HANSER

Leseprobe

Frank Haas

Oracle Tuning in der Praxis

Rezepte und Anleitungen für Datenbankadministratoren und -entwickler

ISBN: 978-3-446-41907-0

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-41907-0

sowie im Buchhandel.

5

5 Performance Tracing und Utilities

In diesem Kapitel werden die verschiedenen Methoden und Werkzeuge vorgestellt, mit denen Sie beim Tuning arbeiten; der Schwerpunkt liegt auf Methoden, die sich möglichst universell einsetzen lassen.

Die meisten der hier besprochenen Utilities lassen sich entweder direkt auf der Kommandozeile oder im SQL direkt aufrufen und sind somit von graphischen Interfaces wie Toad oder SQL*Developer unabhängig, obwohl sie auch dort verwendet werden können und sollen.

Es geht aber auch schöner. Abschließend werden die Tuning-Möglichkeiten des Enterprise Managers – genauer gesagt im Database Control – besprochen, lassen sich doch damit viele Tuning-Aufgaben bequem erledigen.

5.1 Utilities

Für das Oracle Tuning gibt es verschiedene nützliche Utilities, mit deren Hilfe wir nachschauen können, warum ein Statement gut oder schlecht läuft. In der Praxis interessiert uns aber meist nur Letzteres. In Oracle 10*g* existieren auch Utilities, die uns konkret beim Tuning unterstützen. Da wären also:

- EXPLAIN PLAN. Dieser SQL-Befehl erlaubt das Erzeugen eines Query Execution Plan. Der Ausführungsplan wird in eine Tabelle, die sinnigerweise PLAN_TABLE heißt, abgelegt und muss von dort selektiert werden. Seit Oracle 8.1 gibt es dafür Scripts. Vorher musste man das Select immer selbst erzeugen. Bereits seit Oracle 7.3 gibt es im SQL*Plus das Kommando AUTOTRACE. Damit lässt sich der Execution Plan automatisch anzeigen. Ab Oracle9 kann der Query Execution Plan direkt aus der Datenbank gezogen werden. Der aktuelle Plan kann dort über V$SQL_PLAN abgerufen werden. Seit 9.2 existiert das DBMS_XPLAN-Package, mit dem man einen Ausführungsplan auf einfache Art und Weise anazeigen kann. Eine sehr einfache Art der Darstellung von Ausführungsplänen bietet auch das Kommando SET AUTOTRACE

ON EXPLAIN in SQL*Plus. Bitte beachten Sie, dass EXPLAIN PLAN und auch AUTOTRACE den Ausführungsplan, wie er bei der Kompilierung der Anweisung entsteht, anzeigen. Dieser kann sich vom Ausführungsplan, wie er zur Laufzeit verwendet wird, unterscheiden, was insbesondere beim Einsatz von Bind-Variablen vorkommen kann. Deshalb sollten Sie, wann immer möglich, sich den aktuell verwendeten Plan mittels V$SQL_PLAN/DBMS_XPLAN anzeigen lassen.

- SQL_TRACE. Dieser ALTER SESSION-Parameter erlaubt das Erzeugen von Trace-Dateien, die wiederum mit dem TKPROF Utility formatiert werden können. Damit lassen sich neben dem Ausführungsplan auch die Ausführungszeiten für die einzelnen Phasen anzeigen. Sie können SQL_TRACE auch in init.ora/spfile setzen, aber dann wird alles getraced. Dies ist eine sehr effiziente Methode, schnell die Festplatte zu füllen, manchmal allerdings – speziell vor Oracle 10*g* – unumgänglich.
- In Oracle 10*g* können Sie sich die Ausführungspläne im laufenden Betrieb über V$SQL_PLAN_STATISTICS detailliert anzeigen lassen. Dazu müssen Sie allerdings STATISTICS_LEVEL auf ALL stellen.
- TKPROF. Mit diesem Utility werden Trace-Dateien, die mittels SQL_TRACE (oder 10046 Events) erzeugt wurden, in eine „lesbare“ Form gebracht.
- Event 10046. Erlaubt die Erzeugung detaillierter Trace-Dateien, die auch die Werte der Binds und Waits enthalten können. Das „Königsevent“ schlechthin bei der detaillierten Untersuchung.
- Seit Oracle 8*i* können die Werte von Bind-Variablen im laufenden Betrieb auch über V$SQL_BIND_DATA und V$SQL_BIND_METAVALUE abgefragt werden. Das funktioniert aber nur in der eigenen Session, was die Nützlichkeit erheblich einschränkt. Falls Sie aber selbst ein Trace-Programm in Ihrer Applikation schreiben wollen, ist das sicher eine in Betracht zu ziehende Möglichkeit.
- In Oracle 10*g* können die Werte von Bind-Variablen ebenfalls im laufenden Betrieb in V$SQL_BIND_CAPTURE abgefragt werden; dazu muss STATISTICS_LEVEL mindestens auf TYPICAL stehen. Bind-Variable für LONG und LOB können so nicht abgefragt werden, und es funktioniert nur für Bind-Variablen, die in den WHERE- oder HAVING-Klauseln verwendet werden, also auch nicht bei jeder Anweisung. Im Unterschied zu vorher können Sie sich damit auch die Werte für SQL-Anweisungen aus anderen Sessions anzeigen lassen.
- DBMS_MONITOR. Dieses Package ist neu in Oracle 10*g* und verbessert das Tracing speziell in Multi-Tier-Umgebungen. Des Weiteren können Sie damit in 10*g* auf einfache Art und Weise Event 10046 in anderen Sessions oder für die ganze Datenbank setzen. Über die Booleschen Parameter WAITS und BINDS können Sie alle Level setzen.
- Event 10053. Mit diesem Event lassen sich Trace-Dateien erzeugen, mit denen untersucht werden kann, warum der CBO einen bestimmten Plan erzeugt hat.
- Automatic Workload Repository (AWR) und Automatic Database Diagnostics Monitor (ADDM). In Oracle 10*g* sind diese Features aktiviert, sofern nicht jemand explizit STATISTICS_LEVEL auf BASIC umgestellt hat.

