akadia
Information Technology

Oracle Datenbank Tuning

Author: Martin Zahn

Akadia AG
Information Technology
Arvenweg 4
CH-3604 Thun
Tel 033 335 86 20
Fax 033 335 86 25
E-Mail info@akadia.ch
Web http://www.akadia.ch

Inhaltsverzeichnis

1. Der Tuning Prozess

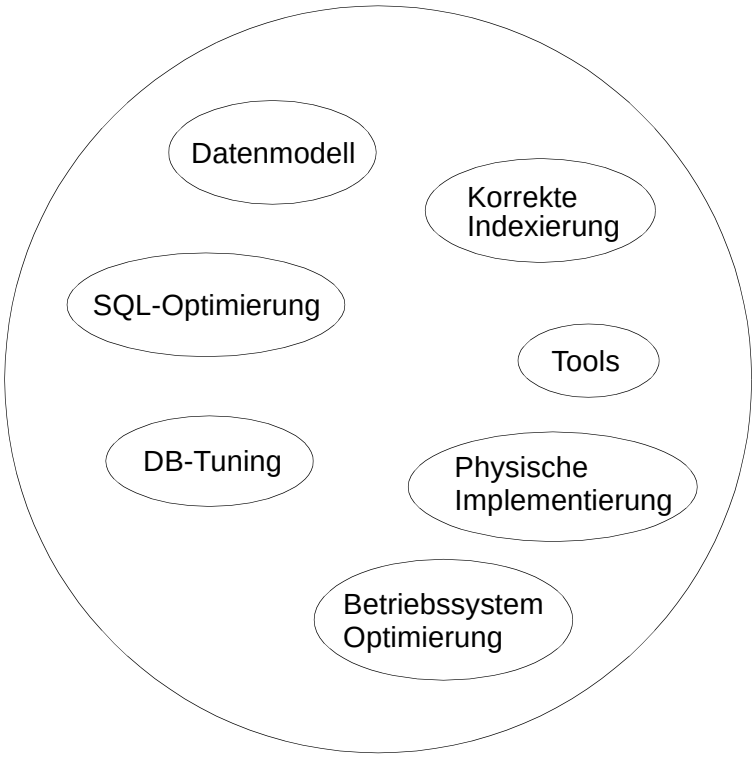
Grundsätzliches

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 erstrekken, 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

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

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).
- Indexe 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).

Die Bedeutung des INIT.ORA Tunings wird oft überschätzt, den grössten Nutzen bezüglich Performance liegt in einem guten Design sowie guter SQL-Programmierung. Messungen zur Performance immer dann vornehmen, wenn die Instance und Datenbank bereits lange im Betrieb waren. Ein DB-Tuning nach dem Startup der DB ist unbrauchbar. Änderungen, die nichts gebracht haben, wieder rückgängig machen. Unbedingt ein Step-by-Step Vorgehen wählen und die Resultate der einzelnen Schritte vergleichen.

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.
- Bei Joins kann über die Join-Attribute nur das Kartesische Produkt verhindert werden. Join-Attribute gelten deshalb nicht als Einschränkung auf Tabellen Ebene.
- 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 vorgegeben Schemas.

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.

Rank	Access Path

1	Single Row by ROWID
	
	Beispiel: empno ist der Primary Key
	
	`SELECT rowid INTO var_rowid`
	`  FROM emp`
	`WHERE empno = 4711`
	`  FOR UPDATE;`
	
	`UPDATE emp SET ename = 'MILLER'`
	`WHERE rowid = var_rowid;`

2	Single Row by Cluster Join
3	Single Row by Hash Cluster Key with Unique or Primary Key

4	Single Row by Unique or Primary Key
	
	Beispiel: empno ist der Primary Key
	
	`SELECT *`
	`  FROM emp`
	`WHERE empno = 7900`

5	Clustered Join
6	Hash Cluster Key
7	Index Cluster Key

8	Composite Index
	
	Beispiel: Composite Index auf (job, deptno)
	
	`SELECT *`
	`  FROM emp`
	`WHERE job = 'CLERK'`
	`  AND depno = 30;`

9	Single Column Indexes
	
	Beispiel: Index auf job
	
	`SELECT *`
	`  FROM emp`
	`WHERE job = 'CLERK';`

10	Bounded Range Search on Indexed Columns
	
	Beispiel: column ist an 1. Stelle in einem Composite Index
	
	`WHERE column = expr`
	`WHERE column >[=] expr AND column <[=] expr`
	`WHERE column BETWEEN expr AND expr`
	`WHERE column LIKE 'ABC%'`

Rank	Access Path

11	Unounded Range Search on Indexed Columns
	
	Beispiel: column ist an 1. Stelle in einem Composite Index
	
	`WHERE column >[=] expr`
	`WHERE column <[=] expr`

12	Sort-Merge Join
13	MAX or MIN of Indexed Columns Beispiel: Auf sal ist ein Index `SELECT MAX(sal) FROM emp;`
14	ORDER BY on Indexed Columns Damit dieser Path benutzt wird müssen folgende Bedingungen erfüllt sein: 1. Die Column der ORDER BY Cluase muss indexiert sein 2. Die Column muss UNIQUE sein. 3. NLS_SORT muss auf BINARY gestellt sein Beispiel: empno ist der Primary Key `SELECT * FROM emp ORDER BY empno;`
14	Full Table Scan

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 ....
```

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

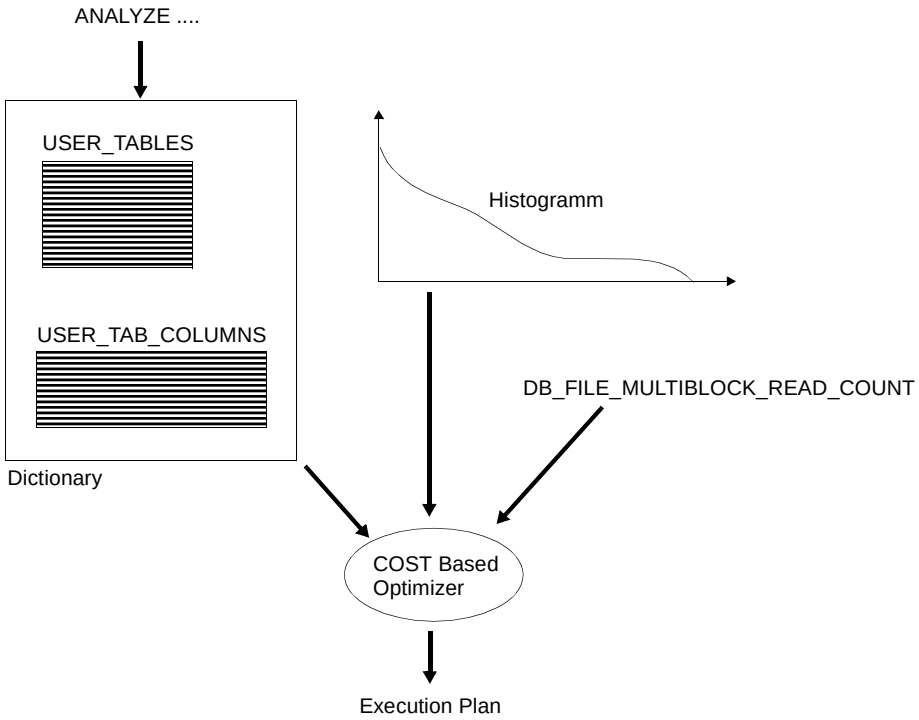
- Der CBO erkennt unnötige Abfragen, wie sie oft von Endanwender Tools generiert werden.
- Umschaltung von «LIKE» auf «=» wenn Wildcard Zeichen fehlen.
- Verwendung von neuen Features wie Hash-Join, Anti-Join etc.
- Entscheidet sich nicht «stur» für einen Indexzugriff wie dies der RULE based Optimizer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

Entscheidungs-grundlagen

Der COST based Optimizer benutzt folgende Entscheidungsgrundlagen:

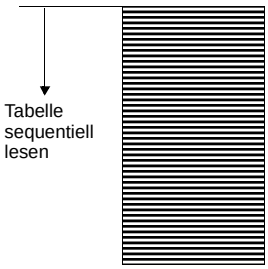


- Statistiken werden im Data Dictionary abgelegt. Sie können eingesehen werden in den Views USER_TABLES und USER_TAB_COLUMNS.
- Berücksichtigung der Selektivität, wenn ein Histogramm zur Verfügung steht. Die Selektivität ist der Anteil an Rows in Prozent, die das Query selektiert. Queries, welche wenig 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 Endresultat 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 werdsen, 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",PHYBLKWRT 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,':'));

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

Zuordnung DB-File und Tablespace

Name	Tablespace	Status	Size (M)	Used (M)
/dev/vg11/rsipp01	RB	ONLINE	990.000	68.285
/dev/vg11/rsipp02	TEMP_PROC	ONLINE	990.000	368.148
/dev/vg11/rsipp03	TAB_SIPP	ONLINE	990.000	57.473
/dev/vg11/rsipp04	TAB_CDR	ONLINE	990.000	387.035
/dev/vg11/rsipp05	TAB_CREDIT	ONLINE	990.000	29.535
/dev/vg11/rsipp06	IDX_SIPP	ONLINE	990.000	195.492
/dev/vg11/rsipp07	IDX_BIG	ONLINE	990.000	385.004
/dev/vg11/rsipp08	TEMP_CLNT	ONLINE	990.000	434.535
/dev/vg11/rsipp09	TAB_CDR	ONLINE	990.000	58.598
/dev/vg11/rsipp10	TAB_CDR	ONLINE	990.000	19.535
/sipp/d1/ts_system/P18M7S_sys1.dbf	SYSTEM	SYSTEM	80.000	13.461
/sipp/d1/ts_system/P18M7S_users1.dbf	USERS	ONLINE	10.000	0.004

Man erkennt, dass der maximale Durchsatz nur auf dem grossen Tablespace TAB_CDR (/dev/vg11/rsipp04, /dev/vg11/rsipp09, /dev/vg11/rsipp10) erreicht wird. Ferner sieht man, dass ein Index Scan diesen Mechanismus nicht verwendet IDX_SIPP und IDX_BIG (/dev/vg11/rsipp06 und /dev/vg11/rsipp07).

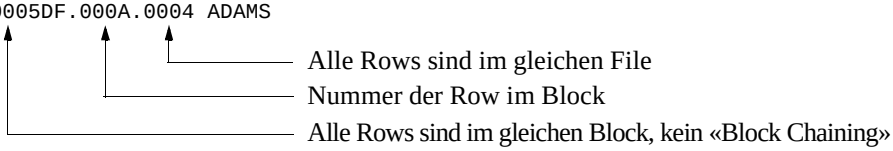
Index Scan

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

Nr des Blocks im File	Nr der Row im Block	Nr des Files

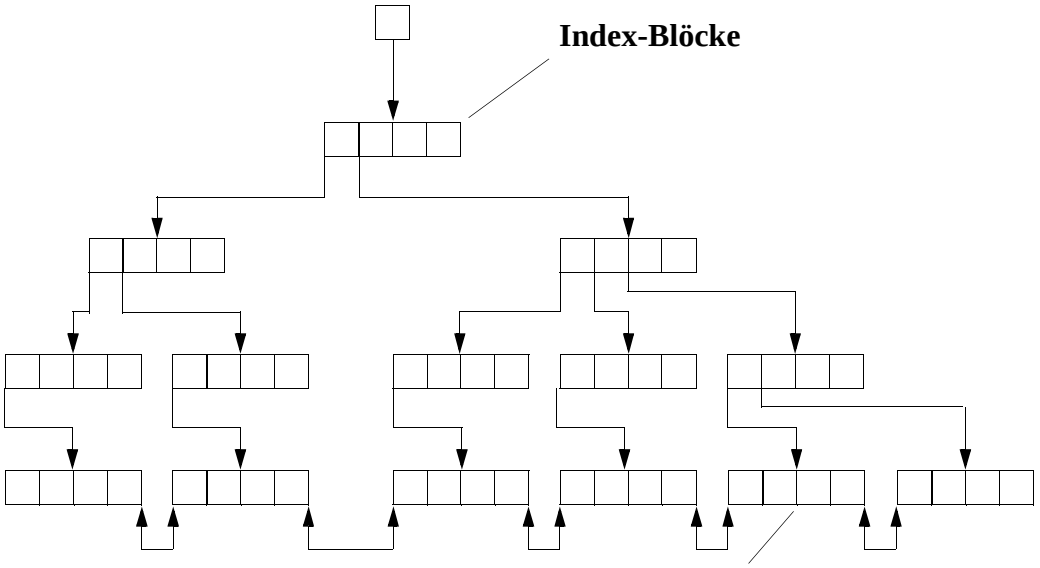
```
SELECT rowid, ename FROM emp;

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
```



Oracle kennt zwei Arten von Indexen, den B*TREE Index und den Bitmapped 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 Sequenzblökken werden die ROWID's der Records gespeichert:



Index-Blöcke: Baumstrukturierter Index-Satz zu den Sequenzblöcken
Sequenz-Blöcke: «Leafs», enthalten die ROWID's

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.

Vorwahl	Telefon	PLZ	Ort	Straße
033	345 02 40	3628	Seftigen	Sonnenrain 5
031	385 30 11	3000	Bern	Zieglerstr. 34

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(edate) = TRUNC(sysdate)      (Index)
... WHERE edate BETWEEN TRUNC(sysdate)       (Index OK)
      AND TRUNC(sysdate) + .99999;
```

- 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 Indexe 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 Data-WareHouse Anwendungen bieten sie sehr grosse Vorteile.

Beispiel: Abfragen auf männlich, weiblich sind sehr unselektiv

ROWID	A	B	C	D	E
M	1	1	0	1	1
W	0	0	1	0	0

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;

   STATUS  COUNT(*)
--------- ---------
        0    176858
        1         4
        2       373
```

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

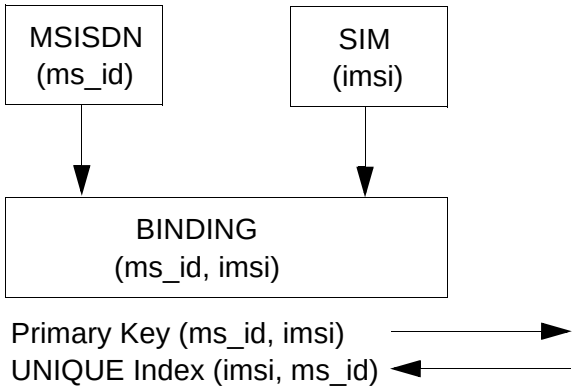
```
SELECT lname, count(*) FROM customer
 GROUP BY lname;

LNAME                                     COUNT(*)
----------------------------------------- ---------
AST Communication                                 3
Abbet                                             1
Abbondioli                                        1
.....
```

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;
```

```
Wert        Anzahl
------------------
23051997       649
22051997       604
24051997       578
02051997       573
21051997       536
17051997       535
09051997       533
03051997       527
16051997       515
01051997       502
30041997       496
```

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;

R-Count    Keys
---------- ----------
         1      63163
       512        593
      1024        115
      1536         26
      2048          9
      2560          5
      3072          1
      3584          1
      4096          1
      4608          0
      5120          0
      5632          0
      6144          0
      6656          0
      7168          0
      7680          0
```

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;

IMEI
---------------
490137201144390
490137203883610
490137203905860
```

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.

