOMER.CUS_ID = MSISDN.CUS_ID
AND MSISDN.MS_ID = BINDING.MS_ID
AND BINDING.IMSI = SIM.IMSI
AND BINDING.STATUS = 0
AND SRN BETWEEN '8941019622417348629' AND '89410196224173486299'

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td>3</td>
<td>0.01</td>
<td>0.02</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 1
Optimizer goal: RULE
Parsing user id: 10 (SIPP)

Rows | Execution Plan
-----------------------------------------------
0    | SELECT STATEMENT GOAL: HINT: RULE
1    | NESTED LOOPS
1    | NESTED LOOPS
1    | NESTED LOOPS
1    | TABLE ACCESS GOAL: ANALYZED (BY ROWID) OF 'SIM'
2    | INDEX GOAL: ANALYZED (RANGE SCAN) OF 'SIM_IDX1' (UNIQUE)
1    | TABLE ACCESS GOAL: ANALYZED (BY ROWID) OF 'BINDING'
1    | INDEX GOAL: ANALYZED (RANGE SCAN) OF 'BINDING_IDX1' (UNIQUE)
1    | TABLE ACCESS GOAL: ANALYZED (BY ROWID) OF 'MSISDN'
1    | INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_MSISDN' (UNIQUE)
1    | TABLE ACCESS GOAL: ANALYZED (BY ROWID) OF 'CUSTOMER'
1    | INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_CUSTOMER' (UNIQUE)

Der RULE based Optimizer wählt hier den optimalsten Execution Plan.
3.4 Subquery Optimierung

Sehr oft finden Vergleiche zwischen Tabellen statt, im folgenden Beispiel wird untersucht, welche Teilnehmer (MSISDN) noch nie einen Reload (CREDIT) durchgeführt haben. Man untersucht also, welche MSISDN Nummern nicht in der Tabelle CREDIT vorhanden sind.

**Variante 1**

Mittels MINUS

```sql
select count(*)
from msisdn
where ms_id in (select ms_id from msisdn minus select ms_id from credit)
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>2</td>
<td>0.01</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>4</td>
<td>0.01</td>
<td>0.15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>2</td>
<td>24.19</td>
<td>33.06</td>
<td>4974</td>
<td>8365</td>
<td>2246</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>8</td>
<td>24.21</td>
<td>33.23</td>
<td>4974</td>
<td>8365</td>
<td>2246</td>
<td>2</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 10 (SIPP)

**Rows Execution Plan**

```text
0 SELECT STATEMENT GOAL: CHOOSE
0 SORT (AGGREGATE)
95087 HASH JOIN
101995 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'MSISDN'
64166 VIEW
139824 MINUS
101995 SORT (UNIQUE)
101995 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'MSISDN'
156788 SORT (UNIQUE)
156788 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'CREDIT'
```

**Variante 2**

Mittels NOT IN

```sql
select count(*)
from msisdn
where ms_id not in (select ms_id from credit)
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>2</td>
<td>0.00</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>2</td>
<td>24.72</td>
<td>26.69</td>
<td>283</td>
<td>818552</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>6</td>
<td>24.72</td>
<td>26.70</td>
<td>283</td>
<td>818552</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 10 (SIPP)

**Rows Execution Plan**

```text
0 SELECT STATEMENT GOAL: CHOOSE
0 SORT (AGGREGATE)
101995 FILTER
101995 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'MSISDN'
101995 INDEX GOAL: ANALYZED (RANGE SCAN) OF 'CREDIT_IDX1' (NON-UNIQUE)
```
**Variante 3**

*Mittels NOT EXISTS*

```sql
select count(*)
from msisdn m
where not exists (select 'x' from credit c where c.ms_id = m.ms_id)
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>2</td>
<td>24.76</td>
<td>27.23</td>
<td>290</td>
<td>818552</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>6</td>
<td>24.76</td>
<td>27.23</td>
<td>290</td>
<td>818552</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 10 (SIPP)

**Execution Plan**

0 SELECT STATEMENT GOAL: CHOOSE
0 SORT (AGGREGATE)
101995 FILTER
101995 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'MSISDN'
101995 INDEX GOAL: ANALYZED (RANGE SCAN) OF 'CREDIT_IDX1' (NON-UNIQUE)

**Variante 4**

*Mittels Outer Join*

```sql
select count(*)
from msisdn m, credit c
where m.ms_id = c.ms_id (+)
and c.ms_id is null
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>2</td>
<td>11.34</td>
<td>21.50</td>
<td>4659</td>
<td>6656</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>8</td>
<td>11.34</td>
<td>21.50</td>
<td>4659</td>
<td>6656</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 10 (SIPP)

**Execution Plan**

0 SELECT STATEMENT GOAL: CHOOSE
0 SORT (AGGREGATE)
220956 FILTER
231345 HASH JOIN (OUTER)
101995 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'MSISDN'
156790 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'CREDIT'

Die Variante 2 darf mit dem RULE-based Optimizer auf keinen Fall angewendet werden. NOT IN ist grundsätzlich mit dem RULE-based Optimizer verboten. Der COST-based Optimizer erkennt das NOT IN und wandelt es in ein EXISTS um. Interessant ist die Variante 4 (Outer Join), jedoch schwer verständlich und sollte eigentlich aus diesem Grund vermieden werden.
3.5 Zählen von Rows

COUNT

Zählen der Anzahl Rows (1'537'723) in der sehr grosse Tabelle CDR.

```sql
select count(*)
from cdr;
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>3</td>
<td>50.16</td>
<td>144.41</td>
<td>191775</td>
<td>192321</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>total</td>
<td>9</td>
<td>50.16</td>
<td>144.41</td>
<td>191775</td>
<td>192321</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimier goal: CHOOSE
Parsing user id: 10  (SIPP)

Rows Execution Plan

```
0 SELECT STATEMENT GOAL: CHOOSE
0 SORT (AGGREGATE)
1537723 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'CDR'
```

Der COST-based Optimizer wählt einen Full-Table Scan, der Aufwand ist beträchtlich. Wie verhält es sich wenn man einen Index-Scan auf dem Primary Key wählt?

```sql
select /*+ index (cdr pk_cdr) */ count(*)
from cdr;
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>4</td>
<td>0.02</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>4</td>
<td>19.57</td>
<td>41.89</td>
<td>8746</td>
<td>42640</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>total</td>
<td>12</td>
<td>19.59</td>
<td>41.90</td>
<td>8746</td>
<td>42640</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimier goal: CHOOSE
Parsing user id: 10  (SIPP)

Rows Execution Plan

```
0 SELECT STATEMENT GOAL: CHOOSE
0 SORT (AGGREGATE)
1537723 INDEX GOAL: ANALYZED (FULL SCAN) OF 'PK_CDR' (UNIQUE)
```

Der Aufwand sinkt beträchtlich, da hier nur der gesamte Index sequentiell auf dem Primary Key gelesen werden muss.
3.6 SQL-Funktionen die Indexzugriff verhindern

Jede SQL Funktion auf der linken Seite der WHERE Klausel verhindert einen Indexzugriff. Oft kann das Statement umformuliert werden um den Indexzugriff zu ermöglichen wie das folgende Beispiel zeigt:

```
SELECT COUNT(*)
FROM acm
WHERE TRUNC(date_cdr1) = TRUNC(sysdate);
```

Mit einem Index auf date_cdr1 kann die Abfrage sehr beschleunigt werden, aber das Statement muss umgeschrieben werden so dass der Index auch gebraucht werden kann.

```
CREATE INDEX acm_date_cdr1_btidx ON SIPP.ACM (DATE_CDR1)
TABLESPACE idx_sipp
STORAGE (INITIAL 5M NEXT 5M MINEXTENTS 1 MAXEXTENTS UNLIMITED PCTINCREASE 0)
PCTFREE 10;
ANALYZE INDEX acm_date_cdr1_btidx COMPUTE STATISTICS;
```

```
SELECT COUNT(*)
FROM acm
WHERE date_cdr1 BETWEEN TRUNC(sysdate) AND TRUNC(sysdate) + .99999;
```

---

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.00</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>3.50</td>
<td>4.83</td>
<td>309</td>
<td>1379</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Total: 3, 3.50, 4.84, 309, 1379, 2, 1

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 10 (SIPP)
Rows Execution Plan

---

```plaintext
126994 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'ACM'
```

************************************************************************************

---

Mit einem Index auf date_cdr1 kann die Abfrage sehr beschleunigt werden, aber das Statement muss umgeschrieben werden so dass der Index auch gebraucht werden kann.

```
CREATE INDEX acm_date_cdr1_btidx ON SIPP.ACM (DATE_CDR1)
TABLESPACE idx_sipp
STORAGE (INITIAL 5M NEXT 5M MINEXTENTS 1 MAXEXTENTS UNLIMITED PCTINCREASE 0)
PCTFREE 10;
ANALYZE INDEX acm_date_cdr1_btidx COMPUTE STATISTICS;
```

```
SELECT COUNT(*)
FROM acm
WHERE date_cdr1 BETWEEN TRUNC(sysdate) AND TRUNC(sysdate) + .99999;
```

---

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Total: 3, 0.01, 0.01, 0, 3, 0, 1

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 10 (SIPP)
Rows Execution Plan

---

```plaintext
126994 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'ACM'
```
4. Datenbank Tuning

Übersicht

Das Datenbank Tuning wird in die folgenden Teilgebiete gegliedert:

- Cache und Memory Optimierung
- Indexierungs Strategie
- Fragmentierungen
- I/O-Optimierung
- Block Contention
- Transaktionen und Locking

Im Performance Pack können einige der genannten Überwachungsaufgaben wahrgenommen werden. Für detailliertere Auswertungen werden weiterhin Scripts eingesetzt.

4.1 Oracle Performance Pack

Overview Window

Der Performance Manager zeigt die wichtigsten Parameter im «Overview» Window.
**Buffer Cache Hit %**

Der Buffer Cache enthält Kopien der gelesenen Daten Blocks aus den Datenbankfiles. Alle Sessions teilen sich den Buffer Cache, der Inhalt des Buffer Caches wird gemäß einem LRU Algorithmus mittels DBWR auf die DB geschrieben. Muss ein Block im Datenbankfile gelesen werden, so handelt es sich dabei um einen **Cache Miss**, wird der Block bereits im Memory gefunden so spricht man von einem **Cache Hit**.

Die Tabelle V$SYSSTAT zeigt die kumulierten Werte seit die Instance gestartet wurde. Die Physical Reads seit dem Instance Startup verschlechtert die Hit-Ratio, nach einer bestimmten Zeit pendelt sich der Wert ein.

Es sollte ein HitRatio mit folgenden Werten angestrebt werden:

Datenbank mit vielen Online-Transaction intensiven Benutzern: .... 98%
Batch intensive Applikation: .............................................................89%

**Berechnung**

\[
\text{HitRatio} \% = \left[ 1 - \frac{\text{SUM (Physical Reads)}}{\text{SUM (Consistent Gets + DB Block Gets)}} \right] \times 100
\]

**SQL Statements**

```
select (1 - (sum(decode(a.name,'physical reads',value,0)) / (sum(decode(a.name,'consistent gets',value,0)) + sum(decode(a.name,'db block gets',value,0))))) * 100,
       ((sum(decode(a.name,'physical reads',value,0)) / (sum(decode(a.name,'consistent gets',value,0)) + sum(decode(a.name,'db block gets',value,0))))) * 100
from v$sysstat a;
```

Bei einer schlechtem HitRatio sollten zuerst die Queries mit dem grössten Einfluss auf die HitRatio isoliert werden. Anschliessend muss unter Umständen der Wert für DB_BLOCK_BUFFERS erhöht werden.
Session mit schlechtem Hit-Ratio

Welche Sessions verursachen den größten negativen Effekt auf die Instance HitRatio?

```
select substr(a.username,1,12) "User",
a.sid "sid",
b.consistent_gets "ConsGets",
b.block_gets "BlockGets",
b.physical_reads "PhysReads",
100 * round((b.consistent_gets + b.block_gets - b.physical_reads) / (b.consistent_gets + b.block_gets),3) HitRatio
from v$session a, v$sess_io b
where a.sid = b.sid
and (b.consistent_gets + b.block_gets) > 0
and a.username is not null
order by HitRatio asc;
```

Sessions mit einem schlechten Buffer Cache Ratio. Diese Session benötigen am meisten Ressourcen.

Logical Reads

Logical Reads ist die Summe der Consistent Gets + Block Gets. Sowohl für Consistent Gets wie auch für Block Gets erfolgt kein physisches I/O, diese Buffer werden im Memory gefunden.
Physisches I/O

Das Diagramm zeigt die Auswirkungen des physischen I/O bei einem konstanten logischen Read (Consistent Gets + Block Gets).