- Der Automatic Workload Repository-Bericht, Statspack und bstat/estat. Das sind alles mehr oder weniger periodische Auswertungen, die die Datenbank als Ganzes beleuchten.
- Der Active Session History-Bericht bietet Ihnen seit Version 10*g* die Möglichkeit, einen historischen Blick auf die Daten in V$SESSION_WAIT zu werfen.
- Awrsqrpt.sql und sprepsql.sql zeigen Ihnen die Ausführungspläne und Statistiken individueller SQL-Anweisungen, die im AWR respektive Statspack abgespeichert sind.

Das sind die wesentlichen Tracing-Methoden. Für besondere Untersuchungen stehen uns daneben noch speziellere Utilties zur Verfügung:

- Für das Tracen von PL/SQL kann der DBMS_PROFILER bzw. ab Version 11 DBMS_HPROF verwendet werden.
- Bei Verwendung von Shared Server können die Informationen für eine Session über mehrere Trace-Dateien verteilt sein, was die Untersuchung recht schwierig macht. Für diesen Zweck existiert das Tool trcsess, mit dem diese Dateien wieder anhand verschiedener Kriterien wie beispielsweise der Session ID, zusammengefügt werden können.
- Das SQL*Net Tracing untersucht den Netzwerkverkehr (sofern er über SQL*Net führt). Das ist manchmal auch sinnvoll für Performance-Untersuchungen. Wurde der SQL*Net Trace mit Level 16 erzeugt, kann er weiter mit dem Tool trcasst analysiert werden.
- Für die Untersuchung von Statements, die parallel abgearbeitet werden, stehen mehrere Events zur Verfügung. Ab Oracle 9*i* kann zusätzlich das _px_trace Event verwendet werden. Parallelisierung ist ein Kapitel für sich, deshalb sehen wir uns auch das Tracing nicht hier, sondern im Kapitel über Parallelisierung an.

5.2 Tuning mit den Advisories

Seit Oracle 10*g* ist das Tuning in einigen Bereichen stark erleichtert. Bereits beim Anlegen der Datenbank wird die entsprechende Infrastruktur eingerichtet. Gesteuert wird es über den Parameter STATISTICS_LEVEL. Der ist erst mal auf TYPICAL eingestellt, was auch in den meisten Fällen reichen sollte. Im neuen SYSAUX-Tablespace wird stündlich ein Schnappschuss des Systems im Automatic Workload Repository permanent gespeichert. Dabei werden die Daten über Aufrufe, die direkt in den Oracle-Kernel eingebaut sind, und einen eigenen Hintergrundprozess runtergeschrieben. Die Systembelastung ist also deutlich reduziert. AWR können Sie sich als Weiterentwicklung von Statspack vorstellen. Der Bericht ist dem Statspack-Bericht früherer Versionen ähnlich. Das Hinunterschreiben der AWR-Daten erfolgt automatisch jede Stunde, kann aber auch manuell in SQL*Plus direkt über die Prozedur DBMS_WORLOAD_REPOSITORY.CREATE_SNAPSHOT erfolgen. Jede Minute werden die Statistiken und Metriken nachgeführt. So können Sie zum Beispiel jede Minute in V$SESSMETRIC die beliebten „Cache Hit Ratios“ überprüfen. In Oracle 10*g* ist auch der Automatic Database Diagnostics Monitor (=ADDM) eingebaut. ADDM

identifiziert potenzielle Probleme und gibt Empfehlungen, wie diese Probleme zu lösen sind. Manchmal muss dafür separat ein Advisor aufgerufen werden. Immer wenn ein AWR Snapshot gezogen wird, wird auch ADDM aufgerufen. Falls Sie ADDM manuell ausführen wollen, können Sie das über das Script addmrpt.sql tun. Unter Unix in $ORACLE_HOME/rdbms/admin zu finden. Hier ein entsprechender Ausschnitt:

```
FINDING 1: 64% impact (382 seconds)
-----------------------------------

SQL statements consuming significant database time were found.   RECOMMENDATION 1:
SQL Tuning, 17% benefit (103 seconds)
      ACTION: Run SQL Tuning Advisor on the SQL statement with SQL_ID
"059q6c9f84f9k".
         RELEVANT OBJECT: SQL statement with SQL_ID 059q6c9f84f9k and
         PLAN_HASH 3556263355
         SELECT /*+NESTED_TABLE_GET_REFS+*/ "RAD"."RADUSAGE_0123".* FROM
         "RAD"."RADUSAGE_0123"
      RATIONALE: SQL statement with SQL_ID "059q6c9f84f9k" was executed 1
         times and had an average elapsed time of 103 seconds.
```