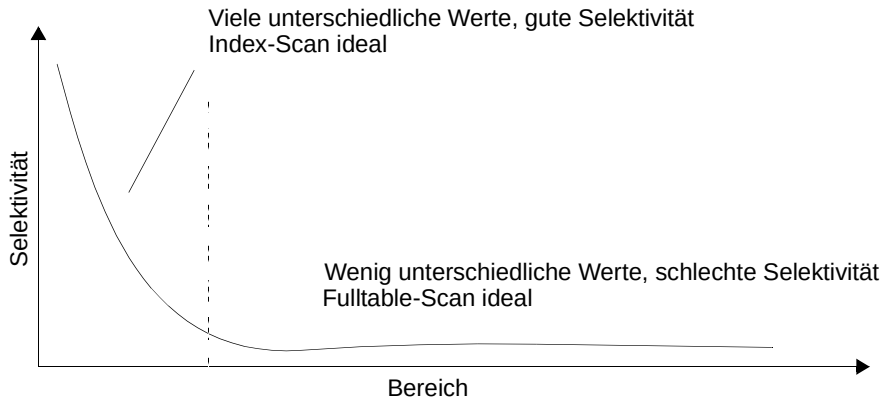
Bei der folgenden Abfrage ist damit zu rechnen, dass nicht die gleiche Selektivität vorhanden ist:

```
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.

Bucket	1	2	3	4	5	6	7	8	9	10
Endwert	10	20	30	40	50	60	70	80	90	100

Gleichmässige Datenverteilung: 10 % der Daten sind in Bucket 10

Bucket	1	2	3	4	5	6	7	8	9	10
Endwert	5	5	5	5	10	20	50	60	70	100

Ungleichmässige Datenverteilung: 30 % der Daten müssen sich Bucket 10 teilen

2.5 Join Algorithmen

Oracle kennt verschiedene Join-Methoden, um zwei Mengen miteinander zu verbinden. Jede Methode hat Vorteile, welche typischerweise von der Datenmenge abhängig ist.

Sort-Merge Join

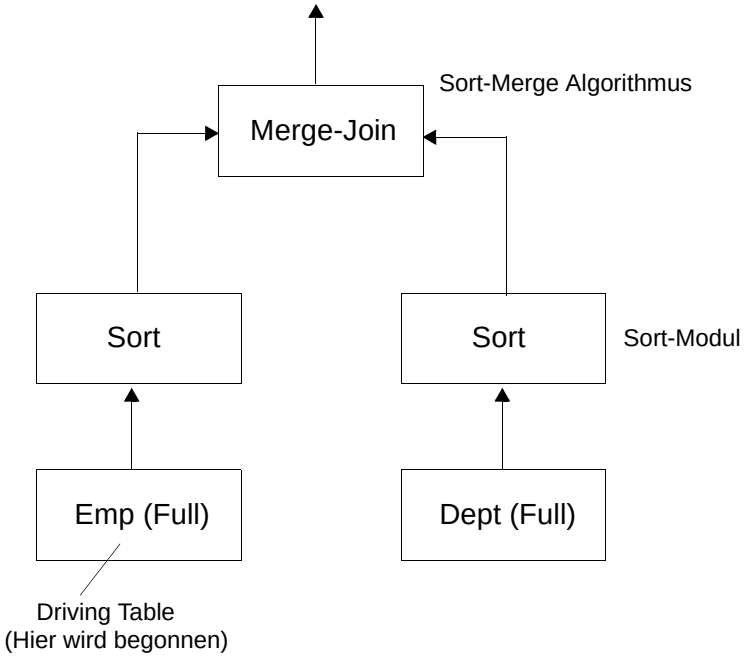
Beim Sort-Merge Join werden die Tabellen über das Sort-Modul aufgrund der Join Bedingung sortiert. Allfällige WHERE Klauseln können direkt während dem Full Table Scan evaluiert werden und schicken dementsprechend weniger Treffer in die nächst höhere Row Source. Grundsätzlich sollte die kleinere Tabelle zuerst gelesen werden. Die kleinere Tabelle ist diejenige, welche nach Berücksichtigung aller WHERE-Klauseln weniger Rows in die nächst höhere Row Source schickt. Sort-Merge Joins eignen sich für grosse Datenmengen, sie können nur bei Equi-Joins verwendet werden.

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

Beispiel

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

Execution Plan
----------------------------------------------------------
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



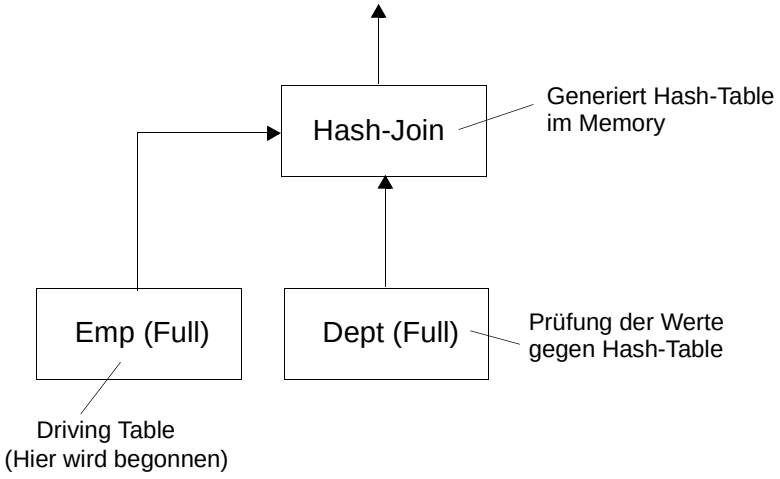
Hash-Join

Hash-Joins sind eine neue Möglichkeit, welche ab Oracle 7.3 angeboten werden. Sie stehen in direkter Konkurrenz zu Sort-Merge Joins und sind diesen meistens auch überlegen. Hash-Joins eignen sich ebenfalls für grosse Datenmengen. Die Zielsetzung des Hash-Joins ist es, den Sort durch einen effizienteren Mechanismus zu ersetzen. Hashing ist die Festlegung eines Wertes durch eine Berechnung. Hash-Joins werden nur vom CBO unterstützt, der RBO kennt diesen Algorithmus nicht. Hash-Joins funktionieren nur auf Equi-Joins. Oracle bildet mit der kleineren Tabelle eine Hash-Table und gleicht danach die grössere Tabelle gegen diese Hash-Table ab.

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

Bei diesem Algorithmus werden zwei verschachtelte Loops verwendet. Für jeden Zugriff im äusseren Loop wird einmal der innere Loop ausgeführt. Die Kosten wachsen somit sehr schnell, womit ein Nested Loop nur für kleine Datenmengen effizient ist. Die zu verarbeitende Datenmenge wird von äusseren Loop bestimmt. Die kleinere Tabelle sollte deswegen im äusseren Loop stehen. Dies ist wiederum die Tabelle, welche nach Berücksichtigung aller WHERE Klauseln weniger Rows in die nächst höhere Row Source schickt.

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

Algorithmus

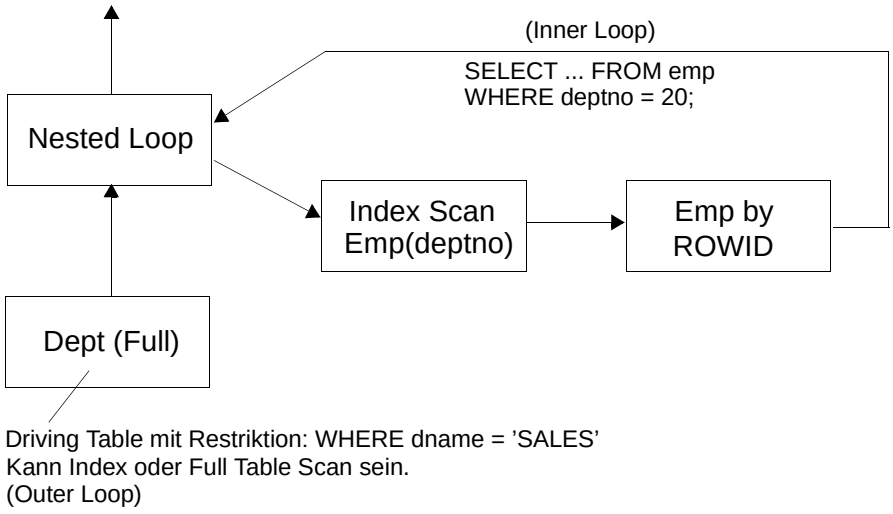
Algorithmus des Nested Loop

```
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



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 Data-WareHouses 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

Damit kann sehr rasch und einfach der Execution Plan kontrolliert werden. Ferner werden Angaben gemacht über die «Kostenberechnung» des Optimizer. Man kann mit diesem Hilfsmittel sehr rasch feststellen, ob sich der Optimizer «verschätzt» hat. Man sieht jedoch die tatsächlich verarbeiteten Datenmengen in den einzelnen Modulen nicht. Für genauere Abklärungen muss der umständlichere Weg über TKPROF gewählt werden.

Installation von AutoTrace

Nur notwendig, wenn dies nicht bereits gemacht wurde.

1. Script **utlxplan.sql** als User SYS laufen lassen. Dieses Script erstellt die notwendige Tabelle PLAN_TABLE. Dieses Script ist im HOME Directory von Oracle zu finden.
2. Die PLAN_TABLE muss an alle User die AutoTrace nutzen dürfen verfügbar gemacht werden.

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

3. Rolle «PLUSTRACE» erstellen mit dem Script **plustrce.sql.** Dieses Script ist im HOME Directory von SQL*PLUS zu finden.
4. Die Rolle «PLUSTRACE» an alle User die AutoTrace nutzen dürfen granten.

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

Nutzen von AutoTrace

Innerhalb von SQL*PLUS können folgende Optionen gesetzt werden:

```
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> 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)
```

```
SQL> set autotrace on explain statistics

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)

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:

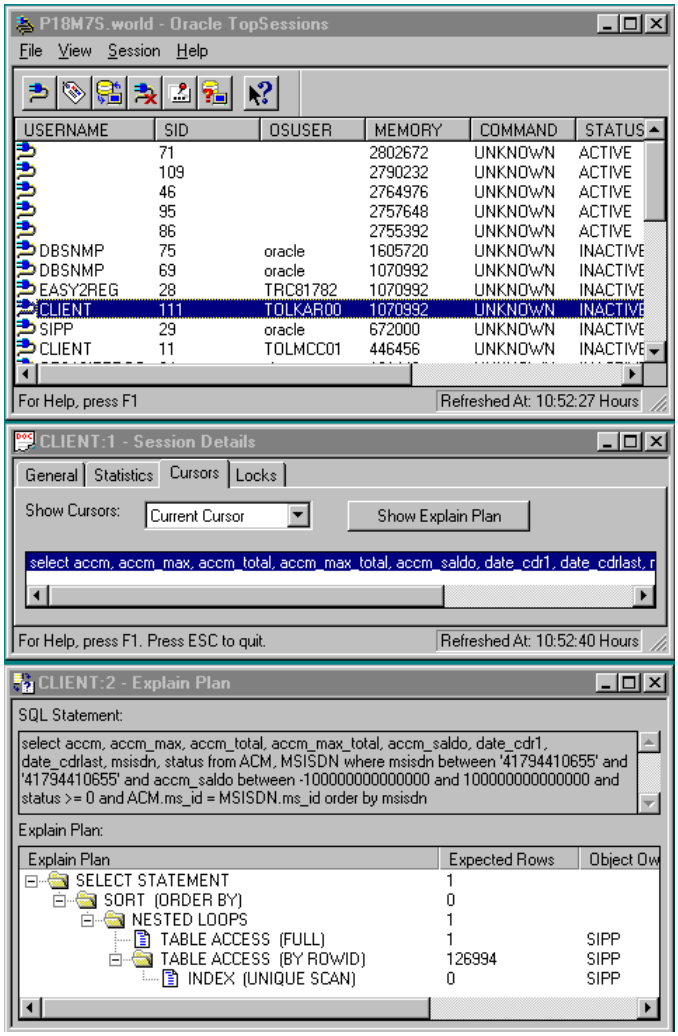
Cost: «Kostenberechnung» des CBO

Card: Cardinality bezeichnet die geschätzte Anzahl Rows, welche auf jeder Stufe (Row-Source) zu verarbeiten sind. Diese Information ist sehr hilfreich, da sich der CBO ja aufgrund von Statistiken entscheidet. Fehlentscheide sind somit sehr schnell ersichtlich. Die Cardinality ist also eine Schätzung des CBO, wieviel Arbeit (Anzahl Rows) auf jeder Stufe zu verrichten sind.

Bytes: Schätzung der zu verarbeitenden Bytes.

TopSession im Enterprise Manager

Das aktuelle SQL-Statement einer beliebigen Session kann sehr rasch und auf einfache Weise analysiert werden.



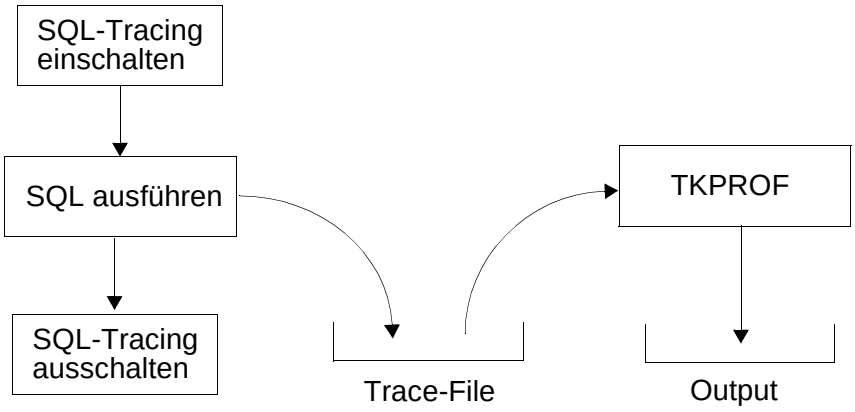
Session auswählen

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.



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.