Je weniger Informationen im Buffer Cache gefunden werden, umso mehr physisches I/O wird generiert, um die Daten aus der Datenbank zu lesen.
Library Cache Hit %

Der Library Cache ist Teil des Shared Pools.

Cache Misses im Library Cache sind «sehr teuer», da das SQL-Statement geladen, gepeichert und ausgeführt werden muss. Hier gilt die Regel, dass 99 % aller SQL-Statements in gepeicherter Form im Memory vorliegen müssen. Ist dies nicht der Fall so muss der Wert SHARED_POOL_SIZE erhöht werden.

Berechnung

\[
\text{HitRatio} \% = \left( 1 - \frac{\text{SUM (Reloads)}}{\text{SUM (Pins)}} \right) \times 100
\]

SQL-Statement

\[
\text{select } (1-(\text{sum(reloads)/sum(pins)})) \times 100 \text{ from v$librarycache};
\]

Data Dictionary Cache Hit %


Berechnung

\[
\text{HitRatio} \% = \left( 1 - \frac{\text{SUM (Getmisses)}}{\text{SUM (Gets)}} \right) \times 100
\]

SQL-Statement

\[
\text{select } (1- (\text{sum(getmisses)/sum(gets)})) \times 100 \\
\text{from v$rowcache};
\]

Memory Sort Hit %

Anteil der Sort Operationen im Memory:

Berechnung

\[
\text{HitRatio} \% = \left( \frac{\text{Sorts (Memory)}}{\text{Sorts (Memory) + Sorts(Disk)}} \right) \times 100
\]

SQL-Statement

\[
\text{select } (\text{sum(decode(name, 'sorts (memory)', value, 0)}) / \\
(\text{sum(decode(name, 'sorts (memory)', value, 0)}) + \\
\text{sum(decode(name, 'sorts (disk)', value, 0)))) \times 100 \\
\text{from v$systats};
\]
Rollback Nowait %

Das Diagramm Rollback Nowait Hit % zeigt die Hits und Misses für die Online Rollback Segmente. Ist dieser Wert zu gross, so müssen mehr Rollbacksegmente erstellt werden.

Berechnung

\[
\text{HitRatio} \% = \left( \frac{\text{Sum(Gets)} - \text{Sum(Waits)}}{\text{Sum(Gets)}} \right) \times 100
\]

SQL-Statement

\[
\text{select } \left( \frac{\text{sum(gets)} - \text{sum(waits)}}{\text{sum(gets)}} \right) \times 100 \text{ from v$rollstat;}
\]

Rollback Segment Waits

Rollback Segment Waits können einfach aus v$waitstat gelesen werden.

\[
\text{SELECT * from v$waitstat;}
\]

<table>
<thead>
<tr>
<th>CLASS</th>
<th>COUNT</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>data block</td>
<td>78</td>
<td>147</td>
</tr>
<tr>
<td>sort block</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>save undo block</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>segment header</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>save undo header</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>free list</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>system undo header</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>system undo block</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>undo header</td>
<td>124</td>
<td>139</td>
</tr>
<tr>
<td>undo block</td>
<td>13</td>
<td>3</td>
</tr>
</tbody>
</table>


Rollback Segments Shrinks

Rollbacksegmente sollten nicht dauernd wachsen und wieder kleiner werden, um den OPTIMAL Parameter einzuhalten. Dies kann mit dem folgenden Query kontrolliert werden. EXTENTS und SHRINKS sollten keine auftreten, sonst muss der Parameter OPTIMAL angepasst werden.

\[
\text{SELECT name, extents, rssize, writes, xacts, gets, waits, extends, shrinks, aveactive from v$rollstat stat, v$rollname name WHERE stat.usn = name.usn AND status = 'ONLINE';}
\]

Keine Extents und Shrinks = OK
**File I/O Rate**

Das File I/O Rate Diagramm zeigt die Anzahl physical reads und writes pro Sekunde der Oracle Datenbankfiles der gesamten Instance.

**SQL-Statement**

```sql
select sum(phyrds), sum(phywrts) from vfilestat;
```

**System I/O Rate**

Das System I/O Rate Diagramm zeigt die Anzahl logischen und physikalischen Reads sowie die Anzahl Blockänderungen pro Sekunde.

**SQL-Statement**

```sql
select (sum(decode(name,'db block gets', value,0)) +
  sum(decode(name,'consistent gets', value,0))),
sum(decode(name,'db block changes', value,0)),
sum(decode(name,'physical reads', value,0))
from vsysstat;
```

Logical Reads/s (grün)

Db Block Changes / s (blau)

Phys. Reads / s (rot)

Logical Reads = Db Block Gets + Consistent Gets
**Throughput**
Das Diagramm zeigt die Anzahl User Calls und Transaktionen pro Sekunde der gesamten Instance.

**SQL-Statement**
```
select sum(decode(name,'user commits', value,0)),
       sum(decode(name,'user calls', value,0))
from v$sysstat;
```

**Active Users**
Das Diagramm «Active Users» zeigt die Anzahl aktive User Sessions, welche die Datenbank Instance zur Zeit benutzen.

**SQL-Statement**
```
select count(*)
from v$session
where status='ACTIVE'
  and type!= 'BACKGROUND';
```
**Circuit**

Das Diagramm «Circuit» zeigt die virtuellen Verbindung zur Datenbank Instance via Dispatcher und Shared Serverprozess, welcher den User Process verarbeitet.

### SQL-Statement

```sql
select rawtohex(c.circuit),d.name,s.sid,s.serial#,c.status,c.queue,c.bytes
from v$circuit c,v$dispatcher d,v$shared_server sl,v$session s
where c.dispatcher = d.paddr(+)
  and c.server = sl.paddr(+)
  and c.saddr = s.saddr(+)
order by c.circuit;
```

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>NAME</th>
<th>SID</th>
<th>SERIAL#</th>
<th>STATUS</th>
<th>QUEUE</th>
<th>BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>C387F38</td>
<td>D002</td>
<td>105</td>
<td>3180</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1601</td>
</tr>
<tr>
<td>C387536C</td>
<td>D000</td>
<td>27</td>
<td>9</td>
<td>NORMAL</td>
<td>NONE</td>
<td>95192596</td>
</tr>
<tr>
<td>C38757A0</td>
<td>D000</td>
<td>29</td>
<td>3</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1720467</td>
</tr>
<tr>
<td>C3875B04</td>
<td>D000</td>
<td>84</td>
<td>9198</td>
<td>NORMAL</td>
<td>NONE</td>
<td>40250056</td>
</tr>
<tr>
<td>C3876008</td>
<td>D002</td>
<td>108</td>
<td>7569</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1665</td>
</tr>
<tr>
<td>C3876B07</td>
<td>D001</td>
<td>32</td>
<td>15965</td>
<td>NORMAL</td>
<td>NONE</td>
<td>116787</td>
</tr>
<tr>
<td>C3876C04</td>
<td>D002</td>
<td>93</td>
<td>2617</td>
<td>NORMAL</td>
<td>NONE</td>
<td>2643</td>
</tr>
<tr>
<td>C3877008</td>
<td>D002</td>
<td>98</td>
<td>16075</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1603</td>
</tr>
<tr>
<td>C3877500C</td>
<td>D001</td>
<td>112</td>
<td>28224</td>
<td>NORMAL</td>
<td>NONE</td>
<td>149265</td>
</tr>
<tr>
<td>C3877900</td>
<td>D000</td>
<td>31</td>
<td>40294</td>
<td>NORMAL</td>
<td>NONE</td>
<td>4666</td>
</tr>
<tr>
<td>C3877D04</td>
<td>D001</td>
<td>11</td>
<td>22751</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1603</td>
</tr>
<tr>
<td>C3878000</td>
<td>D000</td>
<td>100</td>
<td>17533</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1603</td>
</tr>
<tr>
<td>C387850C</td>
<td>D001</td>
<td>87</td>
<td>2306</td>
<td>NORMAL</td>
<td>NONE</td>
<td>2644</td>
</tr>
<tr>
<td>C3878E04</td>
<td>D002</td>
<td>23</td>
<td>23536</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1603</td>
</tr>
<tr>
<td>C387960C</td>
<td>D001</td>
<td>65</td>
<td>21222</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1601</td>
</tr>
<tr>
<td>C3879F14</td>
<td>D002</td>
<td>102</td>
<td>20336</td>
<td>NORMAL</td>
<td>NONE</td>
<td>61048</td>
</tr>
<tr>
<td>C387A001</td>
<td>D001</td>
<td>7</td>
<td>21894</td>
<td>NORMAL</td>
<td>NONE</td>
<td>2314</td>
</tr>
<tr>
<td>C387A702</td>
<td>D001</td>
<td>68</td>
<td>40001</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1665</td>
</tr>
<tr>
<td>C387ABB0</td>
<td>D000</td>
<td>18</td>
<td>28880</td>
<td>NORMAL</td>
<td>NONE</td>
<td>2149</td>
</tr>
<tr>
<td>C387CAEB</td>
<td>D001</td>
<td>56</td>
<td>3310</td>
<td>NORMAL</td>
<td>NONE</td>
<td>138351</td>
</tr>
<tr>
<td>C387C90C</td>
<td>D002</td>
<td>95</td>
<td>11271</td>
<td>NORMAL</td>
<td>NONE</td>
<td>3475</td>
</tr>
<tr>
<td>C387CD50</td>
<td>D002</td>
<td>69</td>
<td>26258</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1603</td>
</tr>
<tr>
<td>C387D050</td>
<td>D002</td>
<td>74</td>
<td>46523</td>
<td>NORMAL</td>
<td>NONE</td>
<td>44518043</td>
</tr>
<tr>
<td>C387D55B</td>
<td>D000</td>
<td>75</td>
<td>12017</td>
<td>NORMAL</td>
<td>NONE</td>
<td>8195</td>
</tr>
<tr>
<td>C387D990</td>
<td>D001</td>
<td>44</td>
<td>11578</td>
<td>NORMAL</td>
<td>NONE</td>
<td>21736</td>
</tr>
<tr>
<td>C387E050</td>
<td>D002</td>
<td>88</td>
<td>10584</td>
<td>NORMAL</td>
<td>NONE</td>
<td>7911</td>
</tr>
<tr>
<td>C3880C5C</td>
<td>D001</td>
<td>107</td>
<td>3497</td>
<td>NORMAL</td>
<td>NONE</td>
<td>1635</td>
</tr>
<tr>
<td>C3881000</td>
<td>D001</td>
<td>41</td>
<td>62817</td>
<td>NORMAL</td>
<td>NONE</td>
<td>445986</td>
</tr>
<tr>
<td>C388140C</td>
<td>D001</td>
<td>20</td>
<td>37238</td>
<td>NORMAL</td>
<td>NONE</td>
<td>56800</td>
</tr>
</tbody>
</table>

**Dispatcher**

Das Diagramm «Dispatcher» zeigt die Statistiken der Dispatcher Prozesse der Datenbank Instance.