Wie sie sehen, quantifiziert ADDM seine Ergebnisse und gibt weitere Empfehlungen. Sie müssen aber den SQL Tuning Advisor, den wir im Folgenden genauer besprechen, in Version 10 nach wie vor manuell ausführen. Nur ADDM selbst wird automatisch ausgeführt.

In Oracle 11 läuft das anders. Dort läuft nachts während des Wartungsfensters ein automatischer Tuning Task, was Sie auch im Alert.log sehen. Hier ein entsprechender Ausschnitt:

```
Setting Resource Manager plan SCHEDULER[0x51B5]:DEFAULT_MAINTENANCE_PLAN via
scheduler window
Setting Resource Manager plan DEFAULT_MAINTENANCE_PLAN via parameter
Sat Jun 20 22:00:03 2009
Begin automatic SQL Tuning Advisor run for special tuning task
"SYS_AUTO_SQL_TUNING_TASK"
Sat Jun 20 22:00:39 2009
End automatic SQL Tuning Advisor run for special tuning task
"SYS_AUTO_SQL_TUNING_TASK"
```

Dieser Task holt sich zuerst aus den AWR-Daten die entsprechenden SQL-Anweisungen. Dabei werden im Wesentlichen Top-SQL-Anweisungen der letzten Woche, Top-SQL-Anweisungen der einzelnen Tage innerhalb der Woche und die Top-SQL-Anweisungen jeder Stunde in der letzten Woche berücksichtigt. Dies bedeutet jetzt aber nicht, dass hier jedes mögliche SQL genommen wird. Es gibt auch Ausnahmen. Beispielsweise bleiben parallele Abfragen, DML und DDL sowie Anweisungen, die kürzlich (im letzten Monat) getuned wurden, unberücksichtigt.

Nachdem die SQL-Anweisungen identifiziert wurden, werden sie mit Hilfe des SQL Tuning Advisor getuned. Zwar werden dabei alle möglichen Empfehlungen berücksichtigt, aber nur über ein SQL-Profile mögliche Verbesserungen werden automatisch auch getestet und implementiert. Dazu wird die Anweisung mit und ohne SQL-Profile ausgeführt. Ergibt sich eine dreifache Verbesserung, kann das SQL-Profile akzeptiert werden. Dreifache Verbesserung bezieht sich hier auf die Summe von CPU- und I/O-Zeit für die Anweisung. Alle anderen Empfehlungen, beispielsweise neue Indizes oder die Restrukturierung der SQL-Anweisung, werden zwar generiert, aber nicht implementiert. Der DBA muss sie

überprüfen und implementieren. Die entsprechenden Ergebnisse können Sie sich dann mit Hilfe der Prozedur DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK ansehen bzw. die Informationen aus DBA_ADVISOR_EXECUTIONS, DBA_ADVISOR_SQLSTATS und DBA_ADVISOR_SQLPLANS holen.

Der Tuning Task selbst kann über DBMS_AUTO_TASK_ADMIN an- und abgeschaltet werden. Verschiedene Parameter, die den Task selbst betreffen, können Sie über die Prozedur DBMS_SQLTUNE.SQT_TUNING_TASK_PARAMETER setzen. Hier ein Beispiel, mit dem wir das Zeitlimit für die Ausführung der einzelnen SQL-Anweisung während des automatischen Tunings auf 1500 Sekunden setzen:

```
exec dbms_sqltune.set_tuning_task_parameter('SYS_AUTO_SQL_TUNING_TASK',
'LOCAL_TIME_LIMIT', 1400);
```

In Oracle 11 wurde ADDM weiter ausgebaut, ADDM analysiert jetzt auch die ganze Datenbank – und nicht nur die einzelne Instanz – im RAC-Umfeld, und das DBMS_ADDM Package wurde eingeführt, mit dem sich einfacher als mit DBMS_ADVISOR bestimmte Aktionen ausführen lassen. So können Sie damit auch Direktiven setzen, die den ADDM Task (oder alle folgenden) beeinflussen. Hier ein Beispiel:

```
var tname VARCHAR2(60);

BEGIN
  DBMS_ADDM.INSERT_FINDING_DIRECTIVE(   NULL,    'Probleme im Temp',
   'Temp Space Contention',    2,    10);
  :tname := 'temp_analyze';
  DBMS_ADDM.ANALYZE_INST(:tname, 6070, 6080);
END;
/
```

In diesem Beispiel sagen wir, dass wir im ADDM-Bericht die Empfehlung für „Temp Space Contention“ nur sehen wollen, wenn sie in der Analyseperiode für mindestens 2 aktive Sessions gültig ist und mindestens 10% der gesamten Datenbankzeit beansprucht hat. Woher wissen Sie jetzt, dass die entsprechende Empfehlung „Temp Space Contention“ heißt? Ganz einfach: die Namen all dieser Empfehlungen finden Sie im Data Dictionary in DBA_ADVISOR_FINDING_NAMES:

```
select finding_name from dba_advisor_finding_names;
```

Diese Namen unterscheiden nach Groß- und Kleinschreibung. Sie können auch Direktiven für Parameter oder SQL-Anweisungen setzen oder den Segment Advisor beeinflussen.