```
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
    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;
```

```
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.putf(vFile,'At %s user %s@%s opens file %s with %s server \n',
                  vDate, USER,vGlobal_name,vSPid,vServer);
    utl_File.fclose(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

Nutzen 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

Output-File

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: exeela

********************************************************************************
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
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.01       0.02          2          3          4           6
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.02       0.03          2          3          4           6

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

Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      6   NESTED LOOPS
      4    TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'DEPT'
     14    TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'EMP'
```

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.
- Die Zeitaufteilung zwischen «Execute» und «Fetch» kann nicht beeinflusst werden. Dies hängt vom jeweiligen Befehl ab. Deshalb sollte nur das Total der beiden beachtet werden.
- 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.
- Das Verständnis des Execution Plan ist von grundlegender Bedeutung um nun Massnahmen zur Verbesserung einzuleiten. Solche Massnahmen bestehen einerseits aus dem Anbringen von Hints, dem Legen von Indexen oder dem Umschreiben von SQL-Statements.

2.7 Optimierung

Nach der Auswertung des Execution Plans stellt sich nun die Frage, wie ein SQL-Statement optimiert werden kann. In erster Linie sollte immer überlegt werden, ob das SQL-Statement optimal formuliert ist und ob die allenfalls notwendigen Indexe vorhanden sind. Man bedenke also nochmals, dass nur 3 - 5 % der SQL-Statements Performance Probleme aufweisen dürfen, ansonsten muss das Datenmodell und die gesamte Konfiguration näher untersucht werden. Oracle stellt für das SQL-Tuning sogenannte «Hints» zur Verfügung, die den Optimizer anweisen, einen anderen Execution Plan zu wählen. Dabei handelt es sich lediglich um einen Hint (Hinweise), und nicht um einen Befehl. Der Optimizer kann diesen Hint also auch ignorieren. Die Angabe des Hint gehört nicht zur Syntax des SQL-Statements, das heisst, Syntax-Fehler bei der Angabe des Hints werden vom Optimizer nie beanstandet. Hints werden in Form von Kommentaren dem SQL-Statement mitgegeben. Ob der Hint berücksichtigt wurde sieht man erst im Execution Plan. Es macht deshalb keinen Sinn einen Hint anzubringen wenn nicht gleichzeitig auch der Execution Plan dazu kontrolliert wird.

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.

1. Test des SQL-Statements mit dem anderen Optimizer. Vergleich des Execution Plans der beiden Optimizer. Gleiche Execution Plans deuten auf eine gute Optimierung hin.

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

Folgende Hints stehen zur Verfügung

Hint	Erläuterung
RULE	Verwendung des Rule Based Optimizer
CHOOSE	Verwendung des Cost Based Optimizer
FIRST_ROWS	CBO Optimierung der Zeit zur Anzeige der ersten Row. Index-Scans und Nested Loops werden vom Optimizer bevorzugt eingesetzt.
ALL_ROWS	CBO Optimierung des Gesamtdurchsatzes. Fulltable-Scans werden vom Optimizer bevorzugt eingesetzt.
FULL(Table)	Fulltable-Scan auf angegebener Tabelle durchführen
ORDERED	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.
USE_MERGE(Table)	Joine die angegebene Tabelle zum Resultat der vorangegangenen Row-Source mittels einem Sort-Merge Join.
USE_HASH(Table)	Joine die angegebene Tabelle zum Resultat der vorangegangenen Row-Source mittels einem Hash Join.
USE_NL(Table)	Joine die angegebene Tabelle zum Resultat der vorangegangenen Row-Source mittels einem Nested Loop. Die angegebene Tabelle wird im inneren Loop verarbeitet.
INDEX(Table Index)	Verwendung des Index einer bestimmten Tabelle

AND_EQUAL(Table Index Index Index ...)	Verwendung eines Index Merge auf bestimmten einzelnen Indizes der angegebenen Tabelle.
	
	`SELECT ename, sal`
	`  FROM emp`
	`WHERE deptno = 10`
	`  AND ename  = 'KING';`

HASH_AJ	CBO: Anti Join als Hash auflösen.
	
	`SELECT ...`
	`WHERE deptno NOT IN (SELECT ...);`
	
	NOT IN Abfragen möglichst vermeiden !

MERGE_AJ	CBO: Anti Join als Merge -Join auflösen.
	
	`SELECT ...`
	`WHERE deptno NOT IN (SELECT ...);`
	
	NOT IN Abfragen möglichst vermeiden !

CACHE	Full Table Scans lassen die Blocks solange wie möglich in der SGA stehen. Ideal für kleine Lookup Tabellen.
PARALLEL	Im Zusammenhang mit der Parallel Query Option einsetzbar. Angabe der concurrent Query Server zur Verarbeitung angeben.
ROWID(Table)	Table Scan über die ROWID der Tabelle.
CLUSTER(Table)	Verwendung des Index Clusters der Tabelle.
HASH (Tabelle)	Verwendung des Hash Clusters der Tabelle.

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.

```
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) (Cost=5 Card=14 Bytes=756)
    TABLE ACCESS (FULL) OF 'DEPT' (Cost=1 Card=4 Bytes=116)
    TABLE ACCESS (FULL) OF 'EMP'
```

DISTINCT

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

Schlecht ist:

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

besser ist:

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:

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

besser ist:

```
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 erweiteren 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:

```
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;
```

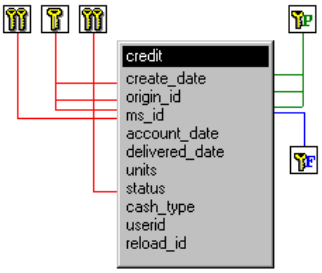
Verwenden der Funktion in normalem SQL-Statement:

```
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.

```
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)));

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     9792     19.88      20.43          6      12117          2      146876
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     9794     19.88      20.43          6      12117          2      146876

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 10  (SIPP)
Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
 147334   TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'CREDIT'
********************************************************************************
```

RULE-based

Wie verhält sich der RULE-based Optimizer ?

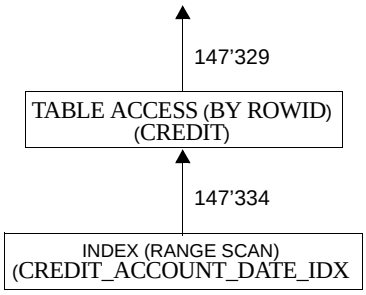
```
SELECT /*+ RULE */ ORIGIN_ID, TO_CHAR(CREATE_DATE,'YYYYMMDDHH24MISS'),
                   TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'), MS_ID, RELOAD_ID
  FROM CREDIT
 WHERE (STATUS = 0 AND ACCOUNT_DATE < (SYSDATE - (1 / 24)))

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     9806     20.55      22.50       2251      12142          2      147075
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     9808     20.56      22.50       2251      12142          2      147075
Misses in library cache during parse: 1
Optimizer goal: RULE
Parsing user id: 10  (SIPP)
Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: HINT: RULE
 147626   TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'CREDIT'
********************************************************************************
```

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.

Index-Zugriff

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_BMIDX 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)))

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.01          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     9792     28.10      30.12        798     294688          0      146878
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     9794     28.11      30.13        798     294688          0      146878

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)
```

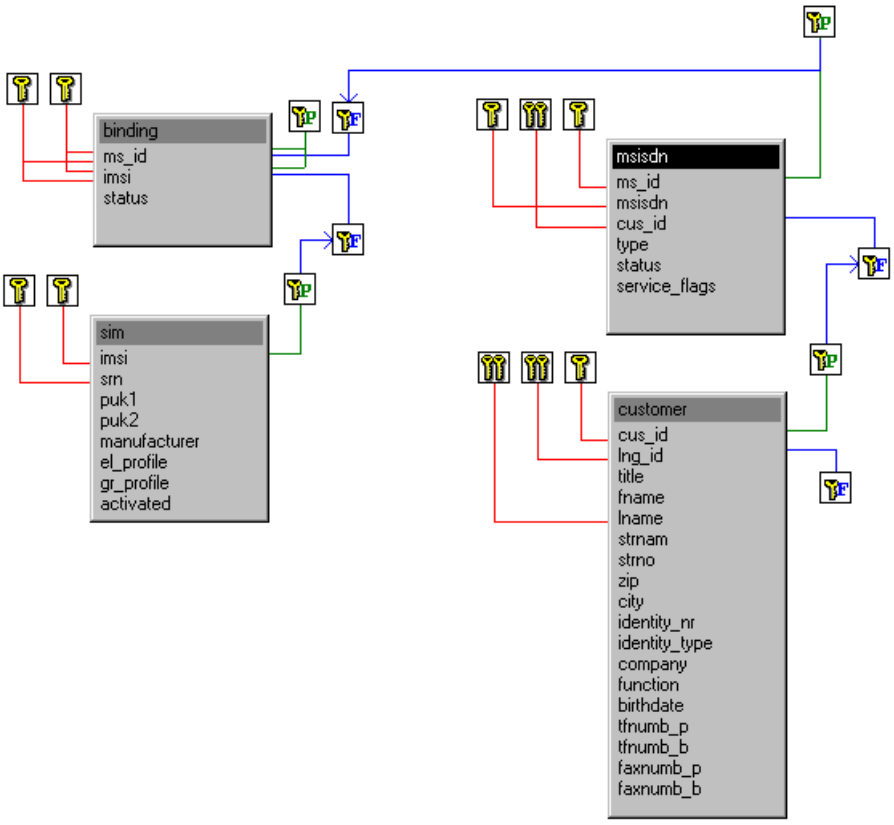


Da sehr viele Rows zurückkommmen ist der Full-Table Scan deutlich schneller. Index auf ACCOUNT_DATE wieder droppen !

3.2 Join-Methoden (Algorithmen)

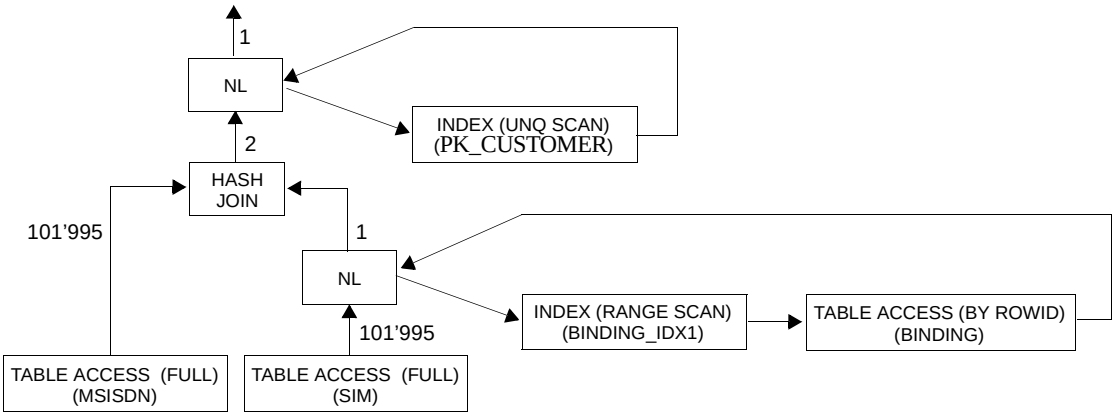
Join-Methoden

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 '89410196224173486299' AND '89410196224173486299'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      2      0.00       0.00          0          0          0           0
Fetch        1      2.71       3.41          7       3238          4           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      2.71       3.41          7       3238          4           1

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)
********************************************************************************
```

Execution Plan grafisch dargestellt:

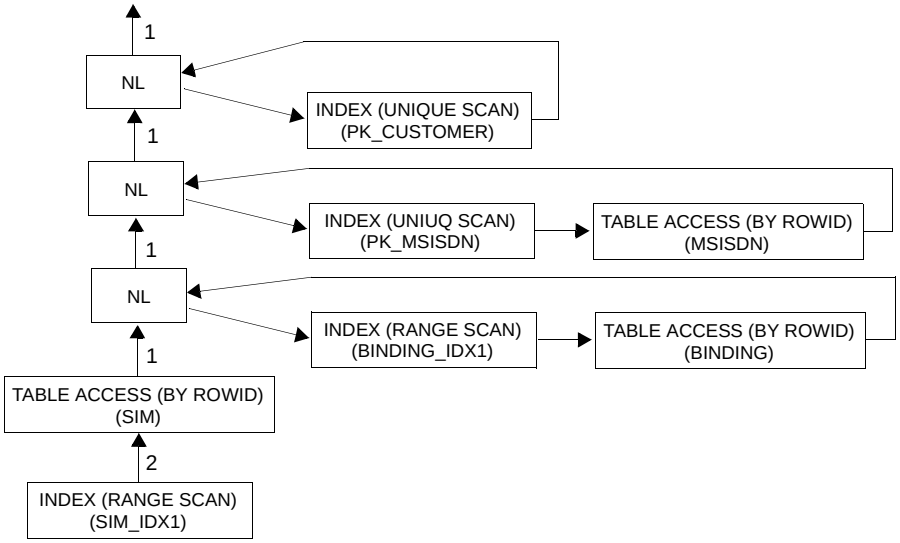


Der Join zwischen der Tabelle SIM und BINDING erfolgt in einem Nested Loop. Dazu wird die gesamte Tabelle SIM gelesen und der innere Loop wird 101’995 mal ausgeführt um genau eine Row zurückzubringen. Der Join zwischen diesem Ergebnis (SIM, BINDING) und der Tabelle MSISDN wird in einem HASH-Join vorgenommen, 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 zurückkommt 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 '89410196224173486299' AND '89410196224173486299';

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute      2      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0         32          0           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6      0.00       0.00          0         32          0           2

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

Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: HINT: FIRST_ROWS
      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'
      2       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    INDEX   GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_CUSTOMER' (UNIQUE)
********************************************************************************
```

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 '89410196224173486299' AND '89410196224173486299'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0         16          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0         16          0           1

Misses in library cache during parse: 0
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'
      2       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    INDEX   GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_CUSTOMER' (UNIQUE)
********************************************************************************
```

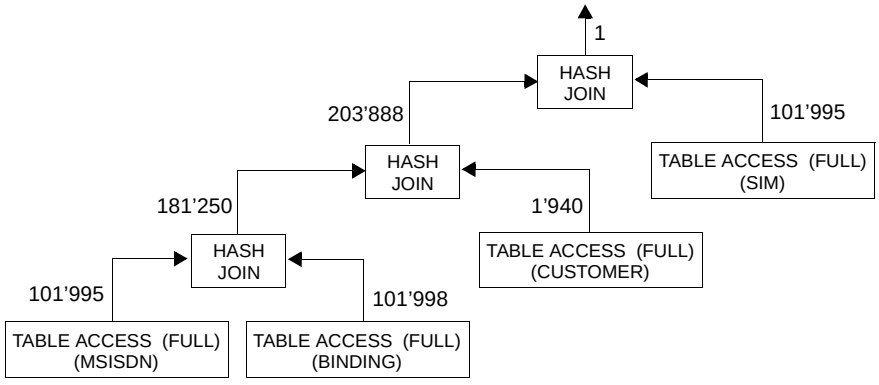
Der RULE based Optimizer wählt in diesem Fall den optimalen Execution Plan, da er wenn möglich Indexe verwendet und Fulltable Scans vermeidet. Die Lösung besteht hier also den RULE based Optimizer zu verwenden, da dadurch der Hint einfacher wird.

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 '89410196224173486299' AND '89410196224173486299'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      2      0.01       0.01          0          0          0           0
Fetch        1      9.80      14.62          5       4075          8           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      9.81      14.63          5       4075          8           1

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 '89410196224173486299' AND '89410196224173486299'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.02          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0         17          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.01       0.02          0         17          0           1

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'
      2       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

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

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.01       0.02          0          0          0           0
Execute      4      0.01       0.15          0          0          0           0
Fetch        2     24.19      33.06       4974       8365       2246           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        8     24.21      33.23       4974       8365       2246           2

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

Rows     Execution Plan
-------  ---------------------------------------------------
      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

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

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.01          0          0          0           0
Execute      2      0.00       0.00          0          0          0           0
Fetch        2     24.72      26.69        283     818552          4           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6     24.72      26.70        283     818552          4           2

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

Rows     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 3 Mittels NOT EXISTS

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

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute      2      0.00       0.00          0          0          0           0
Fetch        2     24.76      27.23        290     818552          4           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        6     24.76      27.23        290     818552          4           2

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

Rows     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

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

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute      4      0.00       0.00          0          0          0           0
Fetch        2     11.34      21.50       4659       6656          8           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        8     11.34      21.50       4659       6656          8           2

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

Rows     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.