### SQL-Statement

```sql
select name, status, accept, messages, bytes, idle, busy
from v$dispatcher
order by name;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ACC</th>
<th>MESSAGES</th>
<th>BYTES</th>
<th>IDLE</th>
<th>BUSY</th>
</tr>
</thead>
<tbody>
<tr>
<td>D000</td>
<td>WAIT</td>
<td>YES</td>
<td>8980779</td>
<td>515802691</td>
<td>114566779</td>
<td>716941</td>
</tr>
<tr>
<td>D001</td>
<td>WAIT</td>
<td>YES</td>
<td>2080331</td>
<td>75276628</td>
<td>115068281</td>
<td>215441</td>
</tr>
<tr>
<td>D002</td>
<td>WAIT</td>
<td>YES</td>
<td>4203849</td>
<td>147948411</td>
<td>114892365</td>
<td>391356</td>
</tr>
</tbody>
</table>
MTS

Konfiguration und Waiting auf MTS Dispatcher und Server.

COLUMN NAME FORMAT A25 TRUNC
COLUMN NETWORK FORMAT A10 TRUNC
COLUMN VALUE FORMAT A40 TRUNC

REM Actual MTS-Parameters

SELECT NAME, VALUE
FROM V$PARAMETER
WHERE NAME LIKE '%mts%' OR NAME LIKE '%MTS%';

REM Max. Number of Server-Processes

SELECT * FROM V$MTS

REM Waits of Dispatchers

SELECT NETWORK, DECODE(SUM(TOTALQ),0,'not activ',
ROUND(SUM(WAIT)/SUM(TOTALQ),2)) AVG Waits (1/100 s)
FROM V$DISPATCHER D, V$QUEUE Q
WHERE Q.TYPE = 'DISPATCHER'
AND D.PADDR = Q.PADDR
GROUP BY NETWORK;

REM Waits of Shared-Server

SELECT DECODE(TOTALQ,0,'not activ',
ROUND(WAIT/TOTALQ,2)) AVG Waits (1/100 s)
FROM V$QUEUE
WHERE TYPE = 'COMMON'

REM Nbr. Processes actually waiting for a shared server

SELECT QUEUED
FROM V$QUEUE
WHERE TYPE = 'COMMON'

Max. Number of Server-Processes

<table>
<thead>
<tr>
<th>MAXIMUM_CONNECTIONS</th>
<th>SERVERS_STARTED</th>
<th>SERVERS_TERMINATED</th>
<th>SERVERS_HIGHWATER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>56</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Waits of Dispatchers

<table>
<thead>
<tr>
<th>NETWORK</th>
<th>AVG Waits (1/100 s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipc</td>
<td>,02</td>
</tr>
<tr>
<td>tcp</td>
<td>,03</td>
</tr>
</tbody>
</table>

Waits of Shared-Server

<table>
<thead>
<tr>
<th>AVG Waits (1/100 s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>,02</td>
</tr>
</tbody>
</table>

Nbr. Processes actually waiting for a shared server

<table>
<thead>
<tr>
<th>QUEUED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Wenn SERVERS_HIGHWATER den Wert von mts_max_servers (=20) erreicht, so muss mts_max_servers erhöht werden.
File I/O Rate Details

Das Diagramm «File I/O Rate Details» zeigt die Anzahl der physischen Reads und Writes pro Sekunde für jedes einzelne Datenbank File.

SQL-Statement

select substr(NAME,1,30),PHYRDS,PHYWRTS,PHYBLKRD,PHYBLKWRT
from V$DBFILE DF, V$FILESTAT FS
where DF.FILE#=FS.FILE#
order by NAME;

<table>
<thead>
<tr>
<th>NAME</th>
<th>PHYRDS</th>
<th>PHYWRTS</th>
<th>PHYBLKRD</th>
<th>PHYBLKWRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/vg11/rsipp01</td>
<td>1098</td>
<td>545164</td>
<td>1098</td>
<td>545164</td>
</tr>
<tr>
<td>/dev/vg11/rsipp02</td>
<td>12288</td>
<td>151119</td>
<td>55335</td>
<td>151119</td>
</tr>
<tr>
<td>/dev/vg11/rsipp03</td>
<td>734343</td>
<td>1306301</td>
<td>4463897</td>
<td>1317065</td>
</tr>
<tr>
<td>/dev/vg11/rsipp04</td>
<td>1135197</td>
<td>76502</td>
<td>17699546</td>
<td>76502</td>
</tr>
<tr>
<td>/dev/vg11/rsipp05</td>
<td>266195</td>
<td>21536</td>
<td>380626</td>
<td>21536</td>
</tr>
<tr>
<td>/dev/vg11/rsipp06</td>
<td>144729</td>
<td>61222</td>
<td>154062</td>
<td>100771</td>
</tr>
<tr>
<td>/dev/vg11/rsipp07</td>
<td>250033</td>
<td>787023</td>
<td>357896</td>
<td>909131</td>
</tr>
<tr>
<td>/dev/vg11/rsipp08</td>
<td>161868</td>
<td>823112</td>
<td>1122768</td>
<td>1613132</td>
</tr>
<tr>
<td>/dev/vg11/rsipp09</td>
<td>0</td>
<td>1499</td>
<td>0</td>
<td>1499</td>
</tr>
<tr>
<td>/dev/vg11/rsipp10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>/sipp/d1/ts_system/P18M7S_sys1</td>
<td>16122</td>
<td>31197</td>
<td>28952</td>
<td>31941</td>
</tr>
<tr>
<td>/sipp/d1/ts_system/P18M7S_user</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Free List Hit%


SQL-Statement

select (sum(value)-(sum(count)/2))/sum(value)*100
from v$waitstat w, v$sysstat s
where w.class='free list'
and s.name in ('db block gets', 'consistent gets');

Berechnung

\[
\text{HitRatio} \% = \left( \frac{\text{Sum(Count)}}{\text{Sum(Value)}} \right) \times 100
\]

select (sum(count)) / (sum(value)) * 100
from v$waitstat w, v$sysstat s
where w.class='free list'
and s.name in ('db block gets', 'consistent gets');
**Latch**

Das Diagramm «Latch» statistics zeigt Low-level Locks an shared internen Memorystrukturen.

**SQL-Statement**

```sql
select substr(ln.name,1,30) "Name",l.gets,l.misses,l.sleeps,
       l.immediate_gets "ImmGets",l.immediate_misses "ImmMiss"
from v$latch l, v$latchname ln, v$latchholder lh
where l.latch#=ln.latch#
  and l.addr=lh.laddr(+)
order by l.level#, l.latch#;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>GETS</th>
<th>MISSES</th>
<th>SLEEPS</th>
<th>ImmGets</th>
<th>ImmMiss</th>
</tr>
</thead>
<tbody>
<tr>
<td>process allocation</td>
<td>1151</td>
<td>0</td>
<td>0</td>
<td>1151</td>
<td>0</td>
</tr>
<tr>
<td>session switching</td>
<td>26746</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cached attr list</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>modify parameter values</td>
<td>100971</td>
<td>17</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>virtual circuit buffers</td>
<td>61160911</td>
<td>196</td>
<td>181</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>session allocation</td>
<td>3502845</td>
<td>19</td>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>session idle bit</td>
<td>34993386</td>
<td>451</td>
<td>519</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cache buffers chains</td>
<td>351978424</td>
<td>1138</td>
<td>1338</td>
<td>805523115</td>
<td>2025</td>
</tr>
<tr>
<td>global transaction</td>
<td>1926073</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>global tx hash mapping</td>
<td>724195</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>NLS data objects</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>global tx free list</td>
<td>260731</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>query server process</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cache buffer handles</td>
<td>3545</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>multiblock read objects</td>
<td>4064844</td>
<td>17</td>
<td>18</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>cache buffers lru chain</td>
<td>32308019</td>
<td>1421</td>
<td>2038</td>
<td>29031168</td>
<td>5631</td>
</tr>
<tr>
<td>RKL name table latch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>loader state object freelist</td>
<td>2774</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dll lock allocation</td>
<td>4328384</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>list of block allocation</td>
<td>2346310</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sort extent pool</td>
<td>13760</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>user lock</td>
<td>19189</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enqueue hash chains</td>
<td>26905627</td>
<td>598</td>
<td>686</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>trace latch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cache protection latch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>archive control</td>
<td>653</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lock element parent latch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>row cache objects</td>
<td>7015533</td>
<td>17</td>
<td>29</td>
<td>6391</td>
<td>0</td>
</tr>
<tr>
<td>process queue reference</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enqueues</td>
<td>10738911</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>instance latch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>undo global data</td>
<td>5275435</td>
<td>27</td>
<td>28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>library cache</td>
<td>49455756</td>
<td>6235</td>
<td>4896</td>
<td>6375</td>
<td>9</td>
</tr>
<tr>
<td>library cache load lock</td>
<td>23658</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>error message lists</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>process queue</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>device information</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>redo copy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>query server freelists</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>parallel query alloc buffer</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>redo allocation</td>
<td>14726380</td>
<td>865</td>
<td>1039</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>shared pool</td>
<td>4605571</td>
<td>46</td>
<td>55</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>messages</td>
<td>7238853</td>
<td>615</td>
<td>595</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>system commit number</td>
<td>30237343</td>
<td>281</td>
<td>426</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RKL freelist latch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>transaction allocation</td>
<td>4792886</td>
<td>13</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sequence cache</td>
<td>62029</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cost function</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>virtual circuit queues</td>
<td>58346794</td>
<td>2263</td>
<td>2111</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>virtual circuits</td>
<td>15239348</td>
<td>69</td>
<td>76</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>parallel query stats</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>latch wait list</td>
<td>9097</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ktm global data</td>
<td>4584</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Library Cache Details


SQL-Statement:

```
select namespace, gets, gethits, round(gethitratio*100) "RATIO%", pins, pinhits, round(pinhitratio*100) "RATIO%" 
from v$librarycache 
order by namespace;
```

<table>
<thead>
<tr>
<th>Namespace</th>
<th>GETS</th>
<th>GETHITS</th>
<th>RATIO%</th>
<th>PINS</th>
<th>PINHITS</th>
<th>RATIO%</th>
</tr>
</thead>
<tbody>
<tr>
<td>BODY</td>
<td>7216</td>
<td>7172</td>
<td>99</td>
<td>7216</td>
<td>7166</td>
<td>99</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>171</td>
<td>84</td>
<td>49</td>
<td>159</td>
<td>77</td>
<td>48</td>
</tr>
<tr>
<td>INDEX</td>
<td>852</td>
<td>540</td>
<td>63</td>
<td>534</td>
<td>380</td>
<td>71</td>
</tr>
<tr>
<td>OBJECT</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>PIPE</td>
<td>21167</td>
<td>21166</td>
<td>100</td>
<td>22158</td>
<td>22157</td>
<td>100</td>
</tr>
<tr>
<td>SQL AREA</td>
<td>1910800</td>
<td>1885069</td>
<td>99</td>
<td>11393215</td>
<td>11870605</td>
<td>99</td>
</tr>
<tr>
<td>TABLE/PROCEDURE</td>
<td>228485</td>
<td>220990</td>
<td>97</td>
<td>458154</td>
<td>446343</td>
<td>97</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>2736</td>
<td>2733</td>
<td>100</td>
<td>2736</td>
<td>2729</td>
<td>100</td>
</tr>
</tbody>
</table>

Lock


SQL-Statement:

```
select s.username, s.sid, s.serial#, l.type, l.id1, l.id2, 
from v$lock l, v$session s 
where l.sid=s.sid 
order by s.sid, l.type;
```

Verursacher des Locks

Wartende Session
Wartende Sessions

Die View ROW_LOCK_WAITS zeigt die wartenden Sessions. Dieses Statement ist als View implementiert, es darf keine Rows zurückbringen, da sonst ein User warten muss.

```sql
create or replace view row_lock_waits
    (username, sid, object_owner, object_name, sql_text, file_nr, block_nr, record_nr)
as
select s.username, s.sid, o.owner, o.object_name, a.sql_text,
    s.row_wait_file#, s.row_wait_block#, s.row_wait_row#
from v$session s, v$sqlarea a, dba_objects o
where o.object_id = s.row_wait_obj#
    and s.sql_address = a.address
    and s.row_wait_obj# > 0;

select * from row_lock_waits;
```

Memory Allocated

Das Diagramm «Memory Allocated» zeigt das allozierte Memory in Bytes für die gesamte Instance (alle Sessions).

SQL-Statement

```sql
select /*+ use_nl(n,s) */ sum(value)
from v$statname n, v$sesstat s
where s.statistic# = n.statistic#
    and name = 'session uga memory';
```

Zur Zeit sind 990952 Bytes alloziert

Die Größe der gesamten SGA kann mittels Show SGA im Server Manager angezeigt werden:

```
SVRMGR> show sga
Total System Global Area     114220552 Byte
Fixed Size                       38904 Byte
Variable Size                 50243088 Byte
Database Buffers              62914560 Byte
Redo Buffers                   1024000 Byte
```
Zur Verfügung stehendes Memory

Wenn mehr Memory für die SGA alloziert wird, so muss das aktuell vorhandene, freie Memory verifiziert werden. Dazu eignet sich das Programm top.

```
<table>
<thead>
<tr>
<th>Name</th>
<th>PID</th>
<th>Username</th>
<th>Pri</th>
<th>NI</th>
<th>Size</th>
<th>Res</th>
<th>State</th>
<th>Time</th>
<th>%CPU</th>
<th>%CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>20862</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
<tr>
<td>1788</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
<tr>
<td>20642</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
<tr>
<td>20862</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
<tr>
<td>20862</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
<tr>
<td>20642</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
<tr>
<td>20862</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
<tr>
<td>20862</td>
<td>1248</td>
<td>sippalt</td>
<td>154</td>
<td>28</td>
<td>312K</td>
<td>434K</td>
<td>sleep</td>
<td>0:02</td>
<td>7.76</td>
<td>7.75</td>
</tr>
</tbody>
</table>
```

Freies Memory

![Top-Output]

System: g3i191
Load averages: 0.25, 0.34, 0.36
216 processes: 205 sleeping, 5 running

CPU states:

```
  USER  NICE  SYS  IDLE  BLOCK  SWAIT  INTR  SSYS
0 30.0% 0.0% 14.7% 55.6%  0.0%   0.0%   0.0%   0.0%
1 34.5%  0.0% 12.7% 52.8%  0.0%   0.0%   0.0%   0.0%
--- --- --- --- --- --- --- ---
avg 32.1% 0.0% 13.7% 54.2%  0.0%   0.0%   0.0%   0.0%
```

Memory: 70892K (30128K) real, 80416K (65256K) virtual, 408572K free PageM 1/20
**Network Bytes Rate**

Das Diagramm «Network Bytes Rate» zeigt die Anzahl Bytes / Sekunde an Daten, die vom Datenbank Server und seinen Clients über SQL*Net ausgetauscht werden.