Oracle 10*g* führte schließlich mit dem SQL Tuning Advisor und dem SQL Access Advisor zwei Utilities ein, die dem Benutzer konkrete Vorschläge beim Tuning unterbreiten. Endlich! Für diese Ratgeber ist als hauptsächliches Interface der Enterprise Manager vorgesehen, man kann die Packages aber auch direkt verwenden. Hier handelt es sich um die DBMS_ADVISOR- und DBMS_SQLTUNE-Packages. Der Benutzer braucht dafür die beiden Privilegien ADVISOR und ADMINISTER SQL TUNING SET. Falls Sie die Empfehlungen gleich in Dateien schreiben wollen, benötigen Sie noch CREATE DIRECTORY. Fangen wir mit DBMS_SQLTUNE an. Ich habe dafür das folgende SQL*Plus-Script verwendet:

```
variable my_empfehlung clob
set long 2000
set longchunksize 100
set linesize 100

declare
sqltext clob;
retval varchar2(4000);
begin
sqltext := 'select count(e.ename), d.dname from emp e, dept d where exists
      (select d2.dname from dept d2 where d2.deptno = e.deptno) group by d.dname';
retval:=dbms_sqltune.create_tuning_task(sqltext,NULL,'SCOTT',time_limit=>10);
dbms_sqltune.execute_tuning_task(retval);
:my_empfehlung:= dbms_sqltune.report_tuning_task(retval);
end;
/
```

Zuerst habe ich das übrigens anders probiert. Wenn Sie jedoch versuchen, DBMS_SQL TUNE.CREATE_TUNING_TASK direkt in SQL zu verwenden, bekommen Sie den kryptischen Fehler ORA-14552. So hat es aber funktioniert. Ich gab für das Tuning in der Prozedur CREATE_TUNING_TASK noch ein Zeitlimit von 10 Sekunden an; voreingestellt sind 60 Minuten. Das Ergebnis dieses Tunings steht nach diesem Ablauf in der Variable my_empfehlung. Hier der entsprechende Ausschnitt:

```
…
1 - Restructure SQL finding (see plan 1 in explain plans section)
--------------------------------------------------------------------
  An expensive cartesian product operation was found at line ID 3 of the
execution plan.
 Recommendation
  --------------
  - Consider removing the disconnected table or view from this statement or
    add a join condition which refers to it.

  Rationale
  ---------
    A cartesian product should be avoided whenever possible because it is an
    expensive operation and might produce a large amount of data.
…
```

Immerhin, gar nicht so schlecht, wiewohl es für den Mann der Praxis offensichtlich ist. In Version 10.1.0.3 gab es für diese Anweisung übrigens noch keine Empfehlungen.

Was kommt heraus, wenn wir über DBMS_ADVISOR vorgehen? Dort existiert für das Tuning von SQL-Anweisungen die Prozedur QUICK_TUNE:

```
variable my_empfehlung clob
set long 2000
set longchunksize 100
set linesize 100

declare
sqltext clob;
taskname varchar2(4000) := 'Quick Task';
begin
sqltext := 'select count(e.ename), d.dname from emp e, dept d where exists
     (select d2.dname from dept d2 where d2.deptno = e.deptno) group by d.dname';
dbms_advisor.quick_tune(DBMS_ADVISOR.SQLACCESS_ADVISOR,taskname,sqltext);
:my_empfehlung:=dbms_advisor.get_task_script(taskname);
end;
/
```

Als Parameter zu QUICK_TUNE habe ich hier auch übergeben, welcher Advisor verwendet werden soll. Falls das Ergebnis Sie überrascht: vor dem Lauf der Advisories hatte ich die Tabelle EMP über mehrere INSERT INTO EMP SELECT * FROM EMP aufgeblasen. Sowohl mit Version 10.2 als auch mit Version 11.1 erhielt ich übrigens das gleiche Ergebnis:

```
print my_empfehlung

MY_EMPFEHLUNG
------------------------------------------------------------------------
Rem  SQL Access Advisor: Version11.1.0.7.0 - Production
Rem
Rem  Username:        SCOTT
Rem  Task:            Quick Task
Rem  Execution date:  Rem
CREATE MATERIALIZED VIEW LOG ON "SCOTT"."EMP" WITH ROWID,
SEQUENCE("ENAME","DEPTNO") INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON "SCOTT"."DEPT" WITH ROWID, SEQUENCE("DNAME")
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW "SCOTT"."MV$$_090B0001"
REFRESH FAST WITH ROWID
ENABLE QUERY REWRITE
AS SELECT SCOTT.EMP.DEPTNO C1, SCOTT.DEPT.DNAME C2, COUNT("SCOTT"."EMP"."ENAME")
M1,COUNT(*) M2
FROM SCOTT.EMP, SCOTT.DEPT
GROUP BY SCOTT.EMP.DEPTNO, SCOTT.DEPT.DNAME;

begin
dbms_stats.gather_table_stats('"SCOTT"','"MV$$_090B0001"',NULL,
dbms_stats.auto_sample_size);
end;
/
```