```
select count(*)
from
 cdr;

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        3      0.00       0.00          0          0          0           0
Execute      3      0.00       0.00          0          0          0           0
Fetch        3     50.16     144.41     191775     192321          9           3
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        9     50.16     144.41     191775     192321          9           3

Misses in library cache during parse: 0
Optimizer 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 ?

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

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        4      0.02       0.01          0          0          0           0
Execute      4      0.00       0.00          0          0          0           0
Fetch        4     19.57      41.89       8746      42640          0           4
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       12     19.59      41.90       8746      42640          0           4

Misses in library cache during parse: 0
Optimizer 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);

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.01          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      3.50       4.83        309       1379          2           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
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
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      0   SORT (AGGREGATE)
 126994    TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'ACM'
********************************************************************************
```

Mit einem Index auf date_cdr1 kann die Abfrgae 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;

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.01          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          3          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
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
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
      0   SORT (AGGREGATE)
     74    FILTER
     75     INDEX   GOAL: ANALYZED (RANGE SCAN) OF 'ACM_DATE_CDR1_BTIDX' (NON-
UNIQUE)
********************************************************************************
```

4. Datenbank Tuning

Übersicht

Das Datenbank Tuning wird in die folgenden Teilgebiete gegliedert:

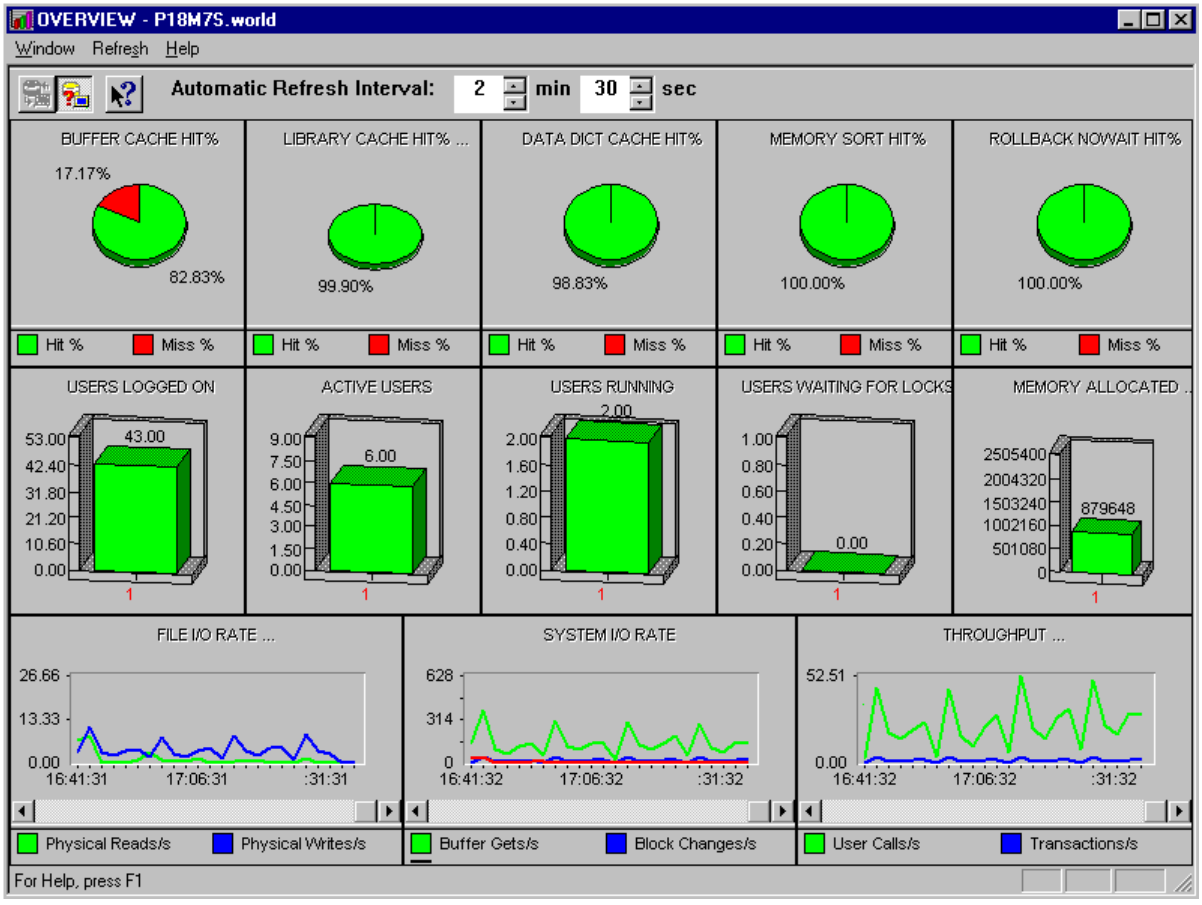
- Cache und Memory Optimierung
- Indexierungs Strategie
- Fragmentierungen
- I/O-Optimierung
- Block Contention
- Transaktionen und Locking

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

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

SQL Statements

Berechnung der Buffer Cache HitRatio in %

```
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 benützen am meisten Ressourcen.

UserBufferCache - P18M7S.world

Window Refresh Help

Automatic Refresh Interval: 0 min 30 sec

	User	sid	ConsGets	BlockGets	PhysReads	HITRATIO
1	SYSADM	112	587237	1170	197530	66
2	CLIENT	110	53758	194	16970	68
3	CLIENT	28	40313	188	11665	71
4	CLIENT	107	822399	3922	228679	72
5	CLIENT	92	558844	2897	130981	76
6	SIPP	43	6638	4390	2159	80
7	OPS$SIPPALT	78	16417604	2595774	2935613	84
8	CLIENT	30	204758	910	26257	87
9	EASY2REG	93	51	18	7	89
10	DBSNMP	29	28	2	3	90
11	CLIENT	20	10084	48	937	90
12	CLIENT	41	250920	1332	23283	90
13	CLIENT	104	11076	40	940	91
14	DBSNMP	27	125	2	7	94
15	EASY2REG	102	11932	1550	741	94
16	EASY2REG	44	494	70	24	95
17	EASY2REG	56	2727	342	131	95
18	EASY2REG	88	128	18	6	95
19	SIPP	75	928	678	61	96
20	EASY2REG	18	27	2	1	96
21	EASY2REG	31	32	6	1	97
22	OPS$SIPPROC	21	371023	531147	11821	98

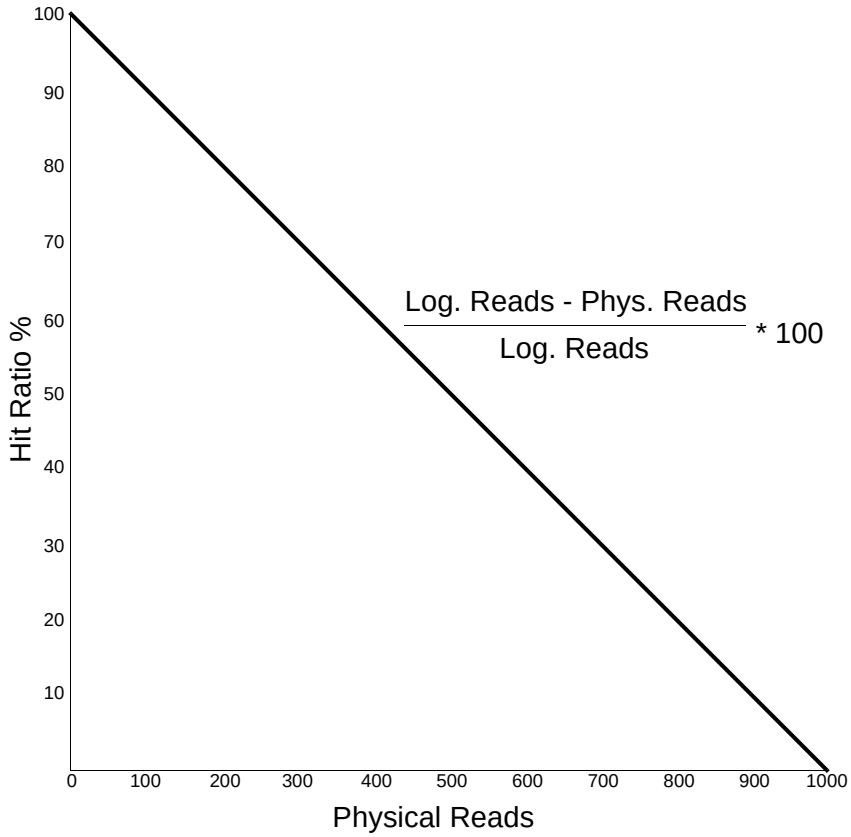
For Help, press F1

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

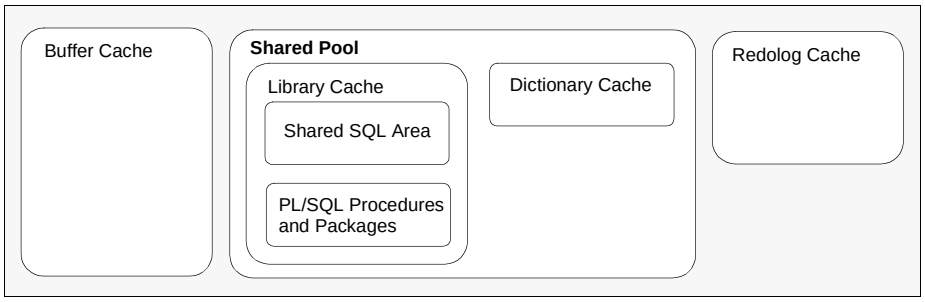
Das Diagramm zeigt die Auswirkungen des physischen I/O bei einem konstanten logischen Read (Consistent Gets + Block Gets).



Je weniger Informationen im Buffer Cache gefunden werden um so 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, geparst und ausgeführt werden muss. Hier gilt die Regel, dass **99 %** aller SQL-Statements in geparster Form im Memory vorliegen müssen. Ist dies nicht der Fall so muss der Wert SHARED_POOL_SIZE erhöht werden.

Berechnung

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

SQL-Statement

```
select (1-(sum(reloads)/sum(pins))) *100 from v$librarycache;
```

Data Dictionary Cache Hit %

Der Data Dictionary Cache ist Teil des Shared Pools. Nach dem Instance Startup werden die Data Dictionary Informationen ins Memory geladen. Nach einer gewissen Betriebszeit sollte sich ein stabiler Zustand einstellen. Der Data Dictionary Cache Miss sollte kleiner als 10 % sein.

Berechnung

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

SQL-Statement

```
select (1- (sum(getmisses)/sum(gets))) * 100
  from v$rowcache;
```

Memory Sort Hit %

Anteil der Sort Operationen im Memory:

Berechnung

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

SQL-Statement

```
select (sum(decode(name, 'sorts (memory)', value, 0)) /
       (sum(decode(name, 'sorts (memory)', value, 0)) +
        sum(decode(name, 'sorts (disk)', value, 0)))) * 100
  from v$sysstat;
```

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

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

SQL-Statement

```
select ((sum(gets)-sum(waits)) / sum(gets)) * 100
  from v$rollstat;
```

Rollback Segment Waits

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

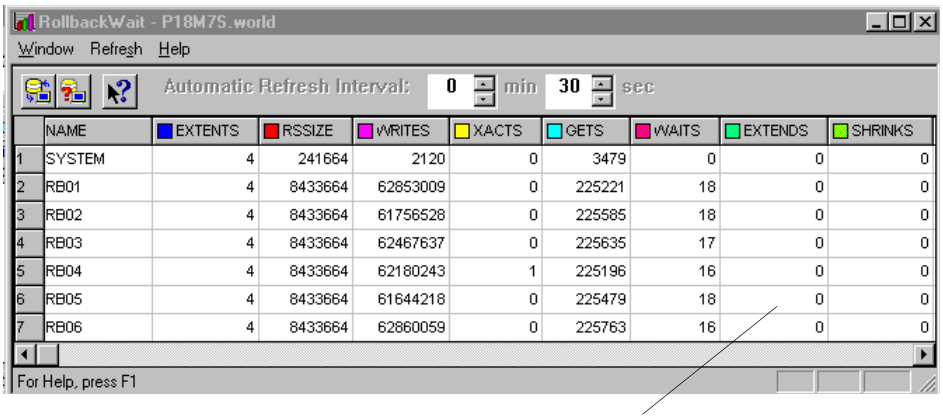
```
SELECT * from v$waitstat;

CLASS                 COUNT       TIME
----------------- ---------- ----------
data block               78        147
sort block                0          0
save undo block           0          0
segment header            0          0
save undo header          0          0
free list                 0          0
system undo header        0          0
system undo block         0          0
undo header             124        139  <---- !
undo block               13          3  <---- !
```

Waits auf «undo header» werden häufig verringert, indem man weitere Rollback Segmente erstellt.
Waits auf «undo block» werden verringert, indem man Rollback Segmente mit mehr Extents erstellt (10 - 20 Extents).

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.

```
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';
```



	NAME	EXTENTS	RSSIZE	WRITES	XACTS	GETS	WAITS	EXTENDS	SHRINKS
1	SYSTEM	4	241664	2120	0	3479	0	0	0
2	RB01	4	8433664	62853009	0	225221	18	0	0
3	RB02	4	8433664	61756528	0	225585	18	0	0
4	RB03	4	8433664	62467637	0	225635	17	0	0
5	RB04	4	8433664	62180243	1	225196	16	0	0
6	RB05	4	8433664	61644218	0	225479	18	0	0
7	RB06	4	8433664	62860059	0	225763	16	0	0

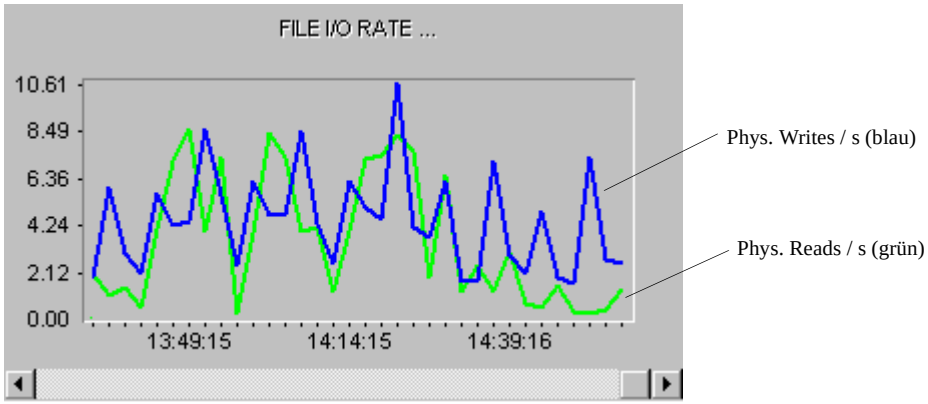
Keine Extents und Shrinks = OK

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