SQL-Statement

```sql
select sum(value) from v$sysstat
where name like 'bytes%SQL*Net%';
```

**Network I/O Rate**

Das Diagramm «Network I/O Rate» zeigt die Anzahl Message Packete / Sekunde die vom Datenbank Server und seinen Clients über SQL*Net ausgetauscht werden.

SQL-Statement

```sql
select sum(total_waits)
from v$system_event
where event like 'SQL*Net%';
```
**Session Events**


**SQL-Statement**

```sql
select w.Sid "Sid", nvl(substr(s.username,1,15),"Background") "User", substr(w.event,1,25) "Event", w.wait_time "Wait Time"
from v$session_wait w, v$session s
where w.sid = s.sid
order by 2,4;
```

<table>
<thead>
<tr>
<th>Sid</th>
<th>User</th>
<th>Event</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Background</td>
<td>pmon timer</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>54</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>Background</td>
<td>smon timer</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Background</td>
<td>rdbms ipc message</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Background</td>
<td>log file parallel write</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>CLIENT</td>
<td>db file scattered read</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>CLIENT</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>CLIENT</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>CLIENT</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>CLIENT</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>CLIENT</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>CLIENT</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>DBSNMP</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>DBSNMP</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>87</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>98</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>88</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>57</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>59</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>95</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>69</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>114</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>EASY2REG</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>78</td>
<td>CPSSSIPPAI</td>
<td>db file scattered read</td>
<td>-1</td>
</tr>
<tr>
<td>64</td>
<td>CPSSSIPPROC</td>
<td>log file sync</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>CPSSSIPPROC</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>CPSSSIPPROC</td>
<td>pipe get</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>CPSSSIPPROC</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>SIPP</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>SIPP</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>SIPP</td>
<td>SQLNet more data from db</td>
<td>0</td>
</tr>
<tr>
<td>106</td>
<td>SIPP</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>75</td>
<td>SIPP</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>SIPP</td>
<td>SQLNet message from clie</td>
<td>18</td>
</tr>
<tr>
<td>112</td>
<td>SYSGADM</td>
<td>SQLNet message from clie</td>
<td>0</td>
</tr>
</tbody>
</table>

Parse Ratio

Das Diagramm «Parse Ratio» zeigt, wie gross der Parsing Aufwand der SQL-Statements ist.

SQL-Statement

```sql
select sum(decode(name,'parse count', value, 0)) / sum(decode(name,'opened cursors cumulative', value, 0))
from v$sysstat;
```

Processes

Das Diagramm «Process» zeigt Informationen zu den Oracle Background und User Prozessen.

SQL-Statement

```sql
select substr(p.pid,1,4) "Pid", p.spid "Spid",
       substr(p.username,1,15) "User",
       decode(s.terminal, NULL, p.terminal, s.terminal) "Terminal",
       decode(s.program, NULL, p.program, s.program) "Program"
from v$process p, v$session s
where p.addr=s.paddr order by pid;
```

Queue

Das Diagramm «Queue» zeigt die Aktivitäten des Multi-Threaded Servers.

SQL-Statement

```sql
select rawtohex(paddr), type, queued, totalq,
       decode(totalq, 0, 0, wait/totalq/100)
from v$queue order by paddr;
```
Redo Alloc Hit%

Das Diagramm «Redo Allocation Hit %» zeigt das Buffer Tuning der Redolog File Aktivitäten. Die Misses dürfen nicht größer als 1 % sein.

Berechnung

\[
\text{HitRatio %} = \frac{\text{Gets} + \text{Immediate Gets}}{\text{Gets} + \text{Immediate Gets} + \text{Misses} + \text{Immediate Misses}} \times 100
\]

SQL-Statement

```
SELECT ((GETS + IMMEDIATE GETS) / (GETS + IMMEDIATE GETS + MISSES + IMMEDIATE MISSES)) * 100
FROM V$LATCH
WHERE NAME = 'redo allocation';
```
### Session
Das Diagramm «Session» zeigt die vorhandenen User Sessions mit ihren SID's, Serial-Nr und PID's.

**SQL-Statement**

```sql
select s.sid, s.serial#, p.pid, s.status, s.username, lockwait,
   decode(s.command, 0, 'NONE', NVL(a.name, 'UNKNOWN'))
from v$session s, v$process p, audit_actions a
where s.paddr = p.addr
   and a.action(+) = s.command
order by s.sid, s.serial#;
```

### Sort Rows Rate
Das Diagramm «Sort Rows Rate» zeigt die Sort Statistiken für die gesamte Instance.

**SQL-Statement**

```sql
select value from v$sysstat
where name = 'sorts (rows)';
```

### Sqlarea
Das Diagramm «SQL Area» zeigt die shared Cursor Informationen im Library Cache.

**SQL-Statement**

```sql
select distinct NVL(username, type) username, sid, sql_text
from v$session, v$sqlarea
where sql_hash_value = hash_value
   and sql_text is not null
order by username;
```

#### SQL of connected sessions

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>SID</th>
<th>SQL_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT</td>
<td>30</td>
<td>select * from OWNERSHIP, IMEI, EQUIPTYPE where OWNERSHIP.imei_id = IMEI.imei_id and IMEI.equip_id = EQUIPTYPE.equip_id and cus_id = '0' order by manufacturer, description</td>
</tr>
<tr>
<td>CLIENT</td>
<td>59</td>
<td>select * from USERS where username = 'kon0'</td>
</tr>
</tbody>
</table>

### System Statistics
Das Diagramm «System Stats» zeigt alle Parameter der wichtigen System Statistiktabelle v$sysstat.

**SQL-Statement**

```sql
select s.name, s.value
from v$sysstat s
order by s.name, s.value;
```

#### NAME

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>background checkpoints completed</td>
<td>888</td>
</tr>
<tr>
<td>background checkpoints started</td>
<td>888</td>
</tr>
<tr>
<td>background timeouts</td>
<td>1001617</td>
</tr>
<tr>
<td>bytes received via SQL*Net from client</td>
<td>883419229</td>
</tr>
<tr>
<td>bytes received via SQL*Net from dblink</td>
<td>1304228985</td>
</tr>
<tr>
<td>bytes sent via SQL*Net to client</td>
<td>4092682369</td>
</tr>
<tr>
<td>bytes sent via SQL*Net to dblink</td>
<td>929074178</td>
</tr>
<tr>
<td>calls to get snapshot scn: kcmgs</td>
<td>13330319</td>
</tr>
<tr>
<td>calls to kcmgs</td>
<td>1382814</td>
</tr>
<tr>
<td>calls to kcmgs</td>
<td>89286</td>
</tr>
<tr>
<td>calls to kcmgs</td>
<td>20090087</td>
</tr>
<tr>
<td>change write time</td>
<td>145527</td>
</tr>
<tr>
<td>cleanouts and rollbacks - consistent read gets</td>
<td>61807</td>
</tr>
<tr>
<td>cleanouts only - consistent read gets</td>
<td>24494</td>
</tr>
<tr>
<td>cluster key scan block gets</td>
<td>875660</td>
</tr>
<tr>
<td>cluster key scans</td>
<td>408030</td>
</tr>
<tr>
<td>commit cleanout failure: write disabled</td>
<td>0</td>
</tr>
<tr>
<td>commit cleanout failures: buffer being written</td>
<td>1217</td>
</tr>
<tr>
<td>commit cleanout failures: callback failure</td>
<td>38</td>
</tr>
</tbody>
</table>

…………………
Auswertung der System Statistik

Aus den Systemstatistiken können wichtige Informationen gewonnen werden. Man beachte dass es sich bei diesen Angaben immer um kumulierten Werte seit dem letzten Startup handelt.

Full Table Scans

<table>
<thead>
<tr>
<th>Scans</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>table scan blocks gotten</td>
<td>61 900 307</td>
</tr>
<tr>
<td>table scan rows gotten</td>
<td>194 6840 695</td>
</tr>
<tr>
<td>table scans (long tables)</td>
<td>13 267</td>
</tr>
<tr>
<td>table scans (short tables)</td>
<td>307 195</td>
</tr>
</tbody>
</table>

Index Scans

<table>
<thead>
<tr>
<th>Scans</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>table fetch by rowid</td>
<td>15 653 655</td>
</tr>
</tbody>
</table>

Redo Waits

<table>
<thead>
<tr>
<th>Scans</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>redo log space requests</td>
<td>1018</td>
</tr>
<tr>
<td>redo log space wait time</td>
<td>21263</td>
</tr>
</tbody>
</table>

Bei grösseren Waits sind die RDO-Files zu vergrössern und er Parameter LOG_BUFFER muss erhöht werden. Die Zeit ist in 1/100 Sekunden angegeben (21263 = 212 Sekunden = 3,5 Minuten in etwa 5 Wochen).
Table Access

Das Diagramm «Table Access» zeigt alle Datenbankobjekte, welche zur Zeit von welcher Session benutzt werden.

**SQL-Statement**

```sql
select sid "Sid", substr(owner,1,15) "Owner",
       substr(object,1,20) "Object"
from v$access
where owner != 'SYS'
order by owner;
```

<table>
<thead>
<tr>
<th>Sid</th>
<th>Owner</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>CLIENT</td>
<td>USERS</td>
</tr>
<tr>
<td>49</td>
<td>CLIENT</td>
<td>USERS</td>
</tr>
<tr>
<td>57</td>
<td>CLIENT</td>
<td>USERS</td>
</tr>
<tr>
<td>92</td>
<td>CLIENT</td>
<td>USERS</td>
</tr>
<tr>
<td>92</td>
<td>CLIENT</td>
<td>MSISDN</td>
</tr>
<tr>
<td>84</td>
<td>CLIENT</td>
<td>USERS</td>
</tr>
<tr>
<td>78</td>
<td>OPS$SIPPALT</td>
<td>ACM</td>
</tr>
<tr>
<td>78</td>
<td>OPS$SIPPALT</td>
<td>SEQ_ORDER_REQ_ORDER_ID</td>
</tr>
<tr>
<td>78</td>
<td>OPS$SIPPALT</td>
<td>MSISDN</td>
</tr>
<tr>
<td>78</td>
<td>OPS$SIPPALT</td>
<td>MESSAGE</td>
</tr>
</tbody>
</table>

Users logged on

Das Diagramm «No. of Users Logged On» zeigt die Anzahl concurrent Users Sessions, unabhängig davon ob sie nun aktiv sind oder nicht.

**SQL-Statement**

```sql
select sessions_current
from v$license;
```

Users running

Das Diagramm «No. of Users Running» zeigt die Users Sessions, welche eine Transaktion ausführen.

**SQL-Statement**

```sql
select count(*)
from v$session_wait
where wait_time!=0;
```

Users waiting

Das Diagramm «No. of Users Waiting» zeigt die User Sessions, die auf einen Event (for whatever reason) warten müssen, um eine Aktion durchzuführen.

**SQL-Statement**

```sql
select substr(w.sid,1,5) "Sid",
       substr(s.username,1,15) "User",
       substr(event,1,30) "Event",
       seconds_in_wait "Wait [s]"
from v$session_wait w, v$session s
where s.sid = w.sid
and state = 'WAITING'
and event not like 'SQL*Net%' 
and event != 'client message'
and event not like '%mon timer'
and event != 'rdbms ipc message'
and event != 'Null Event';
```

<table>
<thead>
<tr>
<th>Sid</th>
<th>User</th>
<th>Event</th>
<th>Wait [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>OPS$SIPPALT</td>
<td>pipe get</td>
<td>48</td>
</tr>
</tbody>
</table>

Users waiting for Locks

Das Diagramm «No. of Users Waiting For Lock» zeigt die User Sessions, die auf die Freigabe eines Locks warten müssen.