DBMS_ADVISOR empfiehlt also das Anlegen einer Materialized View. Falls diese Abfrage oft ausgeführt würde, wäre das auch sinnvoll. Beachten Sie bitte, dass der Advisor gleich das Script zum Erstellen der Materialized View mitgeliefert hat. DBMS_ADVISOR ist ein sehr mächtiges und umfangreiches Package. Den etwas unschönen Namen für die Materialized View hätte man zum Beispiel über das Anpassen des Parameters MVIEW_NAME_TEMPLATE gleich beim Erstellen verschönern und über den Parameter DEF_MVIEW_TABLESPACE auch gleich angeben können, in welchem Tablespace die Materialized View erzeugt werden soll.

Wir hätten uns auch gleich das Script über die Prozedur CREATE_FILE in eine Datei schreiben lassen können. Ich hätte auch angeben können, ob ich die Empfehlungen eher für eine OLTP oder eher für eine Data Warehouse oder eine gemischte Umgebung möchte. Dadurch wird dann beispielsweise auch festgelegt, wie stark Bitmap-Indizes berücksichtigt werden. Neben QUICK_TUNE existiert noch die TUNE_MVIEW Prozedur, mit der man schnell Materialized Views analysieren kann. Das Ergebnis unterscheidet sich auch, je nachdem, welche Version Sie verwenden. In Version 10.1 wurde hier nur eine Materialized View ohne Materialized View Logs empfohlen. Das führt dann zu einer Materialized View, die immer komplett aufgefrischt werden muss. Allerdings setzt seit Version 10.2 ein Fast Refresh nicht mehr zwingend Materialized View Logs voraus; falls PCT (= Partition Change Tracking) auf einer partitionierten Tabelle aktiviert ist, kann darauf verzichtet wer-

den. In Oracle 11 wurde DBMS_ADVISOR weiter ausgebaut. Dort untersucht DBMS_ADVISOR dann auch, ob die Partitionierung einer nicht partitionierten Tabelle oder eines nicht partitionierten Index vorteilhaft sein könnte. Was allerdings voraussetzt, dass die Tabelle mindestens 10000 Zeilen hat und in der SQL-Anweisung Prädikate und/oder Joins vorkommen, bei denen die betroffenen Spalten als Datentyp NUMBER oder DATE haben. Das hierbei entstehende Script verwendet dann oft das Package DBMS_REDEFINITION für die Implementierung.

Idealerweise setzen Sie DBMS_ADVISOR aber zusammen mit AWR ein. Nehmen wir mal an, Sie möchten eine bestimmte Abfrage tunen. Zufälligerweise handelt es sich um die vorige Abfrage, die wir jetzt in ein Script packen. Sie führen diese Abfrage jetzt mehrfach zwischen zwei Schnappschüssen vom System aus. Um sicherzustellen, dass keine Daten bereits im Hauptspeicher sind, führen wir auch noch ein Flush des Buffer Cache aus. Diese Anweisung ist neu in 10*g*:

```
alter system flush buffer_cache;
exec DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT;
@abfrage
@abfrage
@abfrage
exec DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT;
```

Für das Tuning ist jetzt also der Zeitraum zwischen den beiden letzten Schnappschüssen interessant, was wir über die folgende SQL-Anweisung erfahren:

```
variable snap_id number;
begin
 select max(snap_id) into :snap_id from dba_hist_snapshot;
end;
/
```

Als Nächstes legen wir ein SQL Tuning Set an und laden dort die Abfragen, die wir tunen wollen, hinein. Uns interessieren nicht alle Abfragen zwischen den beiden letzten Schnappschüssen. Deshalb schränken wir die Abfragen ein auf die mehr als einmal ausgeführten und bei denen mehr als 100 mal von der Festplatte gelesen wurde. Andere Auswahlkriterien wären hier zum Beispiel die Anzahl der Bufferzugriffe oder Elapsed Time. Wir laden die Anweisungen direkt aus dem AWR. Eine andere Möglichkeit bestünde im Laden der Anweisungen aus dem Cursor Cache über die Prozedur DBMS_SQLTUNE.SELECT_SQLSET. Wir laden die Anweisungen im Beispiel über einen Cursor:

```
declare
baseline_ref_cursor DBMS_SQLTUNE.SQLSET_CURSOR;
begin
 DBMS_SQLTUNE.CREATE_SQLSET('MY_SQL_TUNING_SET');
 open baseline_ref_cursor for
 select VALUE(p) from table(DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY
(:snap_id - 1, :snap_id,
             'executions > 1 and disk_reads > 100',NULL,'disk_reads')) p;
 DBMS_SQLTUNE.LOAD_SQLSET('MY_SQL_TUNING_SET', baseline_ref_cursor);
end;
/
```