```
select sum(phyrds), sum(phywrts) from v$filestat;
```

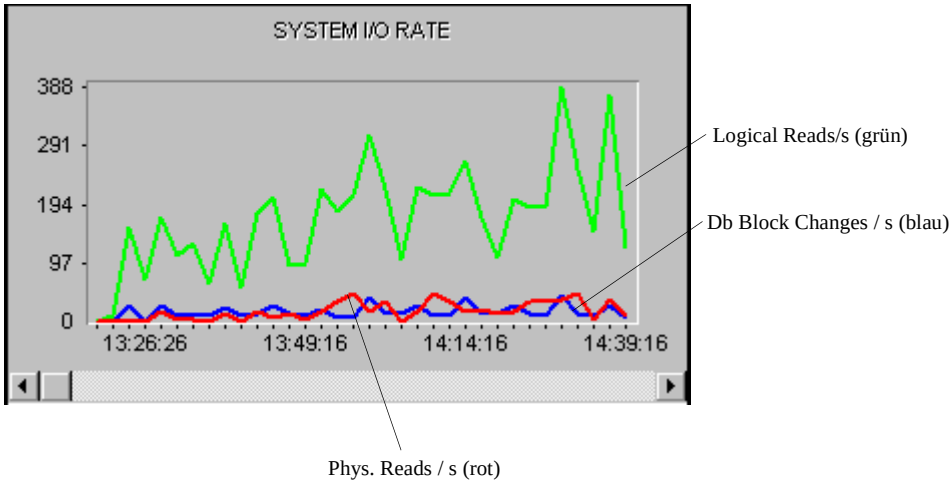


System I/O Rate

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

SQL-Statement

```
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 v$sysstat;
```



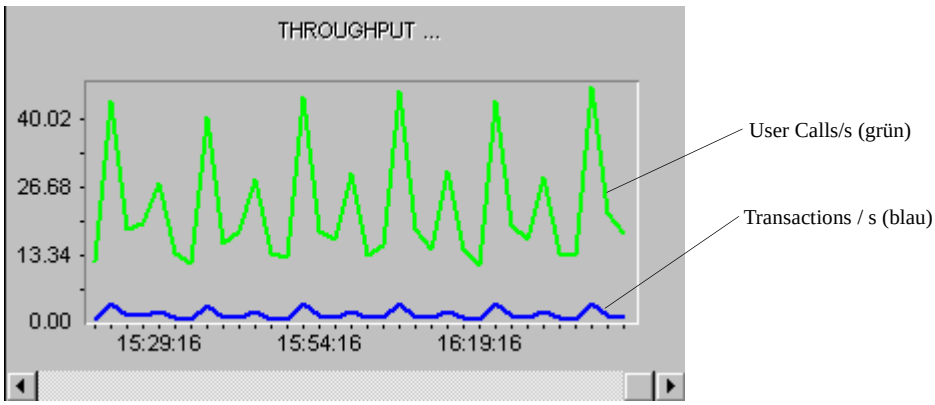
Logical Reads = Db Block Gets + Consistent Gets

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;
```

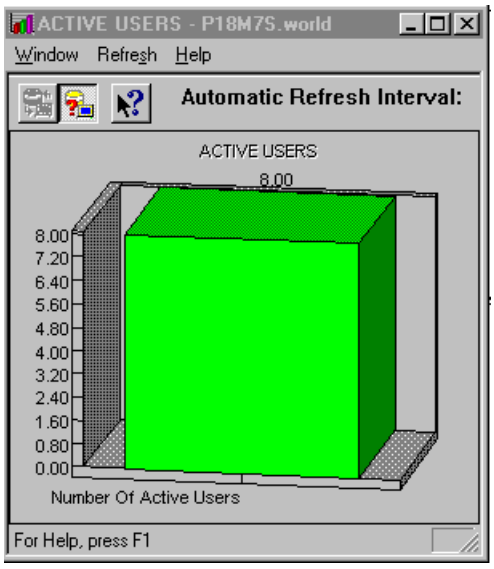


Active Users

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

```
select rawtohex(c.circuit),d.name,s.sid,s.serial#,
       c.status, c.queue,c.bytes
  from v$circuit c,v$dispatcher d, v$shared_server s1,v$session s
 where c.dispatcher = d.paddr(+)
   and c.server = s1.paddr(+)
   and c.saddr = s.saddr(+)
 order by c.circuit;

CIRCUIT  NAME  SID        SERIAL#    STATUS           QUEUE           BYTES
-------- ----- ---------- ---------- ---------------- ---------- ----------
C3874F38 D002         105       3180 NORMAL           NONE              1601
C387536C D000          27          9 NORMAL           NONE          95192596
C38757A0 D000          29          3 NORMAL           NONE           1720467
C3875BD4 D000                      1 NORMAL           NONE         402530056
C3876008 D002          84       9198 NORMAL           NONE              1603
C387643C D002         108       7569 NORMAL           NONE              1665
C3876870 D001          32      15965 NORMAL           NONE            116787
C3876CA4 D002          93       2617 NORMAL           NONE              2643
C38770D8 D002          98      16075 NORMAL           NONE              1603
C387750C D001         112      28224 NORMAL           NONE            149265
C3877940 D001          31      40294 NORMAL           NONE              4666
C3877D74 D001          11      22751 NORMAL           NONE              1603
C38781A8 D002         100      17533 NORMAL           NONE              1603
C38785DC D001          87       2306 NORMAL           NONE              2644
C3878E44 D002          23      23536 NORMAL           NONE              1603
C38796AC D001          65      21222 NORMAL           NONE              1601
C3879F14 D002         102      20336 NORMAL           NONE            611048
C387A348 D001           7      21894 NORMAL           NONE              2314
C387A77C D001          68      40001 NORMAL           NONE              1665
C387ABB0 D002          18      28880 NORMAL           NONE              2149
C387C4E8 D001          56       3310 NORMAL           NONE            138351
C387C91C D002          95      11271 NORMAL           NONE              3475
C387CD50 D002          69      26258 NORMAL           NONE              1603
C387D184 D002          74      46523 NORMAL           NONE          44518043
C387D5B8 D000          75      12017 NORMAL           NONE              8195
C387D9EC D001          44      11978 NORMAL           NONE             21736
C387E688 D002          88      10584 NORMAL           NONE              7911
C3880C5C D001         107       3497 NORMAL           NONE              1635
C3881090 D001          41      62817 NORMAL           NONE            445986
C38814C4 D001          20      37238 NORMAL           NONE             56800
```

Dispatcher

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

SQL-Statement

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

NAME  STATUS           ACC MESSAGES   BYTES      IDLE       BUSY
----- ---------------- --- ---------- ---------- ---------- ----------
D000  WAIT             YES    8980779  515802691  114566779     716941
D001  WAIT             YES    2080331   75276628  115068281     215441
D002  WAIT             YES    4203849  147948411  114892365     391356
```

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

MAXIMUM_CONNECTIONS SERVERS_STARTED SERVERS_TERMINATED SERVERS_HIGHWATER
------------------- --------------- ------------------ -----------------
                 56               0                  0                 5

Waits of Dispatchers

NETWORK    AVG Waits (1/100 s)
---------- ---------------------------------------
ipc        ,02
tcp        ,03

Waits of Shared-Server

AVG Waits (1/100 s)
---------------------------------------
,02

Nbr.Processes actually waiting for a shared server

   QUEUED
---------
       0
```

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;
```

NAME	PHYRDS	PHYWRTS	PHYBLKRD	PHYBLKWRT
/dev/vg11/rsipp01	1098	545164	1098	545164
/dev/vg11/rsipp02	12288	151119	55335	151119
/dev/vg11/rsipp03	734343	1306301	4463897	1317065
/dev/vg11/rsipp04	1135197	76502	17699546	76502
/dev/vg11/rsipp05	266195	21536	3380626	21536
/dev/vg11/rsipp06	144729	61222	154062	100771
/dev/vg11/rsipp07	250033	787023	357896	909131
/dev/vg11/rsipp08	161868	823112	1122768	1613132
/dev/vg11/rsipp09	0	1499	0	1499
/dev/vg11/rsipp10	0	0	0	0
/sipp/d1/ts_system/P18M7S_sys1	16122	31197	28952	31941
/sipp/d1/ts_system/P18M7S_user	0	0	0	0

Free List Hit%

Das Diagramm «Free List Hit %» zeigt Informationen zur Datenblock Contention. Für jedes Segment unterhält Oracle ein oder mehrere Free-lists. Freelists enthalten allozierte Datenblocks für diesen Segment-Extent mit freiem Platz für INSERTS. Bei vielen concurrent INSERTS sind unter Umständen mehrere Freelists zu erstellen.

SQL-Statement

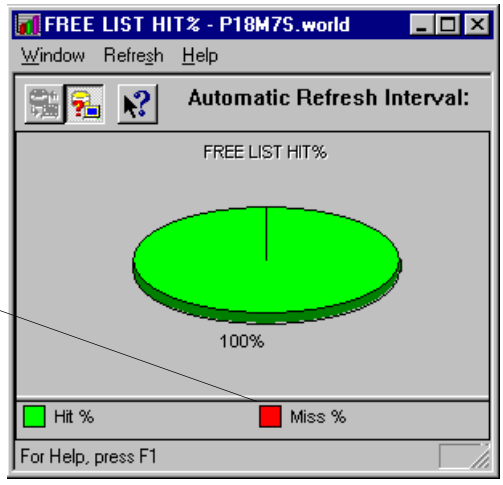
```
select (sum(value)-(sum(count)/2))/sum(value)*100
  from v$waitstat w, v$sysstat s
 where w.class='free list'
   and s.name in ('db block gets', 'consistent gets');
```

Berechnung

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

```
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');
```

Freelist Contention darf nicht mehr als 1 % betragen



Latch

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

SQL-Statement

```
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#;
```

Name	GETS	MISSES	SLEEPS	ImmGets	ImmMiss
process allocation	1151	0	0	1151	0
session switching	26746	0	0	0	0
cached attr list	0	0	0	0	0
modify parameter values	100971	17	10	0	0
virtual circuit buffers	61160911	196	181	0	0
session allocation	3502845	19	21	0	0
session idle bit	34993386	451	519	0	0
cache buffers chains	351978424	1138	1338	805523115	2025
global transaction	1926073	0	0	0	0
global tx hash mapping	724195	0	0	1	0
NLS data objects	1	0	0	0	0
global tx free list	260731	0	0	0	0
query server process	0	0	0	0	0
cache buffer handles	3545	0	0	0	0
multiblock read objects	4064844	17	18	9	0
cache buffers lru chain	32308019	1421	2038	29031168	5631
KCL name table latch	0	0	0	0	0
loader state object freelist	2774	0	0	0	0
dml lock allocation	4328384	11	11	0	0
list of block allocation	2346310	3	3	0	0
sort extent pool	13760	0	0	0	0
user lock	19189	0	0	0	0
enqueue hash chains	26905627	598	686	0	0
trace latch	0	0	0	0	0
cache protection latch	0	0	0	0	0
archive control	653	0	0	0	0
lock element parent latch	0	0	0	0	0
row cache objects	7015533	17	29	6391	0
process queue reference	0	0	0	0	0
enqueues	10738911	18	18	0	0
instance latch	0	0	0	0	0
undo global data	5275435	27	28	0	0
library cache	49455756	6235	4896	6375	9
library cache load lock	23658	0	0	0	0
error message lists	0	0	0	0	0
process queue	0	0	0	0	0
device information	0	0	0	0	0
redo copy	0	0	0	0	0
query server freelists	0	0	0	0	0
parallel query alloc buffer	0	0	0	0	0
redo allocation	14726380	865	1039	0	0
shared pool	4605571	46	55	0	0
messages	7238853	615	595	0	0
system commit number	30233743	281	426	0	0
KCL freelist latch	0	0	0	0	0
transaction allocation	4792886	13	18	0	0
sequence cache	62029	0	0	0	0
cost function	1	0	0	0	0
virtual circuit queues	58346794	2263	2111	0	0
virtual circuits	15239348	69	76	0	0
parallel query stats	0	0	0	0	0
latch wait list	9097	0	0	0	0
ktm global data	4584	0	0	0	0

Library Cache Details

Das Diagramm «Library Cache Details» zeigt Detailinformationen des Library Cache im Shared Pool der Instance. Der Library Cache enthält SQL und PL/SQL Code in geparster Form. Es ist wichtig, dass die Ratio für diese Bereiche nahezu 100% beträgt.

SQL-Statement

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

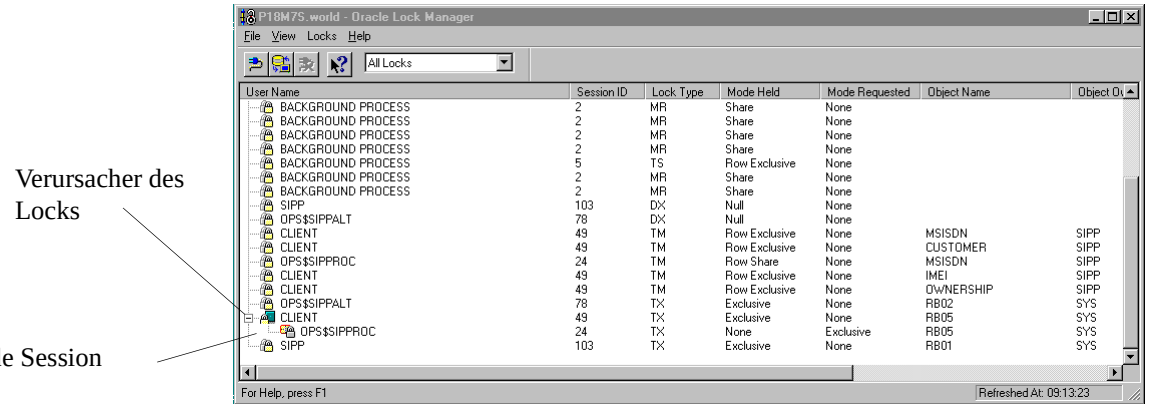
NAMESPACE	GETS	GETHITS	RATIO%	PINS	PINHITS	RATIO%
BODY	7216	7172	99	7216	7166	99
CLUSTER	171	84	49	159	77	48
INDEX	852	540	63	534	380	71
OBJECT	0	0	100	0	0	100
PIPE	21167	21166	100	22158	22157	100
SQL AREA	1910800	1885069	99	11932152	11870605	99
TABLE/PROCEDURE	228485	220990	97	458154	446343	97
TRIGGER	2736	2733	100	2736	2729	100

Lock

Das Diagramm «Lock» zeigt aktive Locks der Datenbank Instance. Eine bessere Übersicht der Locksituation bietet der Lockmanager an, der auch wartende Sessions in der Treelist anzeigt.

SQL-Statement

```
select s.username, s.sid, s.serial#, l.type, l.id1, l.id2,
       decode(l.lmode, 1, 'NULL', 2, 'ROW SHARE', 3,
       'ROW EXCLUSIVE', 4, 'SHARE', 5, 'SHARE ROW EXCLUSIVE', 6,
       'EXCLUSIVE', '?'), decode(l.request, 1, 'NULL', 2, 'ROW SHARE', 3,
       'ROW EXCLUSIVE', 4,'SHARE', 5, 'SHARE ROW EXCLUSIVE', 6,
       'EXCLUSIVE', '?')
 from v$lock l, v$session s
where l.sid=s.sid
order by s.sid, l.type;
```



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.

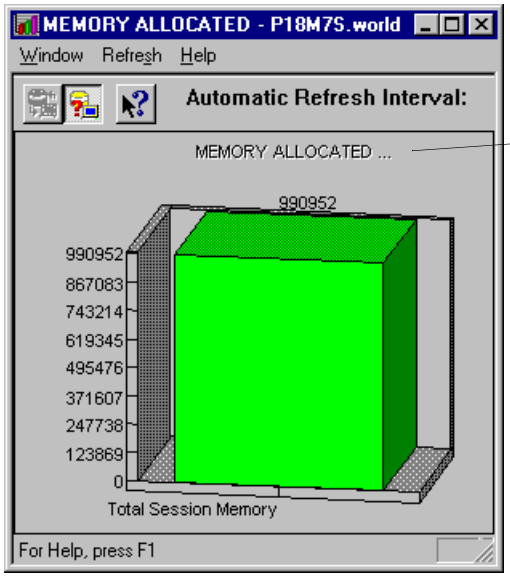
```
create or replace view row_lock_waits
   (username, sid, object_owner,
   object_name, sql_text, file_nr, block_nr, record_nr)
as
select s.username, s.sid,o.owner,o.object_name,a.sql_text,
       s.row_wait_file#,s.row_wait_block#,s.row_wait_row#
 from v$session s, v$sqlarea a, dba_objects o
where 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