**SQL-Statement**

```sql
select count(*)
from v$session
where lockwait is not null;
```
**Object Status**

Um sicher zu sein, dass alle Objekte den Status = VALID haben, kann das folgende Query benutzt werden.

```
/* SQL-Code */
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>OBJECT_NAME</th>
<th>OBJECT_TYPE</th>
<th>STATUS</th>
<th>LAST_DDL_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIPP</td>
<td>ACM</td>
<td>TABLE</td>
<td>VALID</td>
<td>12.05.97 11:19:48</td>
</tr>
<tr>
<td>SIPP</td>
<td>ACM_DATE_CDR1_BI</td>
<td>INDEX</td>
<td>VALID</td>
<td>12.05.97 11:19:48</td>
</tr>
<tr>
<td>SIPP</td>
<td>BINDING</td>
<td>TABLE</td>
<td>VALID</td>
<td>05.06.97 11:38:22</td>
</tr>
<tr>
<td>SIPP</td>
<td>BINDING_IDX1</td>
<td>INDEX</td>
<td>VALID</td>
<td>06.05.97 10:42:08</td>
</tr>
<tr>
<td>SIPP</td>
<td>CDR</td>
<td>TABLE</td>
<td>VALID</td>
<td>11.05.97 07:10:43</td>
</tr>
<tr>
<td>SIPP</td>
<td>CDR_IDX1</td>
<td>INDEX</td>
<td>VALID</td>
<td>30.05.97 16:30:00</td>
</tr>
<tr>
<td>SIPP</td>
<td>CHECK_ACM_MAX</td>
<td>PROCEDURE</td>
<td>VALID</td>
<td>21.05.97 07:00:06</td>
</tr>
<tr>
<td>SIPP</td>
<td>CLIENT_REG</td>
<td>TABLE</td>
<td>VALID</td>
<td>20.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>CREDIT</td>
<td>TABLE</td>
<td>VALID</td>
<td>20.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>CREDIT_IDX1</td>
<td>INDEX</td>
<td>VALID</td>
<td>21.04.97 16:55:24</td>
</tr>
<tr>
<td>SIPP</td>
<td>CREDIT_STATUS_DMI</td>
<td>INDEX</td>
<td>VALID</td>
<td>09.05.97 15:00:14</td>
</tr>
<tr>
<td>SIPP</td>
<td>CUSTOMER</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>CUSTOMER_IDX1</td>
<td>INDEX</td>
<td>VALID</td>
<td>21.04.97 16:55:34</td>
</tr>
<tr>
<td>SIPP</td>
<td>CUSTOMER_IDX2</td>
<td>INDEX</td>
<td>VALID</td>
<td>21.04.97 16:55:34</td>
</tr>
<tr>
<td>SIPP</td>
<td>EQUIP_TYPE</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>ESPRFILE</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>ESPLOG</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>ESRTOTAL</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>EVENTLOG</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>IMEI</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>IMEI_IDX1</td>
<td>INDEX</td>
<td>VALID</td>
<td>21.04.97 16:50:37</td>
</tr>
<tr>
<td>SIPP</td>
<td>LANGUAGE</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>MESSAGE</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>MESSAGE_IDX1</td>
<td>INDEX</td>
<td>VALID</td>
<td>21.04.97 16:50:37</td>
</tr>
<tr>
<td>SIPP</td>
<td>MODULES</td>
<td>TABLE</td>
<td>VALID</td>
<td>28.05.97 10:42:05</td>
</tr>
<tr>
<td>SIPP</td>
<td>MSISDN</td>
<td>TABLE</td>
<td>VALID</td>
<td>05.06.97 11:40:27</td>
</tr>
<tr>
<td>SIPP</td>
<td>MSISDN_IDX1</td>
<td>INDEX</td>
<td>VALID</td>
<td>21.04.97 16:55:57</td>
</tr>
<tr>
<td>SIPP</td>
<td>MSISDN_IDX2</td>
<td>INDEX</td>
<td>VALID</td>
<td>21.04.97 16:57:03</td>
</tr>
</tbody>
</table>
4.2 Detaillierte Analyse des Shared Pools

Das dringendste Problem im Zusammenhang mit dem Shared Pool bildet die Fragmentierung des Memory in kleine «Stücke» im Laufe des Betriebs. Oracle hat verschiedene Massnahmen getroffen, um diese Fragmentierung zu reduzieren:

- Memory wird nicht mehr in grossen Chunks alloziert.
- Reduzieren des Memory Bedarfs für das Per-User Memory (UGA).
- Keeping von grossen PL/SQL Teilen im Memory, um das Flushing von anderen Objekten aus dem Memory zu verhindern.

Jeder Versuch ein grosses PL/SQL Package zu laden, hat zur Folge dass andere Objekte aus dem Pool geflushed werden, was zu einem ORA-4031 Error führen kann.

Objekte die kleiner sind als `SHARED_POOL_RESERVED_MIN_ALLOC`, kommen in den «linken» Teil, Objekte die grösser sind als diesen Parameter kommen in den rechten Teil des Shared Pools.

In der Fixed Tabelle `X$KSMLRU` wird vermerkt, welche Objekte ein Flush anderer Objekte auslösen. Diese Tabelle kann dazu verwendet werden, die Objekte zu identifizieren, welche für die Fragmentierung verantwortlich sind.

<table>
<thead>
<tr>
<th>Column</th>
<th>Header</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ksmlrhnum</td>
<td>&quot;Flushed</td>
<td>SP-Pieces&quot;</td>
</tr>
<tr>
<td>ksmlrcmm</td>
<td>&quot;Comment&quot;</td>
<td></td>
</tr>
<tr>
<td>ksmlrhlon</td>
<td>&quot;Command&quot;</td>
<td></td>
</tr>
<tr>
<td>ksmlrsiz</td>
<td>&quot;Size&quot;</td>
<td></td>
</tr>
</tbody>
</table>

```
SELECT ksmlrhnum, ksmlrcmm, ksmlrhlon, ksmlrsiz
FROM sys.x$ksmlru;
```
Auswertung

<table>
<thead>
<tr>
<th>ksmlrnum</th>
<th>Anzahl Objekte, die aus dem Shared Pool geklasse wurden infolge Memory Allocation des geladenen Objekts. (Flushed SP-Pieces).</th>
</tr>
</thead>
<tbody>
<tr>
<td>ksmlrcom</td>
<td>Allocation Kommentar, welches die Art der Memory Allocation umschreibt. Angaben wie 'MPCODE' oder 'PLSQL%' heissen, dass ein grosses PL/SQL Objekt geladen wurde (Comment).</td>
</tr>
<tr>
<td>ksmlrhor</td>
<td>Name des Objects, das geladen wurde, nur bei PL/SQL und Cursor Objekten (Command).</td>
</tr>
<tr>
<td>ksmlrsiz</td>
<td>Groesse des Contiguous Memorybereichs der alloziert wurde (Size).</td>
</tr>
</tbody>
</table>

Werte > 5K     Start to be a Problem.
Werte > 10K    Serious Problem
Werte > 20K    Very serious Problem

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
<th>Flushed Size SP-Pieces</th>
</tr>
</thead>
<tbody>
<tr>
<td>object info</td>
<td>8664</td>
<td>506</td>
</tr>
<tr>
<td>KGF Slot Table</td>
<td>19528</td>
<td>513</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Im obigen Fall haben wir es also bereits mit einem very serious Problem zu tun !

Beachte !

Diese Tabelle darf nicht mehrmals kurz hintereinander abgefragt werden, da sie immer nur die letzten Angaben enthält. Nach dem Abfragen sind keine Angaben mehr in der Tabelle vorhanden !
## Vorhandene Memory Segmente

Die Tabelle `X$KSMSP` gibt Auskunft über die Memory Segmente und deren Stati im Shared Pool.

**Sum**: Total sum(size) of all memory segments of this type

**State**:
- `recr` = Flushable segment.
- `freeabl` = This segment is in use and can’t be flushed.
- `free` = Free segment
- `perm` = Permanently allocated memory segment

**Max**: This shows the biggest memory segments of this type.

### SQL-Statement zur Auslistung der Memory Segmente

```sql
SELECT SUBSTR(ksmchcom,1,16) ksmchcom,
       DECODE(ksmchcls, 'recr', 'Flushable',
              'freeabl', 'InUse',
              'free', 'Free',
              'perm', 'Permanent',
              ksmchcls) text_ksmsp,
       SUM(ksmchsiz) sum_ksmsp,
       MAX(ksmchsiz) max_ksmsp,
       COUNT(*) count_ksmsp
FROM sys.x$ksmsp
GROUP BY ksmchcom, ksmchcls
ORDER BY ksmchcom, ksmchcls;
```

### ***** Memory Segments and their state in the SHARED POOL **************

<table>
<thead>
<tr>
<th>Description</th>
<th>State</th>
<th>Sum (Size) [Bytes]</th>
<th>Max (Size) [Bytes]</th>
<th>Number of Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>character set m</td>
<td>InUse</td>
<td>32896</td>
<td>15152</td>
<td>9</td>
</tr>
<tr>
<td>dictionary cach</td>
<td>InUse</td>
<td>1376624</td>
<td>4256</td>
<td>814</td>
</tr>
<tr>
<td>fixed allocatio</td>
<td>Free</td>
<td>240</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>free memory</td>
<td>Free</td>
<td>6258376</td>
<td>6216568</td>
<td>743</td>
</tr>
<tr>
<td>free memory</td>
<td>R-free</td>
<td>8000000</td>
<td>8000000</td>
<td>1</td>
</tr>
<tr>
<td>NGL handles</td>
<td>Flushable</td>
<td>1596184</td>
<td>960</td>
<td>5272</td>
</tr>
<tr>
<td>kxfp subheap</td>
<td>Flushable</td>
<td>584</td>
<td>584</td>
<td>1</td>
</tr>
<tr>
<td>kzull</td>
<td>InUse</td>
<td>304</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>library cache</td>
<td>InUse</td>
<td>2545592</td>
<td>584</td>
<td>8643</td>
</tr>
<tr>
<td>library cache</td>
<td>Free</td>
<td>2210656</td>
<td>584</td>
<td>4708</td>
</tr>
<tr>
<td>multiblock rea</td>
<td>InUse</td>
<td>560</td>
<td>280</td>
<td>2</td>
</tr>
<tr>
<td>permanent memor</td>
<td>Permanent</td>
<td>16420872</td>
<td>16420872</td>
<td>1</td>
</tr>
<tr>
<td>PLS cca hp desc</td>
<td>InUse</td>
<td>168</td>
<td>168</td>
<td>1</td>
</tr>
<tr>
<td>PLS non-lib hp</td>
<td>Flushable</td>
<td>2104</td>
<td>2104</td>
<td>1</td>
</tr>
<tr>
<td>PL/SQL DIANA</td>
<td>InUse</td>
<td>1044264</td>
<td>2112</td>
<td>626</td>
</tr>
<tr>
<td>PL/SQL MPCODE</td>
<td>InUse</td>
<td>449968</td>
<td>8400</td>
<td>179</td>
</tr>
<tr>
<td>reserved stoppe</td>
<td>R-freea</td>
<td>48</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>row cache lru</td>
<td>Flushable</td>
<td>1944</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>session param v</td>
<td>InUse</td>
<td>31440</td>
<td>2096</td>
<td>15</td>
</tr>
<tr>
<td>sql area</td>
<td>InUse</td>
<td>7230464</td>
<td>4256</td>
<td>2333</td>
</tr>
<tr>
<td>sql area</td>
<td>Flushable</td>
<td>2559104</td>
<td>4264</td>
<td>808</td>
</tr>
<tr>
<td>table definiti</td>
<td>Flushable</td>
<td>64</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>trigger definiti</td>
<td>Flushable</td>
<td>1080</td>
<td>1080</td>
<td>1</td>
</tr>
<tr>
<td>library cache</td>
<td>InUse</td>
<td>2545592</td>
<td>584</td>
<td>8643</td>
</tr>
<tr>
<td>library cache</td>
<td>Flushable</td>
<td>2210656</td>
<td>584</td>
<td>4708</td>
</tr>
<tr>
<td>multiblock rea</td>
<td>InUse</td>
<td>560</td>
<td>280</td>
<td>2</td>
</tr>
<tr>
<td>permanent memor</td>
<td>Permanent</td>
<td>16420872</td>
<td>16420872</td>
<td>1</td>
</tr>
<tr>
<td>PLS cca hp desc</td>
<td>InUse</td>
<td>168</td>
<td>168</td>
<td>1</td>
</tr>
<tr>
<td>PLS non-lib hp</td>
<td>Flushable</td>
<td>2104</td>
<td>2104</td>
<td>1</td>
</tr>
<tr>
<td>PL/SQL DIANA</td>
<td>InUse</td>
<td>1044264</td>
<td>2112</td>
<td>626</td>
</tr>
<tr>
<td>PL/SQL MPCODE</td>
<td>InUse</td>
<td>449968</td>
<td>8400</td>
<td>179</td>
</tr>
<tr>
<td>PL/SQL MPCODE</td>
<td>Flushable</td>
<td>97344</td>
<td>2104</td>
<td>90</td>
</tr>
<tr>
<td>reserved stoppe</td>
<td>R-freea</td>
<td>48</td>
<td>24</td>
<td>2</td>
</tr>
</tbody>
</table>
**Grosse Objecte lokalisieren**


**Auswertung**

Alle Objekte im Shared Pool welche grösser als 10 KBytes sind:

<table>
<thead>
<tr>
<th>Owner</th>
<th>Name</th>
<th>Type</th>
<th>Kept</th>
<th>Loads</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>STANDARD</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>120,628</td>
</tr>
<tr>
<td>SYS</td>
<td>STANDARD</td>
<td>PACKAGE BODY</td>
<td>NO</td>
<td>1</td>
<td>28,700</td>
</tr>
<tr>
<td>SYS</td>
<td>INDEDEG</td>
<td>PROCEDURE</td>
<td>NO</td>
<td>1</td>
<td>25,188</td>
</tr>
<tr>
<td>SIPP</td>
<td>CHECK_ACM_MAX_TOTAL</td>
<td>PROCEDURE</td>
<td>NO</td>
<td>1</td>
<td>23,505</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_SYS_SQL</td>
<td>PACKAGE BODY</td>
<td>NO</td>
<td>1</td>
<td>22,448</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_SHARED_POOL</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>22,244</td>
</tr>
<tr>
<td>SYS</td>
<td>V$SQLAREA</td>
<td>VIEW</td>
<td>NO</td>
<td>1</td>
<td>19,597</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS.Utility</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>19,592</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_LJQ</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>16,853</td>
</tr>
<tr>
<td>SYS</td>
<td>V$OB_OBJECT_CACHE</td>
<td>VIEW</td>
<td>NO</td>
<td>1</td>
<td>16,629</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_PIPE</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>15,853</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_STANDARD</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>14,521</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_LJQ</td>
<td>PACKAGE BODY</td>
<td>NO</td>
<td>1</td>
<td>13,853</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_OUTPUT</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>13,703</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS.Utility</td>
<td>PACKAGE BODY</td>
<td>NO</td>
<td>1</td>
<td>13,480</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_EXPORT_EXTENSION</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>12,561</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_APPLICATION_INFO</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>12,129</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_SQL</td>
<td>PACKAGE BODY</td>
<td>NO</td>
<td>1</td>
<td>11,900</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_DEFER_IMPORT/Internal</td>
<td>PACKAGE</td>
<td>NO</td>
<td>1</td>
<td>10,798</td>
</tr>
</tbody>
</table>

**Grosse Anonymous PL/SQL Blocks**

Grosse anonymous PL/SQL Blocks sollten in kleine Packages unterteilt werden, welche Funktionen aufrufen.

**Identifikation von grossen anonymous PL/SQL Blocks:**