Falls Sie jetzt nachschauen wollen, ob unsere Abfrage im SQL Tuning Set auch wirklich dabei ist, funktioniert dies wie folgt:

```
set long 1024
select SQL_TEXT from table(DBMS_SQLTUNE.SELECT_SQLSET('MY_SQL_TUNING_SET'));
```

Damit ist das meiste vorbereitet, jetzt können wir das Tuning durchführen. Dazu legen wir über DBMS_ADVISOR.CREATE_SQLWKLD einen SQL Workload an und importieren in dieses Workload unser Tuning Set. Dann erzeugen wir eine Tuning Task für den SQL Access Advisor, assoziieren über DBMS_ADVISOR.ADD_SQLWKLD_REF den Workload mit der Tuning Task und führen schließlich die Tuning Task aus:

```
VARIABLE name VARCHAR2(20)
VARIABLE saved_stmts NUMBER;
VARIABLE failed_stmts NUMBER;
VARIABLE task_id NUMBER
VARIABLE task_name VARCHAR2(255)

begin
     :name := 'MY_NEW_WORKLOAD';
     DBMS_ADVISOR.CREATE_SQLWKLD(:name);
     DBMS_ADVISOR.IMPORT_SQLWKLD_STS('MY_STS_WORKLOAD', 'MY_SQL_TUNING_SET',
'NEW', 1, :saved_stmts, :failed_stmts);
     :task_name := 'NEW_SQLACCESS_TASK';
     DBMS_ADVISOR.CREATE_TASK ('SQL Access Advisor',:task_id,:task_name);
     DBMS_ADVISOR.ADD_SQLWKLD_REF('NEW_SQLACCESS_TASK','NEW_STS_WORKLOAD');
     DBMS_ADVISOR.EXECUTE_TASK('NEW_SQLACCESS_TASK');
end;
/
```

Jetzt können wir uns die Empfehlungen des SQL Access Advisor zu Gemüte führen. Mit der folgenden Abfrage sehen wir die Wichtigkeit (Rank) und die Verbesserung in den Optimizerkosten (Benefit) für jede Empfehlung:

```
SQL> SELECT rec_id, rank, benefit
 2 FROM user_advisor_recommendations
 3 WHERE task_name = :task_name;

 REC_ID     RANK       BENEFIT
---------- ---------- ----------
 1          1          8208
```

Wir sehen, es gibt nur eine Empfehlung. Wie viele Kosten würden wir uns dadurch ersparen?

```
SQL> SELECT sql_id, rec_id, precost, postcost, (precost-postcost)*100/precost AS
percent_benefit
 2 FROM user_advisor_sqla_wk_stmts
 3 WHERE task_name = :task_name AND workload_name = :name;

    SQL_ID     REC_ID    PRECOST   POSTCOST PERCENT_BENEFIT
---------- ---------- ---------- ---------- ---------------
        21          1         33         21      36.3636364
```

Sehen wir uns an, was dabei konkret herauskam. Überraschenderweise empfiehlt uns der SQL Access Advisor das Anlegen einer Materialized Views und der entsprechenden Materialized View Logs. Wer hätte das gedacht?

```
SQL> SELECT rec_id, action_id, substr(command,1,30) AS command
 2 FROM user_advisor_actions
 3 WHERE task_name = :task_name
 4 ORDER BY rec_id, action_id;

    REC_ID  ACTION_ID COMMAND
```

```
---------- ---------- ------------------------------
         1          1 CREATE MATERIALIZED VIEW LOG
         1          3 CREATE MATERIALIZED VIEW LOG
         1          5 CREATE MATERIALIZED VIEW
         1          6 GATHER TABLE STATISTICS
```

Das wollen wir jetzt noch im Detail sehen und geben die Empfehlungen als Script aus:

```
create directory advisor_results as '/tmp';
execute
DBMS_ADVISOR.CREATE_FILE (DBMS_ADVISOR.GET_TASK_SCRIPT(:task_name), -
 'ADVISOR_RESULTS', 'implement_script.sql');
```

Man kann sich auch gleich ein Undo-Script – das also die ganzen Veränderungen wieder zurücknimmt – generieren lassen. Das erfolgt dann über das Schlüsselwort „UNDO“ als zweitem Parameter in DBMS_ADVISOR.GET_TASK_SCRIPT. In Version 10.1.0.3 ist dieses Script aber leer, das ist Oracle Bug 3117513. Hier aber erst mal das Script mit den empfohlenen Änderungen – bei meinem Test erhielt ich identische Ergebnisse mit den Versionen 10.2 und 11.1:

```
Rem  SQL Access Advisor: Version 11.1.0.7  - Production
Rem
Rem  Username:        SCOTT
Rem  Task:            NEW_SQLACCESS_TASK
Rem  Execution date:  22/06/2009 13:11
Rem

CREATE MATERIALIZED VIEW LOG ON "SCOTT"."EMP"
WITH ROWID, SEQUENCE("ENAME","DEPTNO") INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON "SCOTT"."DEPT"
WITH ROWID, SEQUENCE("DNAME") INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW "SCOTT"."MV$$_09810001"
REFRESH FAST WITH ROWID
ENABLE QUERY REWRITE
AS SELECT SCOTT.EMP.DEPTNO C1, SCOTT.DEPT.DNAME C2, COUNT("SCOTT"."EMP"."ENAME")
M1, COUNT(*) M2 FROM SCOTT.EMP, SCOTT.DEPT GROUP BY SCOTT.EMP.DEPTNO,
SCOTT.DEPT.DNAME;

begin
dbms_stats.gather_table_stats('"SCOTT"','"MV$$_09810001"',NULL,
dbms_stats.auto_sample_size);
end;
/
```