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



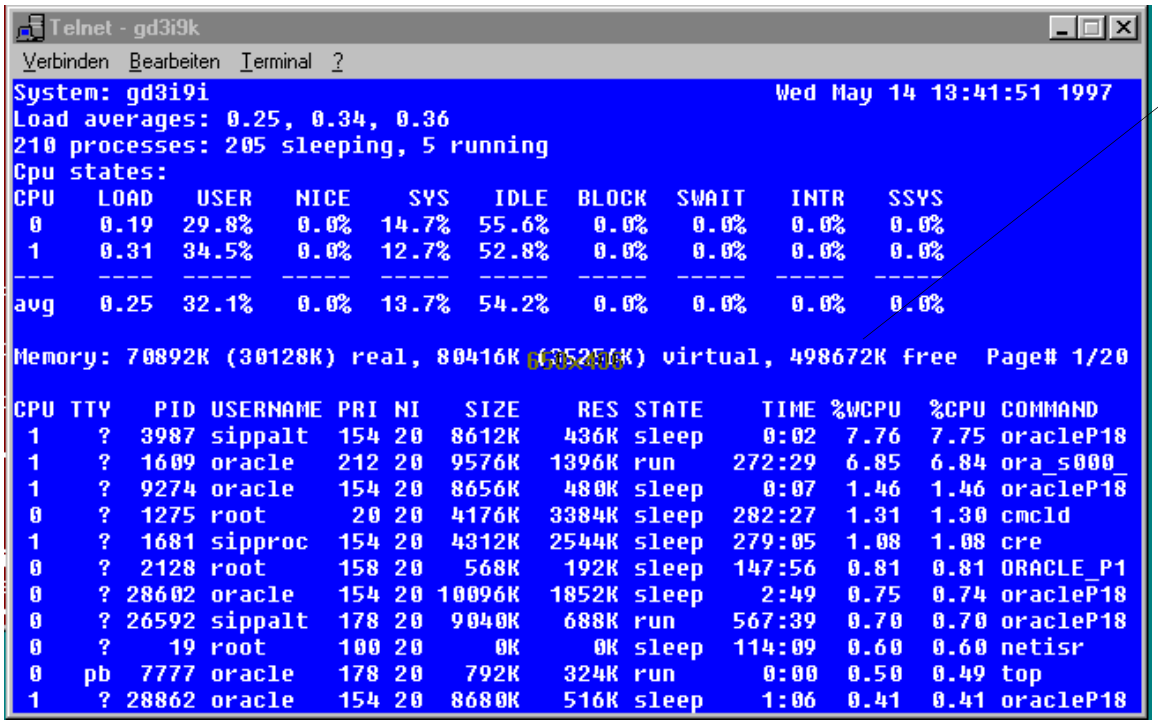
Zur Zeit sind 990952 Bytes alloziert

Die Gösse 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.**

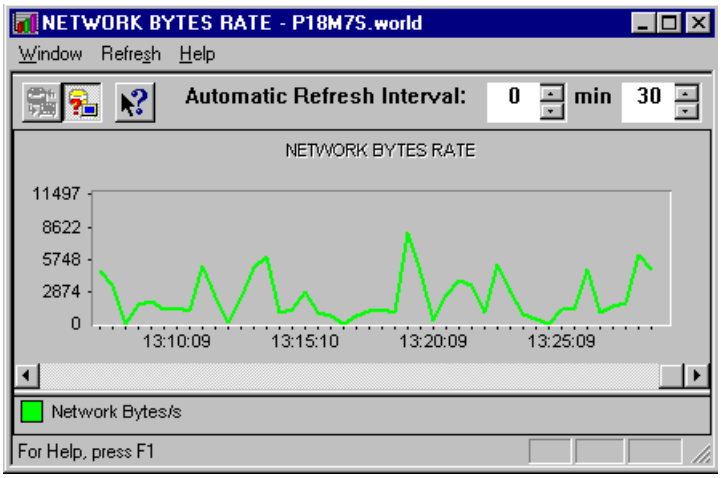


Network Bytes Rate

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

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

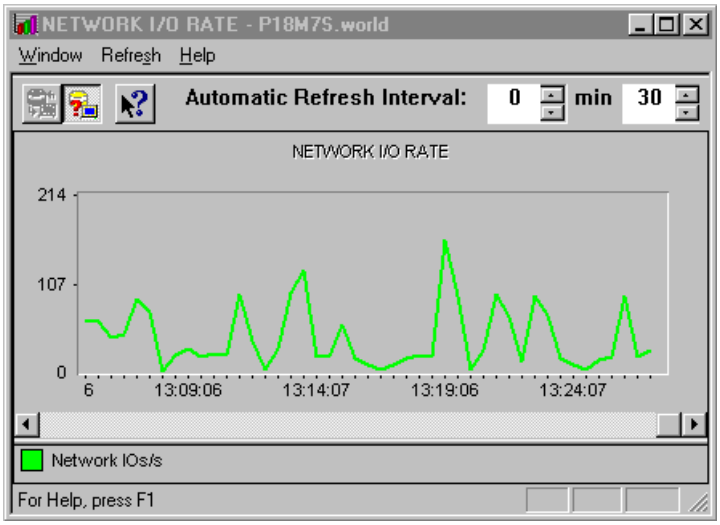


Network I/O Rate

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

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



Session Events

Die V$SESSION_WAIT View enthält sämtliche Events, welche die User- und System Sessions in den Wartezustand versetzen. Diese View kann verwendet werden, um rasch einen Performance Engpass herauszufinden.

SQL-Statement

```
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;
```

```
Sid        User            Event                     Wait Time
---------- --------------- ------------------------- ----------
         1 Background      pmon timer                         0
         2 Background      rdbms ipc message                  0
         3 Background      rdbms ipc message                  0
        19 Background      rdbms ipc message                  0
        13 Background      rdbms ipc message                  0
        54 Background      rdbms ipc message                  0
        70 Background      rdbms ipc message                  0
         5 Background      smon timer                         0
       111 Background      rdbms ipc message                  0
         6 Background      rdbms ipc message                  0
         4 Background      log file parallel write            0
        92 CLIENT          db file scattered read             0
        41 CLIENT          SQL*Net message from clie          0
        84 CLIENT          SQL*Net message from clie          0
       110 CLIENT          SQL*Net message from clie          0
        32 CLIENT          SQL*Net message from clie          0
        30 CLIENT          SQL*Net message from clie          0
        28 CLIENT          SQL*Net message from clie          0
        29 DBSNMP          SQL*Net message from clie          0
        27 DBSNMP          SQL*Net message from clie          0
         7 EASY2REG        SQL*Net message from clie          0
        31 EASY2REG        SQL*Net message from clie          0
        87 EASY2REG        SQL*Net message from clie          0
        98 EASY2REG        SQL*Net message from clie          0
        88 EASY2REG        SQL*Net message from clie          0
        57 EASY2REG        SQL*Net message from clie          0
        44 EASY2REG        SQL*Net message from clie          0
        56 EASY2REG        SQL*Net message from clie          0
        23 EASY2REG        SQL*Net message from clie          0
        11 EASY2REG        SQL*Net message from clie          0
        59 EASY2REG        SQL*Net message from clie          0
        95 EASY2REG        SQL*Net message from clie          0
        76 EASY2REG        SQL*Net message from clie          0
        69 EASY2REG        SQL*Net message from clie          0
       114 EASY2REG        SQL*Net message from clie          0
       100 EASY2REG        SQL*Net message from clie          0
        78 OPS$SIPPALT     db file scattered read            -1
        64 OPS$SIPPROC     log file sync                      0
        24 OPS$SIPPROC     SQL*Net message from clie          0
        12 OPS$SIPPROC     pipe get                           0
        21 OPS$SIPPROC     SQL*Net message from clie          0
        80 SIPP            SQL*Net message from clie          0
        43 SIPP            SQL*Net message from clie          0
       103 SIPP            SQL*Net more data from db          0
       106 SIPP            SQL*Net message from clie          0
        75 SIPP            SQL*Net message from clie          0
        18 SIPP            SQL*Net message from clie         18
       112 SYSADM          SQL*Net message from clie          0
```

Eine Waiting Time von 0 zeigt an, dass die Session gerade auf einen Event wartet. Grosse Wait Times weisen auf ein Performance Problem hin (siehe Oracle Tuning Guide Seite A-4).

Parse Ratio

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

SQL-Statement

```
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

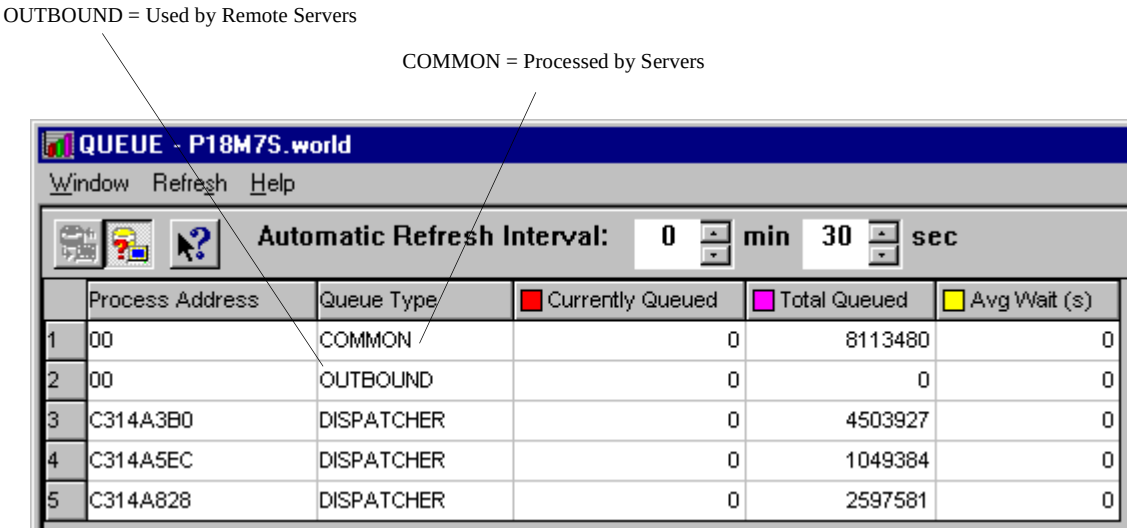
```
select substr(p.pid,1,4) "Pid", p.spid "Spid",
       substr(p.username,1,15) "User",
        decode(s.terminal, NULL, p.terminal, s.terminal) "Terminal",
       decode(s.program, NULL, p.program, s.program) "Program"
  from v$process p, v$session s
 where p.addr=s.paddr order by pid;
```

Queue

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

SQL-Statement

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



	Process Address	Queue Type	Currently Queued	Total Queued	Avg Wait (s)
1	00	COMMON	0	8113480	0
2	00	OUTBOUND	0	0	0
3	C314A3B0	DISPATCHER	0	4503927	0
4	C314A5EC	DISPATCHER	0	1049384	0
5	C314A828	DISPATCHER	0	2597581	0

Redo Alloc Hit%

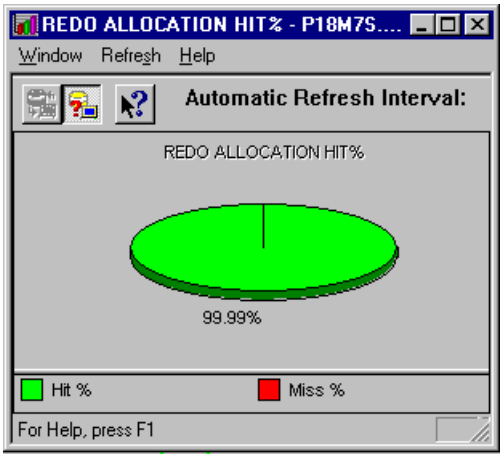
Das Diagramm «Redo Allocation Hit %» zeigt das Buffer Tuning der Redolog File Aktivitäten. Die Misses dürfen nicht grösser als 1 % sein.

Berechnung

$$\text{HitRatio \%} = \left[\frac{\text{Gets + ImmediateGets}}{\text{Gets + ImmediateGets + Misses + ImmediateMisses}} \right] * 100$$

SQL-Statement

```
select ((gets+immediate_gets) /
        (gets+immediate_gets+misses+immediate_misses)) *100
  from v$latch
 where name = 'redo allocation';
```

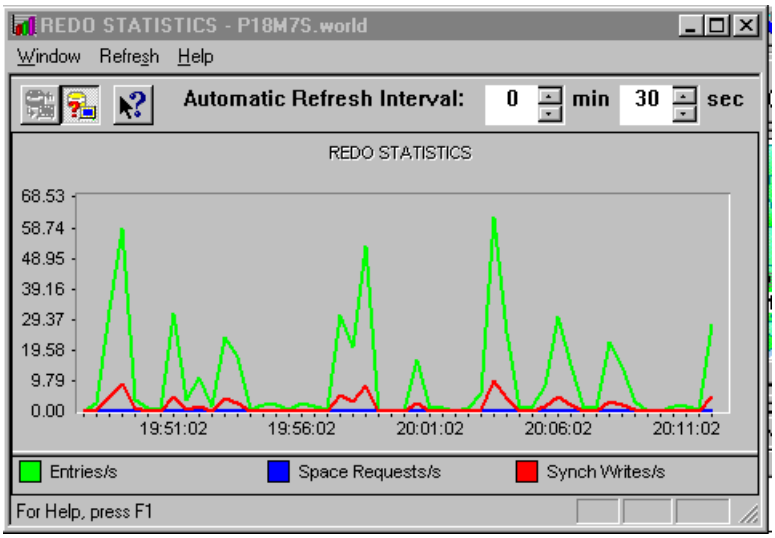


Redo Statistics

Das Diagramm «Redo Statistics» zeigt die Anzahl Redo Entries, Space Requests und Synch. Writes pro Sekunde für die Datenbank Instance.

SQL-Statement

```
select sum(decode(name,'redo entries', value,0)),
       sum(decode(name,'redo log space requests', value,0)),
       sum(decode(name,'redo synch writes', value,0))
  from v$sysstat;
```



Session

Das Diagramm «Session» zeigt die vorhandenen User Sessions mit ihren SID's, Serial-Nr und PID's.

SQL-Statement

```
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

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

Sqlarea

Das Diagramm «SQL Area» zeigt die shared Cusor Informationen im Library Cache.