```sql
COLUMN sql_text FORMAT A60 HEADING 'SQL-Text > 500 Bytes'
SELECT sql_text FROM v$sqlarea
WHERE command_type = 47  -- Command type for anonymous block
AND length(sql_text) > 500;
```
### Grösse des Shared Pools bestimmen

Die Grösse des Shared Pools ist abhängig von der Applikation und kann meist erst während der Produktion festgelegt werden.

![Shared Pool Diagram](image)

**DB-Objects**

Memory im Shared Pool für die Datenbank Objekte wie Packages.

```sql
COLUMN total_mem FORMAT 999,999,999 HEADING 'Space needed for DB-Objects in SP'
SELECT SUM(sharable_mem) total_mem FROM v$db_object_cache;
```

<table>
<thead>
<tr>
<th>Space needed for DB-Objects in SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>836,096</td>
</tr>
</tbody>
</table>

**SQL**

Memory im Shared Pool für normale SQL-Statements.

```sql
COLUMN total_mem FORMAT 999,999,999 HEADING 'Space needed for SQL in SP'
SELECT SUM(sharable_mem) total_mem FROM v$sqlarea;
```

<table>
<thead>
<tr>
<th>Space needed for SQL in SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>25,617,655</td>
</tr>
</tbody>
</table>

**Per-User and Per-Cursor**

Pro concurrent User wird ca 250 Bytes benötigt.

```sql
COLUMN total_mem FORMAT 999,999,999 HEADING 'Space needed for Users in SP'
SELECT SUM(250 * users_opening) total_mem FROM v$sqlarea;
```

<table>
<thead>
<tr>
<th>Space needed for Users in SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,125</td>
</tr>
</tbody>
</table>

**MTS**

Es muss genügend Memory für jeden MTS-User (Multithreaded Server) reserviert werden, der sich über SQL*NET connected.

```sql
COLUMN total_mem FORMAT 999,999,999 HEADING 'Space needed for MTS in SP'
SELECT SUM(value) total_mem
FROM v$sesstat s, v$statname n
WHERE s.statistic# = n.statistic#
AND n.name = 'session uga memory max';
```

<table>
<thead>
<tr>
<th>Space needed for MTS in SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,234,992</td>
</tr>
</tbody>
</table>

Dies ergibt Total DB-Objects + SQL + User + MTS

\[
836,096 + 25,617,655 + 15,125 + 7,234,992 = 33,703,868 \\
\text{Zusätzlich 30 \% Overhead} = 10,111,160 \\
\text{Total} = 40,815,028
\]

Zur Auswertung kann die View V$SHARED_POOL_RESERVED beigezogen werden.

Folgende Attribute der View enthalten Werte nur wenn der INIT.ORA Parameter SHARED_POOL_RESERVED_SIZE gesetzt ist.

```
SQL> desc v$shared_pool_reserved;
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREE_SPACE</td>
<td>is the total amount of free space on the reserved list.</td>
</tr>
<tr>
<td>AVG_FREE_SIZE</td>
<td>is the average size of the free memory on the reserved list.</td>
</tr>
<tr>
<td>FREE_COUNT</td>
<td>is the number of free pieces of memory on the reserved list.</td>
</tr>
<tr>
<td>MAX_FREE_SIZE</td>
<td>is the size of the largest free piece of memory on the reserved list.</td>
</tr>
<tr>
<td>USED_SPACE</td>
<td>is the total amount of used memory on the reserved list.</td>
</tr>
<tr>
<td>AVG_USED_SIZE</td>
<td>is the average size of the of the used memory on the reserved list.</td>
</tr>
<tr>
<td>USED_COUNT</td>
<td>is the number of used pieces of memory on the reserved list.</td>
</tr>
<tr>
<td>MAX_USED_SIZE</td>
<td>is the size of the largest used piece of memory on the reserved list.</td>
</tr>
<tr>
<td>REQUESTS</td>
<td>is the number of times that the reserved list was searched for a free piece of memory.</td>
</tr>
<tr>
<td>REQUEST_FAILURES</td>
<td>is the number of times no memory was found to satisfy a request (e.g., number of times ORA-4031 occurred).</td>
</tr>
<tr>
<td>LAST_FAILURE_SIZE</td>
<td>is the request size of the last failed request (e.g., the request size of last ORA-4031).</td>
</tr>
<tr>
<td>ABORTED_REQUEST_THRESHOLD</td>
<td>is the minimum size of a request which will signal an ORA-4031 error without flushing objects. See the procedure aborted_request_threshold described above.</td>
</tr>
<tr>
<td>LAST_ABORTED_SIZE</td>
<td>is the last size of the request which returned an ORA-4031 error without flushing objects from the LRU list.</td>
</tr>
</tbody>
</table>

Folgende Attribute haben immer einen definierten Wert, auch wenn der INIT.ORA Parameter SHARED_POOL_RESERVED_SIZE nicht gesetzt ist.

```
SQL> desc v$shared_pool_reserved;
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_FAILURES</td>
<td>is the number of times no memory was found to satisfy a request (e.g., number of times ORA-4031 occurred).</td>
</tr>
<tr>
<td>LAST_FAILURE_SIZE</td>
<td>is the request size of the last failed request (e.g., the request size of last ORA-4031).</td>
</tr>
<tr>
<td>ABORTED_REQUEST_THRESHOLD</td>
<td>is the minimum size of a request which will signal an ORA-4031 error without flushing objects. See the procedure aborted_request_threshold described above.</td>
</tr>
<tr>
<td>LAST_ABORTED_SIZE</td>
<td>is the last size of the request which returned an ORA-4031 error without flushing objects from the LRU list.</td>
</tr>
</tbody>
</table>
Tuning Tips Folgende Tuning Tips können angewendet werden:

- Initiale Größe von SHARED_POOL_RESERVED_SIZE:
  Dieser Wert kann zu Beginn des Tunings auf 10 % des Werts von SHARED_POOL_SIZE gesetzt werden.

- Initiale Größe von SHARED_POOL_RESERVED_MIN_ALLOC
  Dieser Wert kann zu Beginn des Tunings auf den Default Wert (5K) gesetzt werden. Wird dieser Wert erhöht, so wird weniger Memory im Bereich des reservierten Teils des Memory (= Reserved List) alloziert, dafür mehr im normalen Shared Pool.

- Tuning von SHARED_POOL_RESERVED_SIZE:
  Die Reserved List sollte so gross gemacht werden, damit keine Objecte aus dem Pool geflushed werden müssen. Im Idealfall muss REQUEST_MISS = 0 angestrebt werden. Auf keine Fall darf REQUEST_MISS im Laufe der Zeit kontinuierlich zunehmen, ansonsten wird ein ORA-4031 Fehler eintreten.

- Ist SHARED_POOL_RESERVED_SIZE zu klein ?
  Sobald der Wert von REQUEST_FAILURES > 0 ist, ist die Reserved List zu klein und muss vergrössert werden.

- Ist SHARED_POOL_RESERVED_SIZE zu gross ?
  Es macht auch keinen Sinn die Reserved List zu gross zu machen. Dies erkennt man wenn REQUEST_MISS = 0 und FREE_MEMORY > 50% von SHARED_POOL_RESERVED_SIZE ist.

- Ist SHARED_POOL_SIZE zu klein ?
  Dies ist der Fall wenn REQUEST_FAILURES > 0 und LAST_FAILURE_SIZE < SHARED_POOL_RESERVED_MIN_ALLOC ist. In diesem Fall muss der Shared Pool vergrössert werden.
Überwachung des Shared Pool

Die in diesem Kapitel vorgestellten Regeln verpackt man am besten in eine Stored Procedure, welche einmal pro Tag gestartet wird. Damit können ORA-4031 Fehler zwar nicht verhindert werden, eine Früherkennung der Fragmentierung ist damit aber möglich.

Stored Procedure `check_shared_pool()`

```sql
CREATE OR REPLACE Procedure CHECK_SHARED_POOL IS

/* Local variables and declarations */

l_fhdl utl_File.File_Type;
l_fdirl v$parameter.value%Type;
l_spsl v$parameter.value%Type;
l_sprsl v$parameter.value%Type;
l_spmal v$parameter.value%Type;
l_fnamel VARCHAR2(30);
l_fmode VARCHAR2(1) := 'W';
l_opentime VARCHAR2(30);
l_current_time VARCHAR2(30);
l_global_name VARCHAR2(30);
l_sum_flush NUMBER(15) := 0;
l_sum_inuse NUMBER(15) := 0;
l_sum_free NUMBER(15) := 0;
l_sum_count NUMBER(15) := 0;
l_size NUMBER(15) := 0;
l_sum_mts NUMBER(15) := 0;
l_sum_obj NUMBER(15) := 0;
l_sum_sql NUMBER(15) := 0;
l_sum_users NUMBER(15) := 0;
l_count INTEGER := 0;
l_message VARCHAR2(100);
l_updt_status NUMBER(1) := 0;

TYPE opstsp_TR IS TABLE OF x$kghlu%ROWTYPE
INDEX BY BINARY_INTEGER;
l_kghlul opstsp_TR;

/* Show memory segments and their state in the shared pool */
CURSOR ksmsp_cur IS
SELECT SUBSTR(ksmchcom,1,16) ksmchcom,
DECODE(ksmchcls,
'recr', 'Flushable',
'freeabl', 'InUse',
'free', 'free',
'perm', 'Permanent',
ksmchcls) text_ksmsp,
SUM(ksmchsiz) sum_ksmsp,
MAX(ksmchsiz) max_ksmsp,
COUNT(*) count_ksmsp
FROM sys.x$ksmsp
GROUP BY ksmchcom,ksmchcls
ORDER BY ksmchcom,ksmchcls;

/* Large objects in shared pool (size > 10 KBytes) */
CURSOR big_objects_cur IS
SELECT owner, name, type, kept, loads, sharable_mem
FROM v$db_object_cache
WHERE sharable_mem > 10000
ORDER BY sharable_mem desc;