Es überrascht uns nicht allzu sehr, dass das die gleichen Empfehlungen sind, die wir zuvor bereits über QUICK_TUNE erzeugten. Dem aufmerksamen Leser liegt jetzt aber eine ganz andere Problematik am Herzen. Konsultieren wir unsere Abfrage:

```
select count(e.ename), d.dname from emp e, dept d where exists
     (select d2.dname from dept d2 where d2.deptno = e.deptno) group by d.dname;
```

In der Hauptabfrage haben wir einen kartesischen Join, und in der Unterabfrage joinen wir noch mal die Tabelle DEPT mit der Tabelle EMP aus der Hauptabfrage. Jedem Mitarbeiter in der Tabelle EMP ist eine Abteilung aus der Tabelle DEPT zugeordnet. Als Ergebnis bekommen wir also die Anzahl der Datensätze aus der Tabelle EMP und die Namen der verschiedenen Abteilungen. Wäre es nicht wesentlich einfacher und besser gewesen, einfach ein SELECT COUNT(*) FROM EMP und dann ein SELECT DISTINCT DNAME FROM DEPT durchzuführen?

Die Ratgeber nehmen Ihnen also nicht das Denken ab. Zwar wird das Tuning durch sie sehr erleichtert, man braucht aber immer noch jemanden, der entscheiden kann, ob etwas sinnvoll ist oder nicht.

5.3 EXPLAIN PLAN

Mit der SQL-Anweisung EXPLAIN PLAN erzeugen Sie einen Ausführungsplan. Damit EXPLAIN PLAN jedoch überhaupt funktioniert, muss es eine Plantabelle mit einer bestimmten Struktur, die je nach Release unterschiedlich sein kann, geben. Die Tabelle heißt eigentlich immer PLAN_TABLE und sieht seit Version 10.2 so aus:

```
SQL> desc plan_table;

 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------
 STATEMENT_ID                                       VARCHAR2(30)
 PLAN_ID                                            NUMBER
 TIMESTAMP                                          DATE
 REMARKS                                            VARCHAR2(4000)
 OPERATION                                          VARCHAR2(30)
 OPTIONS                                            VARCHAR2(255)
 OBJECT_NODE                                        VARCHAR2(128)
 OBJECT_OWNER                                       VARCHAR2(30)
 OBJECT_NAME                                        VARCHAR2(30)
 OBJECT_ALIAS                                       VARCHAR2(65)
 OBJECT_INSTANCE                                    NUMBER(38)
 OBJECT_TYPE                                        VARCHAR2(30)
 OPTIMIZER                                          VARCHAR2(255)
 SEARCH_COLUMNS                                     NUMBER
 ID                                                 NUMBER(38)
 PARENT_ID                                          NUMBER(38)
 DEPTH                                              NUMBER(38)
 POSITION                                           NUMBER(38)
 COST                                               NUMBER(38)
 CARDINALITY                                        NUMBER(38)
 BYTES                                              NUMBER(38)
 OTHER_TAG                                          VARCHAR2(255)
 PARTITION_START                                    VARCHAR2(255)
 PARTITION_STOP                                     VARCHAR2(255)
 PARTITION_ID                                       NUMBER(38)
 OTHER                                              LONG
 OTHER_XML                                          CLOB
 DISTRIBUTION                                       VARCHAR2(30)
 CPU_COST                                           NUMBER(38)
 IO_COST                                            NUMBER(38)
 TEMP_SPACE                                         NUMBER(38)
 ACCESS_PREDICATES                                  VARCHAR2(4000)
 FILTER_PREDICATES                                  VARCHAR2(4000)
 PROJECTION                                         VARCHAR2(4000)
 TIME                                               NUMBER(38)
 QBLOCK_NAME                                        VARCHAR2(30)
```

Das entsprechende CREATE TABLE-Script zum Anlegen der Tabelle ist utlxplan.sql. Es ist im Verzeichnis $ORACLE_HOME/rdbms/admin zu finden. Streng genommen könnte man die Plantabelle auch anders benennen, solange nur die Struktur stimmt. Beim Kommando EXPLAIN PLAN kann man mit der INTO-Klausel angeben, in welcher Tabelle die Plandaten abgelegt werden sollen, was aber nicht sehr sinnvoll ist, denn dann müsste man alle Ausgabescripts ebenfalls entsprechend anpassen. Man kann dem Statement in der