SQL-Statement

```
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

```
USERNAME      SID SQL_TEXT
----------- ---- ------------------------------------------------------------
CLIENT         30 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

CLIENT         59 select * from USERS where username = 'kon0'
```

System Statistics

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

SQL-Statement

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

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

Auswertung der System Statistik

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

Full Table Scans

```
table scan blocks gotten       61'900'307
table scan rows gotten        194'6840'695
table scans (long tables)          13'267
table scans (short tables)        307'195
```

Index Scans

```
table fetch by rowid           15'653'655
```

Redo Waits

```
redo log space requests             1018
redo log space wait time           21263
```

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

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

Sid        Owner           Object
---------- --------------- --------------------
        31 CLIENT          USERS
        49 CLIENT          USERS
        57 CLIENT          USERS
        92 CLIENT          USERS
        92 CLIENT          MSISDN
        84 CLIENT          USERS
        78 OPS$SIPPALT     ACM
        78 OPS$SIPPALT     SEQ_ORDER_REQ_ORD_ID
        78 OPS$SIPPALT     MSISDN
        78 OPS$SIPPALT     MESSAGE
```

Users logged on

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

SQL-Statement

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

Users running

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

SQL-Statement

```
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

```
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';

Sid   User            Event                          Wait [s]
----- --------------- ------------------------------ ----------
   12 OPS$SIPPROC     pipe get                               48
```

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

```
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.

PPB-ObjStatus - P18M7S.world

Window Refresh Help

Automatic Refresh Interval: 0 min 30 sec

	OWNER	OBJECT_NAME	OBJECT_TYPE	STATUS	LAST_DDL_TIME
1	SIPP	ACM	TABLE	VALID	12.06.97 11:19:48
2	SIPP	ACM_DATE_CDR1_BT	INDEX	VALID	12.06.97 11:19:48
3	SIPP	BINDING	TABLE	VALID	05.06.97 11:38:22
4	SIPP	BINDING_IDX1	INDEX	VALID	06.05.97 10:42:08
5	SIPP	CDR	TABLE	VALID	11.06.97 07:10:43
6	SIPP	CDR_IMEI_BTIDX	INDEX	VALID	30.05.97 16:30:00
7	SIPP	CHECK_ACCM_MAX_	PROCEDURE	VALID	21.05.97 07:09:06
8	SIPP	CLIENT_REQ	TABLE	VALID	28.05.97 10:42:06
9	SIPP	CREDIT	TABLE	VALID	28.05.97 10:42:05
10	SIPP	CREDIT_IDX1	INDEX	VALID	21.04.97 16:55:24
11	SIPP	CREDIT_STATUS_BMII	INDEX	VALID	09.05.97 15:00:14
12	SIPP	CUSTOMER	TABLE	VALID	28.05.97 10:42:05
13	SIPP	CUSTOMER_IDX1	INDEX	VALID	21.04.97 16:55:34
14	SIPP	CUSTOMER_IDX2	INDEX	VALID	21.04.97 16:55:34
15	SIPP	EQUIPTYPE	TABLE	VALID	28.05.97 10:42:05
16	SIPP	ESRFILE	TABLE	VALID	28.05.97 10:42:05
17	SIPP	ESRLOG	TABLE	VALID	28.05.97 10:42:05
18	SIPP	ESRTOTAL	TABLE	VALID	28.05.97 10:42:05
19	SIPP	EVENTLOG	TABLE	VALID	28.05.97 10:42:05
20	SIPP	IMEI	TABLE	VALID	28.05.97 10:42:05
21	SIPP	IMEI_IDX1	INDEX	VALID	21.04.97 16:56:37
22	SIPP	LANGUAGE	TABLE	VALID	28.05.97 10:42:05
23	SIPP	MESSAGE	TABLE	VALID	28.05.97 10:42:05
24	SIPP	MESSAGE_IDX1	INDEX	VALID	21.04.97 16:56:37
25	SIPP	MODULES	TABLE	VALID	28.05.97 10:42:05
26	SIPP	MSISDN	TABLE	VALID	05.06.97 11:40:27
27	SIPP	MSISDN_IDX1	INDEX	VALID	21.04.97 16:56:57
28	SIPP	MSISDN_IDX2	INDEX	VALID	21.04.97 16:57:03

For Help, press F1

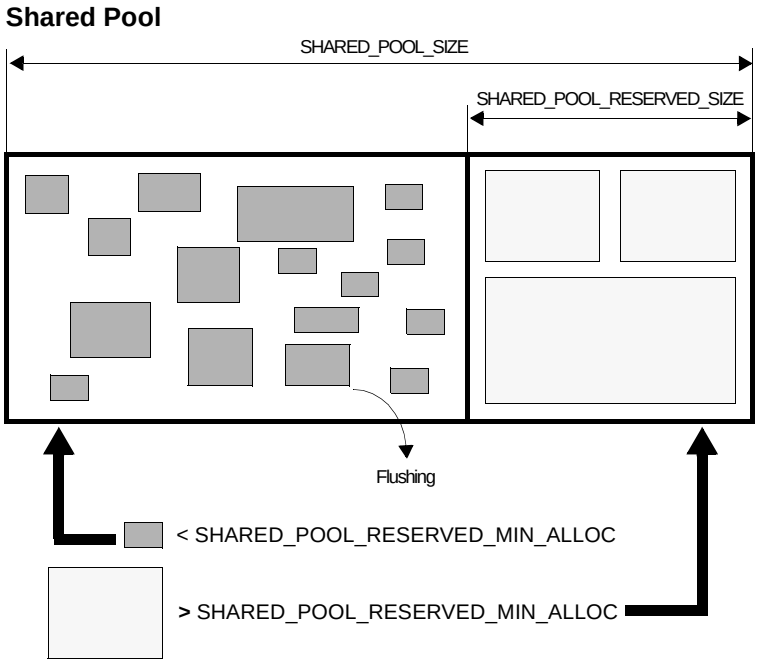
4.2 Detaillierte Analyse des Shared Pools

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.

Memory Fragmentierung

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

Erkennen des Flushings

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.

```
COLUMN ksmlrnum HEADING "Flushed|SP-Pieces" FORMAT 999,990
COLUMN ksmlrcom HEADING "Comment"
COLUMN ksmlrhon HEADING "Command"
COLUMN ksmlrsiz HEADING "Size"

SELECT ksmlrhon,ksmlrcom,ksmlrsiz,ksmlrnum
  FROM sys.x$ksmlru;
```

Auswertung

ksmlrnum Anzahl Objekte, die aus dem Shared Pool geflushed wurden infolge Memory Allocation des geladenen Objekts. (Flushed SP-Pieces).

ksmlrcom Allocation Kommentar, welches die Art der Memory Allocation umschreibt. Angaben wie 'MPCODE' oder 'PLSQL%' heissen, dass ein grosses PL/SQL Objekt geladen wurde (Comment).

ksmlrhon Name des Objects, das geladen wurde, nur bei Pl/SQL und Cursor Objekten (Command).

ksmlrsiz Grösse des Contiguous Memorybereichs der alloziert wurde (Size).

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

Command	Comment	Size	Flushed SP-Pieces
	object info	8664	506
	KXHF Slot Table	19528	513
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0

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

SQL-Statement zur Auslistung der Memory Segmente

```
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 ******************
Description      State                Sum (Size)      Max (Size)      Number of
                                        [Bytes]         [Bytes]       Segments
---------------- ------------ ---------------- --------------- -------------
character set m  InUse                 32896           15152             9
dictionary cach  InUse               1376624            4256           814
fixed allocatio  Flushable               240              40             6
free memory      Free                6258376         6216568           743
free memory      R-free              8000000         8000000             1
KGL handles      Flushable           1596184             960          5272
kxfp subheap     Flushable               584             584             1
kzull            InUse                   304              64             6
library cache    InUse                2545592             584          8643
library cache    Flushable            2210656             584          4708
multiblock rea   InUse                    560             280             2
permanent memor  Permanent           16420872        16420872             1
PLS cca hp desc  InUse                   168             168             1
PLS non-lib hp   Flushable              2104            2104             1
PL/SQL DIANA     InUse                1044264            2112           626
PL/SQL DIANA     Flushable             114392            2104            72
PL/SQL MPCODE    InUse                 449968            8400           179
PL/SQL MPCODE    Flushable              97344            2104            90
reserved stoppe  R-freea                   48              24             2
row cache lru    Flushable               1944              64            40
session param v  InUse                  31440            2096            15
sql area         InUse                7230464            4256          2333
sql area         Flushable            2559104            4264           808
table definiti   Flushable                 64              64             1
trigger defini   Flushable               1080            1080             1
library cache    InUse                2545592             584          8643
library cache    Flushable            2210656             584          4708
multiblock rea   InUse                    560             280             2
permanent memor  Permanent           16420872        16420872             1
PLS cca hp desc  InUse                   168             168             1
PLS non-lib hp   Flushable              2104            2104             1
PL/SQL DIANA     InUse                1044264            2112           626
PL/SQL DIANA     Flushable             114392            2104            72
PL/SQL MPCODE    InUse                 449968            8400           179
PL/SQL MPCODE    Flushable              97344            2104            90
reserved stoppe  R-freea                   48              24             2
```

Grosse Objecte lokalisieren

Grosse Objecte in Shared Pool führen zu Problemen. Sie haben zur Folge, dass andere kleinere Objekte geflushed werden. Damit grosse Objekte nicht mehr jedesmal geladen werden müssen und so kleinere Objekte verdrängen was zu einer Fragmentierung führt, können solche Objekte als 'keep' markiert werden. Sie werden dann nach dem erstmaligen Laden nicht mehr geflushed. Ab Version 7.3 müssen grosse Objekte nicht mehr als 'keep' markiert werden, da allgemein grosse Objekte in kleine Einheiten zerlegt werden.

Kontrolle, welche grossen Objekte im Shared Pool sind:

```
SELECT owner, name, type, kept, loads, sharable_mem
  FROM v$db_object_cache
 WHERE sharable_mem > 10000
ORDER BY sharable_mem desc;
```

Auswertung

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

Owner	Name	Type	Kept	Loads	Size Bytes
SYS	STANDARD	PACKAGE	NO	1	120,628
SYS	STANDARD	PACKAGE BODY	NO	1	28,700
SYS	INDREORG	PROCEDURE	NO	1	25,188
SIPP	CHECK_ACCM_MAX_TOTAL	PROCEDURE	NO	1	23,505
SYS	DBMS_SYS_SQL	PACKAGE BODY	NO	1	22,448
SYS	DBMS_SHARED_POOL	PACKAGE	NO	1	22,244
SYS	V$SQLAREA	VIEW	NO	1	19,597
SYS	DBMS_UTILITY	PACKAGE	NO	1	19,592
SYS	DBMS_IJOB	PACKAGE	NO	1	16,853
SYS	V$DB_OBJECT_CACHE	VIEW	NO	1	16,629
SYS	DBMS_PIPE	PACKAGE	NO	1	15,853
SYS	DBMS_STANDARD	PACKAGE	NO	1	14,521
SYS	DBMS_IJOB	PACKAGE BODY	NO	1	13,853
SYS	DBMS_OUTPUT	PACKAGE	NO	1	13,703
SYS	DBMS_UTILITY	PACKAGE BODY	NO	1	13,480
SYS	DBMS_EXPORT_EXTENSION	PACKAGE	NO	1	12,561
SYS	DBMS_APPLICATION_INFO	PACKAGE	NO	1	12,129
SYS	DBMS_SQL	PACKAGE BODY	NO	1	11,900
SYS	DBMS_DEFER_IMPORT_INTERNAL	PACKAGE	NO	1	10,798

Grosse Anonymous PL/SQL Blocks

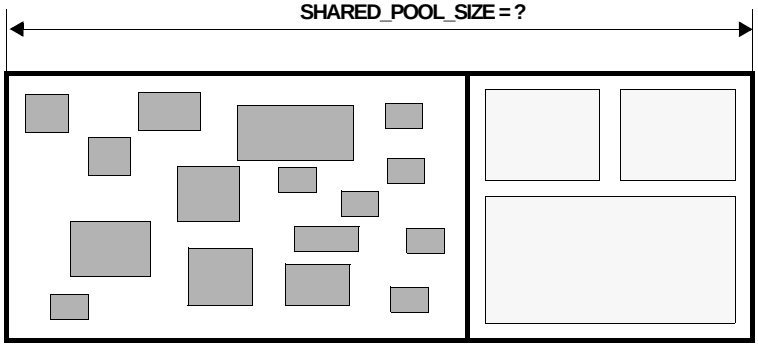
Grosse anonymous PL/SQL Blocks sollten in kleine Packages unterteilt werden, welche Funktionen aufrufen.

Identifikation von grossen anonymous PL/SQL Blocks:

```
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.



DB-Objects

Memory im Shared Pool für die Datenbank Objekte wie Packages.

```
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;

Space needed for DB-Objects in SP
---------------------------------
                           836,096
```

SQL

Memory im Shared Pool für normale SQL-Statements.

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

Space needed for SQL in SP
--------------------------
                25,617,655
```

Per-User and Per-Cursor

Pro concurrent User wird ca 250 Bytes benötigt.

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

Space needed for Users in SP
----------------------------
                      15,125
```

MTS

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

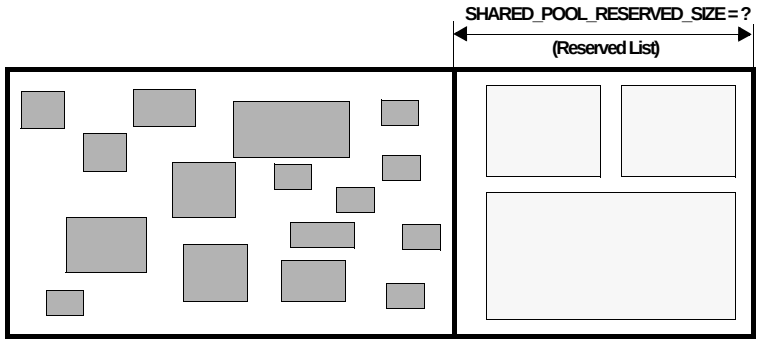
```
COLUMN total_mem FORMAT 999,999,999 HEADING 'Space needed for MTS in SP'
SELECT SUM(value) total_mem
  FROM v$sesstat s, v$statname n
 WHERE s.statistic# = n.statistic#
   AND n.name = 'session uga memory max';

Space needed for MTS in SP
--------------------------
                7,234,992
```

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

```
836,096 + 25,617,655 + 15,125 + 7,234,992 = 33,703,868
Zuzüglich 30 % Overhead                   = 10,111,160
-----------------------------------------------------
Total                                       40,815,028
=====================================================
```

Grösse des Reserved Pools bestimmen

Die Grösse des Reserved Pools ist abhängig von der Grösse und Menge der zu erwartenden grossen Memory Segmente, welche in der Applikation auftreten werden. Der optimale Wert kann meist erst während der Produktion festgelegt werden.



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;
```

Column	Description
FREE_SPACE	is the total amount of free space on the reserved list.
AVG_FREE_SIZE	is the average size of the free memory on the reserved list.
FREE_COUNT	is the number of free pieces of memory on the reserved list.
MAX_FREE_SIZE	is the size of the largest free piece of memory on the reserved list.
USED_SPACE	is the total amount of used memory on the reserved list.
AVG_USED_SIZE	is the average size of the of the used memory on the reserved list.
USED_COUNT	is the number of used pieces of memory on the reserved list.
MAX_USED_SIZE	is the size of the largest used piece of memory on the reserved list.
REQUESTS	is the number of times that the reserved list was searched for a free piece of memory.
REQUEST_MISSES	is the number of times the reserved list didn't have a free piece of memory to satisfy the request, and proceeded to start flushing objects from the LRU list.
LAST_MISS_SIZE	is the request size of the last REQUEST_MISS.
MAX_MISS_SIZE	is the request size of the largest REQUEST_MISS.