/* Track allocations in the shared pool that cause */
/* other objects in the shared pool to flushed out */
```
CURSOR flushed_obj_cur IS
SELECT NVL(ksmlrhon,'Unknown') ksmlrhon,
       NVL(ksmlrcom,'Not defined') ksmlrcom,
       ksmlrsiz,ksmlrnum
FROM sys.x$ksmlru;
/* Tuning of SHARED_POOL_RESERVED_SIZE */
CURSOR spc_cur IS
SELECT * FROM v$shared_pool_reserved;
BEGIN
/* Open logfile */
BEGIN
SELECT value
INTO l_fdir
FROM v$parameter
WHERE name = 'utl_file_dir';
IF (l_fdir IS NULL) THEN
  RAISE NO_DATA_FOUND;
END IF;
l_frame := 'chksp_ || TO_CHAR(SYSDATE,'DDMMYYYY')';
l_fhd := utl_File.FOpen(l_fdir,l_frame,l_fmode);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE NO_DATA_FOUND;
END;
/* Report Header */
SELECT TO_CHAR(TO_DATE(open_time,'MM/DD/YY HH24:MI:SS'),
              'DD.MM.YYYY HH24:MI')
INTO l_open_time, l_current_time
FROM v$thread;
SELECT global_name
INTO l_global_name
FROM global_name;
utl_File.Put_Line(l_fhd,'------------------------- Shared Pool statistics -----------------------------');
utl_File.Putf(l_fhd,'%s - %s
',l_open_time,l_current_time);
utl_File.Putf(l_fhd,'%s
',l_global_name);
utl_File.Put_Line(l_fhd,'');
/* Current size of the SHARED POOL */
SELECT value INTO l_sps
FROM v$parameter
WHERE name = 'shared_pool_size';
SELECT value INTO l_sprs
FROM v$parameter
WHERE name = 'shared_pool_reserved_size';
SELECT value INTO l_sprma
FROM v$parameter
WHERE name = 'shared_pool_reserved_min_alloc';
utl_File.Put_Line(l_fhd,'*********************** Current Size of SHARED POOL ***************************');
utl_File.Putf(l_fhd,'
Current size of SHARED_POOL_SIZE .......... = %s [Bytes]
',LPAD(l_sps,10,' '));
utl_File.Putf(l_fhd,'
Current size of SHARED_POOL_RESERVED_SIZE .... = %s [Bytes]
',LPAD(l_sprs,10,' '));
utl_File.Putf(l_fhd,'
Current size of SHARED_POOL_RESERVED_MIN_ALLOC = %s [Bytes]
',LPAD(l_sprma,10,' '));
How much space is used in the SHARED POOL?

One very difficult judgement that needs to be make in oracle7 is to determine the proper size of the shared pool. The following provides some guidelines for this. It should be emphasized that these are just guidelines, there are no hard and fast rules here and experimentation will be needed to determine a good value.

The shared pool size is highly application dependent. To determine the shared pool size that will be needed for a production system it is generally necessary to first develop the application and run it on a test system and take some measurements. The test system should be run with a very large value for the shared pool size to make the measurements meaningful.

1) OBJECTS STORED IN THE DATABASE

The amount of shared pool that needs to be allocated for objects that are stored in the database like packages and views is easy to measure. You can just measure their size directly with the following statement. This is especially effective because all large pl/sql object should be ’kept’ in the shared pool at all times.

```
SELECT SUM(sharable_mem) INTO l_sum_obj
FROM v$db_object_cache;
```

2) SQL-STATEMENTS

The amount of memory needed to store sql statements in the shared pool is more difficult to measure because of the needs of dynamic sql. If an application has no dynamic sql then the amount of memory can simply be measured after the application has run for a while by just selecting it out of the shared pool as follows:

```
SELECT SUM(sharable_mem) INTO l_sum_sql
FROM v$sqlarea;
```

3) PER-USER PER-CURSOR MEMORY

You will need to allow around 250 bytes of memory in the shared pool per concurrent user for each open cursor that the user has whether the cursor is shared or not. During the peak usage time of the production system, you can measure this as follows:

```
SELECT SUM(250 * users_opening) INTO l_sum_users
FROM v$sqlarea;
```

4) MTS (MULTI-THREADED SERVER)

If you are using multi-threaded server, then you will need to allow enough memory for all the shared server users to put their session memory in the shared pool. This can be measured with the following query:

```
SELECT SUM(value) INTO l_sum_mts
FROM v$sesstat s, v$statname n
WHERE s.statistic# = n.statistic#
AND n.name = 'session uga memory max';
```
**Oracle Datenbank Tuning**

Ausgabe 5. Januar 1995 Seite -75

---

```
orig_file.Put(l_fdh,'//';
orig_file.Put_Line(l_fdh,'************** Used space in SHARED POOL (approximative) **************');
orig_file.Put_Line(l_fdh,'');
orig_file.Put(l_fdh,'Total size for Objects ...... = %s [Bytes]
   LPAD(TO_CHAR(l_sum_obj),10,' '));
orig_file.Put(l_fdh,'Total size for SQL-Statements = %s [Bytes]
   LPAD(TO_CHAR(l_sum_sql),10,' '));
orig_file.Put(l_fdh,'Total size for User Sessions  = %s [Bytes]
   LPAD(TO_CHAR(l_sum_users),10,' '));
orig_file.Put(l_fdh,'Total size for MTS .......... = %s [Bytes]
   LPAD(TO_CHAR(l_sum_mts),10,' '));
orig_file.Put(l_fdh,'Total size used ............. = %s [Bytes]
   LPAD(TO_CHAR(l_sum_obj+l_sum_sql+l_sum_users+l_sum_mts),10,' '));
/* ----------------------------------------------------------------
Show memory segments in the shared pool and their state
Sum:   Total sum(size) of all memory segments of this type
State: recr  = Flushable segment.
freeabl = This segment is in use and can’t be flushed.
free    = Free segment
perm    = Permanently allocated memory segment
Max:   This shows the biggest memory segments of this type.
----------------------------------------------------------------- */
orig_file.Put(l_fdh,'//');
orig_file.Put_Line(l_fdh,'************* Memory Segments and their state in the SHARED POOL **************');
orig_file.Put_Line(l_fdh,'');
orig_file.Put_Line(l_fdh,'Description State                Sum (Size)      Max (Size)      Number of Segments');
orig_file.Put_Line(l_fdh,'---------------- ------------- ----------------- --------------- --------------');
FOR ksmsp_rec IN ksmsp_cur LOOP
    orig_file.Put(l_fdh,'%s%s%s%s%s
   RPAD(ksmsp_rec.ksmchcom,17,''),
   RPAD(ksmsp_rec.text_ksmsp,14,''),
   LPAD(TO_CHAR(ksmsp_rec.sum_ksmsp),17,''),
   LPAD(TO_CHAR(ksmsp_rec.max_ksmsp),16,''),
   LPAD(TO_CHAR(ksmsp_rec.count_ksmsp),15,' '));
    IF (ksmsp_rec.text_ksmsp = 'Flushable') THEN
        l_sum_flush := l_sum_flush + ksmsp_rec.sum_ksmsp;
    ELSIF (ksmsp_rec.text_ksmsp = 'InUse') THEN
        l_sum_inuse := l_sum_inuse + ksmsp_rec.sum_ksmsp;
    ELSIF (ksmsp_rec.text_ksmsp = 'Free') THEN
        l_sum_free := l_sum_free + ksmsp_rec.sum_ksmsp;
    ELSIF (ksmsp_rec.text_ksmsp = 'Permanent') THEN
        l_sum_perm := l_sum_perm + ksmsp_rec.sum_ksmsp;
    END IF;
END LOOP;
/* -------------------------------------------------------------------
Number and average size of free memory segments in the shared pool.
The number of free memory segments should be small, but the average
size shuld be big. If these is not the case, the shared pool is
fragmented and we will get the ORA-4031 Error soon.
--------------------------------------------------------------------- */
SELECT COUNT(*), AVG(ksmchsiz)
INTO l_sum_count, l_size
FROM sys.x$ksmsp
WHERE ksmchcom = 'free memory';
```
utl_File.Put(l_fhd,'Average size of free memory segments .. = %s [Bytes]\n', LPAD(TO_CHAR(l_size),10, ' '));

utl_File.Put(l_fhd,'Number free memory segments ........ = %s\n', LPAD(TO_CHAR(l_sum_count),10, ' '));

utl_File.Put(l_fhd,'Average size / number free segments ... = %s\n', LPAD(TO_CHAR(l_size/l_sum_count),10, ' '));

utl_File.Put_File(l_fhd,'----------------------------------');

utl_File.Put(l_fhd,'\\n');

/* Large objects in shared pool (size > 10 KBytes) */

utl_File.Put_Line(l_fhd,'************** Big objects in the SHARED POOL (size > 10 KBytes) **************');

utl_File.Put_Line(l_fhd,'Owner Name Type Kept Loads Bytes');

utl_File.Put_Line(l_fhd,'-------- ------------------ ------------- ---- ----- --------------');

FOR big_objects_rec IN big_objects_cur LOOP
    utl_File.Put(l_fhd,'%s%s%s%s%s', RPAD(big_objects_rec.owner,9,' '), RPAD(big_objects_rec.name,31,' '), RPAD(big_objects_rec.type,14,' '), RPAD(big_objects_rec.kept,5,' '), LPAD(TO_CHAR(big_objects_rec.loads),5,' '));

    utl_File.Put(l_fhd,'%s', LPAD(TO_CHAR(big_objects_rec.sharable_mem),15,' '));

END LOOP;

/* Summary statistic of the shared pool operations */

utl_File.Put(l_fhd,'\\n');

utl_File.Put_Line(l_fhd,'*************** Summary statistic of the SHARED POOL operations ***************');

utl_File.Put_Line(l_fhd,'LRU-Operations Flushed Recurrent Segments Transient Segments');

utl_File.Put_Line(l_fhd,'----------------------- ------------------ ------------------- ------------------');

FOR ksmlru_rec IN (SELECT * FROM x$kghlu) LOOP
    l_count := l_count + 1;
    l_kghlu(l_count) := ksmlru_rec;

    utl_File.Put(l_fhd,'%s%s%s%s', LPAD(TO_CHAR(l_kghlu(l_count).kghluops),23,' '), LPAD(TO_CHAR(l_kghlu(l_count).kghlufsh),17,' '), LPAD(TO_CHAR(l_kghlu(l_count).kghlurcr),20,' '), LPAD(TO_CHAR(l_kghlu(l_count).kghlutrn),19,' '));

    utl_File.Put(l_fhd,'\n');

END LOOP;

/* Track allocations in the shared pool that cause */
/* other objects in the shared pool to flushed out */

utl_File.Put(l_fhd,'\n');

utl_File.Put_Line(l_fhd,'***** Allocations that cause other objects in the SHARED POOL to flushed out *****');

utl_File.Put_Line(l_fhd,'Command Comment Size Obj flushed');

utl_File.Put_Line(l_fhd,'-------------------------------- -------------------- ---------- --------------');

l_count := 0;
l_message := NULL;

FOR flushed_obj_rec IN flushed_obj_cur LOOP
    IF (flushed_obj_rec.ksmlrnum > 0) THEN
        l_count := l_count + 1;
        utl_File.Put(l_fhd,'%s%s%s%s', RPAD(flushed_obj_rec.ksmlrhon,33,' '), RPAD(flushed_obj_rec.ksmlrcom,21,' '), LPAD(flushed_obj_rec.ksmlrsiz,10,' '), LPAD(flushed_obj_rec.ksmlrnum,15,' '));

        IF (flushed_obj_rec.ksmlrsiz >= 5000 AND flushed_obj_rec.ksmlrsiz < 10000) THEN
            l_message := 'ORA-4031 will start soon !';
            l_updt_status := 1;
        END IF;
    END IF;

END LOOP;
ELSIF (flushed_obj_rec.kmlrsize > 10000
AND flushed_obj_rec.kmlrsize < 20000) THEN
  l_message := 'ORA-4031 will be a serious problem soon!';
  l_updt_status := 2;
ELSIF (flushed_obj_rec.kmlrsize > 20000) THEN
  l_message := 'ORA-4031 will now starting!';
  l_updt_status := 3;
END IF;
END IF;
END LOOP;
IF (l_count = 0) THEN
  utl_File.Put_Line(l_fhd,'No objects have been flushed out ... wonderful!');
  l_updt_status := 0;
END IF;
IF (l_message IS NOT NULL) THEN
  utl_File.Put(l_fhd,'%s
', l_message);
END IF;
/* Tuning of SHARED_POOL_RESERVED_SIZE */
utl_File.Put(l_fhd,'
');
utl_File.Put_Line(l_fhd,'********** Tuning of SHARED_POOL_RESERVED_SIZE **********
');
utl_File.Put(l_fhd,'FreeSpace FreeSegm UsedSpace UsedSegm Requests Flushes ORA-4031
[Bytes] Counts [Bytes] Counts Counts Counts
------------------------------- ------------------- ------------------- -------------------
');
FOR spr_rec IN spr_cur LOOP
  utl_File.Put(l_fhd,'%s%s%s%s%s%s
', LPAD(spr_rec.free_space,14,' '),
              LPAD(spr_rec.free_count,10,' '),
              LPAD(spr_rec.used_space,15,' '),
              LPAD(spr_rec.used_count,10,' '),
              LPAD(spr_rec.requests,11,' '),
              LPAD(spr_rec.request_misses,10,' '),
              LPAD(spr_rec.request_failures,9,' '));
  utl_File.Put(l_fhd, '
');
  IF (spr_rec.request_failures > 0) THEN
    l_message := 'Your System has ORA-4031 Errors!';
    l_updt_status := 3;
    utl_File.Put(l_fhd,'%s
', l_message);
  END IF;
  IF (spr_rec.last_failure_size > l_spma
AND spr_rec.max_free_size < l_spma
AND spr_rec.free_space < l_spma) THEN
    l_message := 'Your SHARED_POOL_RESERVED_SIZE seems to bee too small!';
    l_updt_status := 2;
    utl_File.Put(l_fhd,'%s
', l_message);
  END IF;
  IF (spr_rec.request_misses = 0
AND spr_rec.free_space > (l_spma/2)) THEN
    l_message := 'Your SHARED_POOL_RESERVED_SIZE seems to be too big';
    utl_File.Put(l_fhd,'%s
', l_message);
  END IF;
  IF (spr_rec.request_failures > 0
AND spr_rec.last_failure_size < l_spma) THEN
    l_message := 'Your SHARED_POOL_SIZE is really too small!';
    l_updt_status := 3;
    utl_File.Put(l_fhd,'%s
', l_message);
  END IF;
END LOOP;
utl_File.Put(l_fhd,'
');
utl_file.FClose(l_fhd);
/* Logging in table eventlog */

BEGIN
    UPDATE eventlog SET status = l_updt_status
    WHERE event = 'shared_pool';
    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'Check_Shared_Pool(): UPDATE failed for EVENTLOG');
        ROLLBACK;
END;
EXCEPTION
    WHEN OTHERS THEN
        l_message := 'Check_Shared_Pool()': || SUBSTR(SQLERRM,1,100);
        RAISE_APPLICATION_ERROR(-20000, l_message);
        ROLLBACK;
END;
Output der Stored Procedure `check_shared_pool()`:

---

**Shared Pool statistics**

27.05.1997 20:56 - 12.06.1997 08:49

T18M7S.WORLD

---

******************** Current Size of SHARED POOL ********************

Current size of `SHARED_POOL_SIZE` ............. = 41943040 [Bytes]
Current size of `SHARED_POOL_RESERVED_SIZE` .... = 8000000 [Bytes]
Current size of `SHARED_POOL_RESERVED_MIN_ALLOC` = 1000000 [Bytes]

******************** Used space in SHARED POOL (approximative) ********************

Total size for Objects ...... = 1885707 [Bytes]
Total size for SQL-Statements = 15009721 [Bytes]
Total size for User Sessions = 4250 [Bytes]
Total size for MTS .......... = 467992 [Bytes]
Total size used ............. = 17367670 [Bytes]

************* Memory Segments and their state in the SHARED POOL *************

<table>
<thead>
<tr>
<th>Description</th>
<th>State</th>
<th>Sum (Size) [Bytes]</th>
<th>Max (Size) [Bytes]</th>
<th>Number of Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>character set m</td>
<td>InUse</td>
<td>32886</td>
<td>15152</td>
<td>9</td>
</tr>
<tr>
<td>dictionary cach</td>
<td>InUse</td>
<td>1376624</td>
<td>4256</td>
<td>814</td>
</tr>
<tr>
<td>fixed allocation</td>
<td>Flushable</td>
<td>240</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>free memory</td>
<td>Free</td>
<td>6258376</td>
<td>6216568</td>
<td>743</td>
</tr>
<tr>
<td>free memory</td>
<td>R-free</td>
<td>8000000</td>
<td>8000000</td>
<td>1</td>
</tr>
<tr>
<td>PL/SQL DIANA</td>
<td>InUse</td>
<td>1044264</td>
<td>2112</td>
<td>626</td>
</tr>
<tr>
<td>PL/SQL DIANA</td>
<td>Flushable</td>
<td>114392</td>
<td>2104</td>
<td>72</td>
</tr>
<tr>
<td>PL/SQL MPCODE</td>
<td>InUse</td>
<td>449968</td>
<td>8400</td>
<td>179</td>
</tr>
<tr>
<td>PL/SQL MPCODE</td>
<td>Flushable</td>
<td>97344</td>
<td>2104</td>
<td>90</td>
</tr>
<tr>
<td>reserved stoppe</td>
<td>R-freea</td>
<td>48</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>row cache lru</td>
<td>Flushable</td>
<td>1944</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>session param v</td>
<td>InUse</td>
<td>31440</td>
<td>2096</td>
<td>15</td>
</tr>
<tr>
<td>sql area</td>
<td>InUse</td>
<td>7230464</td>
<td>4256</td>
<td>2333</td>
</tr>
<tr>
<td>sql area</td>
<td>Flushable</td>
<td>2559104</td>
<td>4264</td>
<td>808</td>
</tr>
<tr>
<td>trigger definiti</td>
<td>Flushable</td>
<td>64</td>
<td>64</td>
<td>1</td>
</tr>
</tbody>
</table>

Total size of flushable memory segments = 6583696 [Bytes]
Total size of memory segments in use .. = 12712280 [Bytes]
Total size of free memory segments .... = 6258376 [Bytes]
Total size of permanent memory segments = 16420872 [Bytes]
Average size of free memory segments .. = 19182 [Bytes]
Number free memory segments ........... = 743
Average size / number free segments ... = 25,8169582

---
Big objects in the SHARED POOL (size > 10 KBytes)

<table>
<thead>
<tr>
<th>Owner</th>
<th>Name</th>
<th>Type</th>
<th>Kept Loads</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>CHECK_SHARED_POOL</td>
<td>PROCEDURE</td>
<td>NO</td>
<td>140981</td>
</tr>
<tr>
<td>SYS</td>
<td>STANDARD</td>
<td>PACKAGE</td>
<td>NO</td>
<td>120628</td>
</tr>
<tr>
<td>SIPP</td>
<td>CHECK_ACM_MAX_TOTAL</td>
<td>PROCEDURE</td>
<td>NO</td>
<td>34577</td>
</tr>
<tr>
<td>SIPP</td>
<td>SUM_RELOAD</td>
<td>PROCEDURE</td>
<td>NO</td>
<td>34559</td>
</tr>
<tr>
<td>SYS</td>
<td>STANDARD</td>
<td>PACKAGE</td>
<td>NO</td>
<td>28700</td>
</tr>
<tr>
<td>SYS</td>
<td>INDREORG</td>
<td>PROCEDURE</td>
<td>NO</td>
<td>25028</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_IREFRESH</td>
<td>PACKAGE</td>
<td>NO</td>
<td>22713</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_SYS_SQL</td>
<td>PACKAGE</td>
<td>NO</td>
<td>22448</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_SHARED_POOL</td>
<td>PACKAGE</td>
<td>NO</td>
<td>21956</td>
</tr>
<tr>
<td>SYS</td>
<td>V$SQLAREA</td>
<td>VIEW</td>
<td>NO</td>
<td>19741</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_INFORMATION</td>
<td>PACKAGE</td>
<td>NO</td>
<td>19664</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_IREFRESH</td>
<td>PACKAGE</td>
<td>NO</td>
<td>19057</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>SSCORDER</td>
<td>SYNONYM</td>
<td>NO</td>
<td>17351</td>
</tr>
<tr>
<td>SYS</td>
<td>X$KGHLU</td>
<td>TABLE</td>
<td>NO</td>
<td>17163</td>
</tr>
<tr>
<td>SIPP</td>
<td>SEQ_ORDER_REQ_ORD_ID</td>
<td>SEQUENCE</td>
<td>NO</td>
<td>17105</td>
</tr>
<tr>
<td>SIPP</td>
<td>SEQ_ESRLOG_LOG_ID</td>
<td>SEQUENCE</td>
<td>NO</td>
<td>17092</td>
</tr>
</tbody>
</table>

Summary statistic of the SHARED POOL operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Flashes</th>
<th>Recurrent Segments</th>
<th>Transient Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU-Operations</td>
<td>771534</td>
<td>0</td>
<td>2652</td>
</tr>
<tr>
<td>Allocations</td>
<td></td>
<td></td>
<td>5611</td>
</tr>
</tbody>
</table>

Allocations that cause other objects in the SHARED POOL to flushed out

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
<th>Size</th>
<th>Obj flushed</th>
</tr>
</thead>
<tbody>
<tr>
<td>No objects have been flushed out ... wonderful !</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tuning of SHARED_POOL_RESERVED_SIZE

<table>
<thead>
<tr>
<th>FreeSpace</th>
<th>FreeSegm</th>
<th>UsedSpace</th>
<th>UsedSegm</th>
<th>Requests</th>
<th>Flushes</th>
<th>ORA-4031</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bytes]</td>
<td>Counts</td>
<td>[Bytes]</td>
<td>Counts</td>
<td>Counts</td>
<td>Counts</td>
<td>Counts</td>
</tr>
<tr>
<td>8000000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Your SHARED_POOL_RESERVED_SIZE seems to be too big
5. Locking Verhalten

5.1 Row Locks

Row Share Table Locks (RS)

Ein Row Share Table Lock zeigt, dass gewisse Rows in der Table für einen UPDATE gelockt worden sind. Der RS-Lock ist der am wenigsten restriktive Mode, erlaubt also die größte Concurreny.

Beispiele

```
SELECT person_nr FROM mitglied FOR UPDATE OF pult_nr;
SELECT person_nr FROM mitglied FOR UPDATE;
LOCK TABLE mitglied IN EXCLUSIVE MODE;
```

Row Exclusive Table Locks (RX)

Ein Row Exclusive Table Lock zeigt, dass gewisse Rows geändert wurden, aber noch kein COMMIT oder ROLLBACK erfolgt ist. Der RX-Lock ist restriktiver als der RS-Lock.

Beispiele

```
UPDATE mitglied SET name = 'Zahn' WHERE name = 'Eiger';
INSERT INTO table ...
DELETE FROM table ...
LOCK TABLE mitglied in ROW EXCLUSIVE MODE;
```
5.2 Table Locks

Table Locks werden ausschließlich durch explizite Locking Commands ausgelöst und sollten grundsätzlich vermieden werden, da sie das Default Lockingverhalten von Oracle ausschalten.

Oracle kennt folgende Table Locks:

- Share Table Locks (S)
  LOCK TABLE mitglied IN SHARE MODE;
- Share Row Exclusive Table Locks (SRX)
  LOCK TABLE mitglied IN SHARE ROW EXCLUSIVE MODE;
- Exclusive Table Locks (X)
  LOCK TABLE mitglied IN EXCLUSIVE MODE;

5.3 Lock Manager Lock Types

Im Enterprise Manager Lock Manager werden folgende Lock Types benutzt:

<table>
<thead>
<tr>
<th>ID</th>
<th>Lock Type</th>
<th>Shortname</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>Media Recovery</td>
<td>IS</td>
</tr>
<tr>
<td>RT</td>
<td>Redo Thread</td>
<td>FS</td>
</tr>
<tr>
<td>UN</td>
<td>User Name</td>
<td>IR</td>
</tr>
<tr>
<td>TX</td>
<td>Transaction</td>
<td>ST</td>
</tr>
<tr>
<td>TM</td>
<td>Data Manipulation</td>
<td>TS</td>
</tr>
<tr>
<td>UL</td>
<td>PL/SQL User Lock</td>
<td>IV</td>
</tr>
<tr>
<td>DX</td>
<td>Distributed Action</td>
<td>LS</td>
</tr>
<tr>
<td>CF</td>
<td>Control File</td>
<td>RW</td>
</tr>
<tr>
<td>SQ</td>
<td>Sequence Number</td>
<td>TE</td>
</tr>
<tr>
<td>TT</td>
<td>Temp Table</td>
<td></td>
</tr>
</tbody>
</table>

Lock Manager: Lock Types