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

Column	Description
REQUEST_FAILURES	is the number of times that no memory was found to satisfy a request (e.g., number of times ORA-4031 occurred).
LAST_FAILURE_SIZE	is the request size of the last failed request (e.g., the request size of last ORA-4031).
ABORTED_REQUEST_THRESHOLD	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.
LAST_ABORTED_SIZE	is the last size of the request which returned an ORA-4031 error without flushing objects from the LRU list.

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()

```
CREATE OR REPLACE
Procedure  CHECK_SHARED_POOL IS

    /* Local variables and declarations */

   l_fhd           utl_File.File_Type;
   l_fdir          v$parameter.value%Type;
   l_sps           v$parameter.value%Type;
   l_sprs          v$parameter.value%Type;
   l_sprma         v$parameter.value%Type;
   l_fname         VARCHAR2(30);
   l_fmode         VARCHAR2(1) := 'W';
   l_open_time     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_perm      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_kghlu         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 spr_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_fname := 'chksp_' || TO_CHAR(SYSDATE,'DDMMYYYYHH24MI');
        l_fhd := utl_File.FOpen(l_fdir,l_fname,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'),
          TO_CHAR(SYSDATE,'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\n',l_open_time,l_current_time);
   utl_File.Putf(l_fhd,'                             %s\n',l_global_name);
   utl_File.Put_Line(l_fhd,'-------------------------------------------------------------------------------');
   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.Put_Line(l_fhd,' ');
   utl_File.Putf(l_fhd,'Current size of SHARED_POOL_SIZE ..... = %s [Bytes]\n', LPAD(l_sps,10,' '));
   utl_File.Putf(l_fhd,'Current size of SHARED_POOL_RESERVED_SIZE .... = %s [Bytes]\n', LPAD(l_sprs,10,' '));
   utl_File.Putf(l_fhd,'Current size of SHARED_POOL_RESERVED_MIN_ALLOC = %s [Bytes]\n', 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';
```

```
utl_File.Putf(l_fhd,'\n');
utl_File.Put_Line(l_fhd,'***************** Used space in SHARED POOL (approximative) ******************');
utl_File.Put_Line(l_fhd,' ');
utl_File.Putf(l_fhd,'Total size for Objects ..... = %s [Bytes]\n', LPAD(TO_CHAR(l_sum_obj),10,' '));
utl_File.Putf(l_fhd,'Total size for SQL-Statements = %s [Bytes]\n', LPAD(TO_CHAR(l_sum_sql),10,' '));
utl_File.Putf(l_fhd,'Total size for User Sessions  = %s [Bytes]\n', LPAD(TO_CHAR(l_sum_users),10,' '));
utl_File.Putf(l_fhd,'Total size for MTS ......... = %s [Bytes]\n', LPAD(TO_CHAR(l_sum_mts),10,' '));
utl_File.Putf(l_fhd,'Total size used ............ = %s [Bytes]\n',
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.
--------------------------------------------------------------- */

utl_File.Putf(l_fhd,'\n');
utl_File.Put_Line(l_fhd,'************* Memory Segments and their state in the SHARED POOL **************');
utl_File.Put_Line(l_fhd,' ');
utl_File.Put_Line(l_fhd,'Description      State               Sum (Size)      Max (Size)      Number of');
utl_File.Put_Line(l_fhd,'                                        [Bytes]         [Bytes]       Segments');
utl_File.Put_Line(l_fhd,'---------------- ------------ ----------------- --------------- --------------');

FOR ksmsp_rec IN ksmsp_cur LOOP
     utl_File.Putf(l_fhd,'%s%s%s%s%s\n', 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;



utl_File.Put_Line(l_fhd,'--------------------------------------------------------------------------------');
utl_File.Putf(l_fhd,'Total size of flushable memory segments = %s [Bytes]\n',
LPAD(TO_CHAR(l_sum_flush),10,' '));
utl_File.Putf(l_fhd,'Total size of memory segments in use .. = %s [Bytes]\n',
LPAD(TO_CHAR(l_sum_inuse),10,' '));
utl_File.Putf(l_fhd,'Total size of free memory segments .... = %s [Bytes]\n',
LPAD(TO_CHAR(l_sum_free),10,' '));
utl_File.Putf(l_fhd,'Total size of permanent memory segments = %s [Bytes]\n',
LPAD(TO_CHAR(l_sum_perm),10,' '));

/* ---------------------------------------------------------------
    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.Putf(l_fhd,'Average size of free memory segments .. = %s [Bytes]\n',
LPAD(TO_CHAR(l_size),10,' '));
utl_File.Putf(l_fhd,'Number free memory segments ..... = %s\n',
LPAD(TO_CHAR(l_sum_count),10,' '));
utl_File.Putf(l_fhd,'Average size / number free segments ... = %s\n',
LPAD(TO_CHAR(l_size/l_sum_count),10,' '));
utl_File.Put_Line(l_fhd,'--------------------------------------------------------------------------------');
utl_File.Putf(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,' ');
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.Putf(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.Putf(l_fhd,'%s\n', LPAD(TO_CHAR(big_objects_rec.sharable_mem),15,' '));
END LOOP;

/* Summary statistic of the shared pool operations */

utl_File.Putf(l_fhd,'\n');
utl_File.Put_Line(l_fhd,'*************** Summary statistic of the SHARED POOL operations ***************');
utl_File.Put_Line(l_fhd,' ');
utl_File.Put_Line(l_fhd,'LRU-Operations                  Flushes  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.Putf(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.Putf(l_fhd,'\n');
END LOOP;

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

utl_File.Putf(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,' ');
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.Putf(l_fhd,'%s%s%s%s\n', 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;
```

```
        ELSIF (flushed_obj_rec.ksmlrsiz > 10000
        AND flushed_obj_rec.ksmlrsiz < 20000) THEN
            l_message := 'ORA-4031 will be a serious problem soon !';
            l_updt_status := 2;
        ELSIF (flushed_obj_rec.ksmlrsiz > 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.Putf(l_fhd,'%s\n', l_message);
END IF;

/* Tuning of SHARED_POOL_RESERVED_SIZE */

utl_File.Putf(l_fhd,'\n');
utl_File.Put_Line(l_fhd,'******************** Tuning of SHARED_POOL_RESERVED_SIZE **********************');
utl_File.Put_Line(l_fhd,' ');
utl_File.Put_Line(l_fhd,'     FreeSpace  FreeSegm      UsedSpace  UsedSegm   Requests   Flushes ORA-4031');
utl_File.Put_Line(l_fhd,'       [Bytes]    Counts        [Bytes]    Counts     Counts    Counts   Counts');
utl_File.Put_Line(l_fhd,'-------------- --------- -------------- --------- ---------- --------- --------');

FOR spr_rec IN spr_cur LOOP
    utl_File.Putf(l_fhd,'%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,' '));
    utl_File.Putf(l_fhd,'%s%s\n', LPAD(spr_rec.request_misses,10,' '),
                                  LPAD(spr_rec.request_failures,9,' '));

    utl_File.Putf(l_fhd,'\n');
    IF (spr_rec.request_failures > 0) THEN
        l_message := 'Your System has ORA-4031 Errors !';
        l_updt_status := 3;
        utl_File.Putf(l_fhd,'%s\n', l_message);
    END IF;
    IF (spr_rec.last_failure_size > l_sprma
        AND spr_rec.max_free_size < l_sprma
        AND spr_rec.free_space < l_sprma) THEN
        l_message := 'Your SHARED_POOL_RESERVED_SIZE seems to bee too small !';
        l_updt_status := 2;
        utl_File.Putf(l_fhd,'%s\n', l_message);
    END IF;
    IF (spr_rec.request_misses = 0
        AND spr_rec.free_space > (l_sprma/2)) THEN
        l_message := 'Your SHARED_POOL_RESERVED_SIZE seems to be too big';
        utl_File.Putf(l_fhd,'%s\n', l_message);
    END IF;
    IF (spr_rec.request_failures > 0
        AND spr_rec.last_failure_size < l_sprma) THEN
        l_message := 'Your SHARED_POOL_SIZE is really too small !';
        l_updt_status := 3;
        utl_File.Putf(l_fhd,'%s\n', l_message);
    END IF;
END LOOP;
utl_File.Putf(l_fhd,'\n');
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 Überwachung

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

Description      State               Sum (Size)      Max (Size)      Number of
                                       [Bytes]         [Bytes]       Segments
---------------- ------------ ---------------- --------------- -------------
character set m  InUse                 32896           15152             9
dictionary cach  InUse               1376624            4256           814
fixed allocatio  Flushable               240              40             6
free memory      Free                6258376         6216568           743
free memory      R-free              8000000         8000000             1
KGL handles      Flushable           1596184             960          5272
kxfp subheap     Flushable               584             584             1
kzull            InUse                   304              64             6
library cache    InUse               2545592             584          8643
library cache    Flushable           2210656             584          4708
multiblock rea   InUse                   560             280             2
permanent memor  Permanent          16420872        16420872             1
PLS cca hp desc  InUse                   168             168             1
PLS non-lib hp   Flushable              2104            2104             1
PL/SQL DIANA     InUse               1044264            2112           626
PL/SQL DIANA     Flushable            114392            2104            72
PL/SQL MPCODE    InUse                449968            8400           179
PL/SQL MPCODE    Flushable             97344            2104            90
reserved stoppe  R-freea                 48              24             2
row cache lru    Flushable              1944              64            40
session param v  InUse                 31440            2096            15
sql area         InUse               7230464            4256          2333
sql area         Flushable           2559104            4264           808
table definiti   Flushable                64              64             1
trigger defini   Flushable              1080            1080             1
-------------------------------------------------------------------------------
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
-------------------------------------------------------------------------------
```

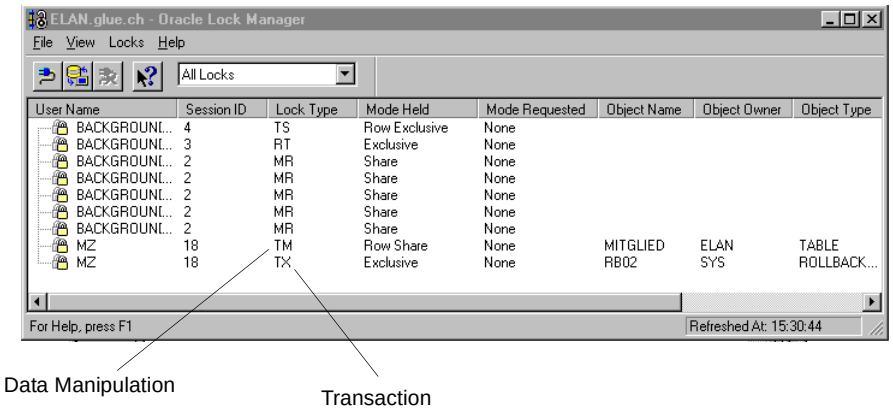
```
************** Big objects in the SHARED POOL (size > 10 KBytes) **************

Owner    Name                           Type          Kept Loads          Bytes
-------- ------------------------------ ------------- ---- ----- --------------
SYS      CHECK_SHARED_POOL              PROCEDURE     NO       1         140981
SYS      STANDARD                       PACKAGE       NO       1         120628
SIPP     CHECK_ACCM_MAX_TOTAL           PROCEDURE     NO       3          34577
SIPP     SUM_RELOAD                     PROCEDURE     NO       4          34559
SYS      STANDARD                       PACKAGE BODY  NO       1          28700
SYS      INDREORG                       PROCEDURE     NO       1          25028
SYS      DBMS_IREFRESH                  PACKAGE BODY  NO       1          22713
SYS      DBMS_SYS_SQL                   PACKAGE BODY  NO       1          22448
SYS      DBMS_SHARED_POOL               PACKAGE       NO       1          21956
SYS      V$SQLAREA                      VIEW          NO       1          19741
SYS      DBMS_UTILITY                   PACKAGE       NO       1          19664
SYS      DBMS_IREFRESH                  PACKAGE       NO       1          19057
SYS      V$SHARED_POOL_RESERVED         VIEW          NO       1          17538
PUBLIC   SSCORDER                       SYNONYM       NO       1          17351
SYS      X$KGHLU                        TABLE         NO       1          17163
SIPP     SEQ_ORDER_REQ_ORD_ID           SEQUENCE      NO      21          17105
SYS      X$KSMLRU                       TABLE         NO       1          17092
SIPP     SEQ_ESRLOG_LOG_ID              SEQUENCE      NO      21          16974

*************** Summary statistic of the SHARED POOL operations ***************

LRU-Operations                Flushes  Recurrent Segments Transient Segments
----------------------- --------------- ------------------ ------------------
                771534               0               2652               5611

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

Command                         Comment                   Size    Obj flushed
------------------------------- ------------------- ---------- --------------
No objects have been flushed out ... wonderful !

******************** Tuning of SHARED_POOL_RESERVED_SIZE **********************

     FreeSpace  FreeSegm      UsedSpace  UsedSegm   Requests   Flushes ORA-4031
       [Bytes]    Counts        [Bytes]    Counts     Counts    Counts   Counts
------------- --------- -------------- --------- ---------- --------- --------
       8000000         1              0         0          0         0        0

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.

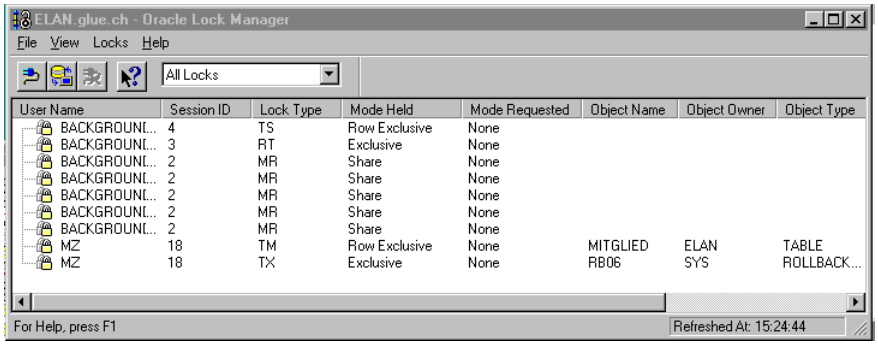


Beispiele

```
SELECT person_nr FROM mitglied FOR UPDATE OF pult_nr;
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:

MR	Media Recovery	IS	Instance State
RT	Redo Thread	FS	File Set
UN	User Name	IR	Instance Recovery
TX	Transaction	ST	Disk Space Transaction
TM	Data Manipulation	TS	Temp Segment
UL	PL/SQL User Lock	IV	Library cache invalidation
DX	Distributed Action	LS	Log Start/Switch
CF	Control File	RW	Row Wait
SQ	Sequence Number	TE	Extend Table
TT	Temp Table		

Lock Manager: Lock Types