Plantabelle noch einen Namen geben, was über EXPLAIN PLAN SET STATEMENT_ID funktioniert, aber nur sinnvoll ist, wenn man die Ausführungspläne in der Plantabelle aufbewahren will, denn die Ausgabeskripts müssen dann ebenfalls angepasst werden. Ansonsten ist das Kommando recht einfach anzuwenden.

```
SQL> explain plan for select * from emp;
EXPLAIN PLAN ausgeführt.
```

Das war's schon. Für die Ausgabe aus der Plantabelle braucht man im Regelfall ein Skript, da ein SELECT * FROM PLAN_TABLE schlichtweg bescheiden aussieht. Damit kann man nichts anfangen. In Oracle 6 sah das noch so aus:

```
SQL> select lpad(' ',2*level)||operation||' '||options||' '||object_name "Query
Plan"
 2 from plan_table
 3 connect by prior id = parent_id start with id = 1;
```

Interessant bei dieser Abfrage ist die Funktion LPAD und das CONNECT BY. Mit dem LPAD erzwingt man, dass bei mehrstufigen Plänen die einzelnen Levels eingerückt werden. Hier natürlich nicht, da es nur einen Full Table Scan gibt. Mit CONNECT BY sind so genannte hierarchische Abfragen möglich, bei denen die Daten in einer bestimmten Zuordnung zueinander stehen. Man kann auch sagen, dass damit Baumstrukturen traversiert werden können, aber das hilft Ihnen ja auch nicht unbedingt weiter, oder? Das kann sehr nützlich sein, damit können Sie stücklistenartige Strukturen abarbeiten. Ein Beispiel hierfür ist die Stückliste für mein Moped. Das Moped besteht aus Rahmen, zwei Rädern, einem Motor und einigen weiteren Kleinigkeiten. Diese einzelnen Komponenten sind wieder aus mehreren Teilen zusammengesetzt. Der Motor zum Beispiel besteht aus Getriebe, Kupplung, Nockenwelle etc. Diese Teile können auch wieder zerlegt werden, bis wir zum Schluss nur noch einzelne Teile haben, die sich nicht weiter auseinandernehmen lassen.

Aufpassen müssen Sie, wenn Sie Bind-Variablen verwenden. Bind-Variablen sind Variablen, die erst zur Laufzeit mit Werten gefüllt werden. Dieselbe Query, die Sie einmal mit Bind-Variablen ausführen und dann wieder mit konstanten Werten, kann völlig unterschiedliche Ausführungspläne liefern. EXPLAIN PLAN nimmt immer an, dass bei einer Bind-Variable der Datentyp VARCHAR2 verwendet wird. Auf das Beispiel übertragen, bedeutet dies: die Anweisung SELECT * FROM EMP WHERE EMPNO=7500 kann einen ganz anderen Ausführungsplan haben als die Anweisung SELECT * FROM EMP WHERE EMPNO = :my_variable. Ab Oracle 9*i* berücksichtigt der Optimizer auch die aktuellen Werte von Bind-Variablen, aber nur beim ersten Mal, wenn er den Execution Plan generiert. In der Version 8*i* müssen Sie aufpassen, wenn Sie Histogramme auf einzelnen Spalten haben, die in dieser Version nicht von Bind-Variablen berücksichtigt werden.

Für die Ausgabe aus der Plantabelle liefert Oracle seit Version 8.1.5 gleich die beiden passenden Scripts mit. Es sind dies:

1. $ORACLE_HOME/rdbms/admin/utlxplp.sql – Ausgabe eines Plans mit Darstellung paralleler Operationen;
2. $ORACLE_HOME/rdbms/admin/utlxpls.sql – Ausgabe eines „normalen" seriellen Plans.

EXPLAIN PLAN gehört zum ANSI SQL-Standard, das Kommando gibt es also schon ewig. Man kann sich den Ausführungsplan ab 9.2 auch sehr einfach über das Package DBMS_XPLAN anzeigen lassen. Diese Abfrage zeigt den Ausführungsplan für die letzte EXPLAIN PLAN-Anweisung:

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

Wenn Sie das Script utlxpls.sql ausführen, wird diese Abfrage ausgeführt. In Oracle 10*g* wurde das noch mal erweitert. Dort lassen sich über DISPLAY_CURSOR der Ausführungsplan für beliebige SQL-Anweisungen aus dem Cursor Cache und über DISPLAY_AWR der Ausführungsplan für beliebige SQL-Anweisungen aus dem AWR anzeigen. Der Benutzer braucht die SELECT_CATALOG-Rolle für diese beiden Funktionen.

Komfortabel geht es seit Oracle 7.3 mit dem Kommando SET AUTOTRACE im SQL*Plus. Das sieht dann so aus:

```
SQL> set autotrace traceonly explain
SQL> select * from emp;

Execution Plan
----------------------------------------------------------
Plan hash value: 3956160932

--------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |    14 |   518 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| EMP  |    14 |   518 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------------
```

Das ist jetzt ein mit Version 10.2 erstelltes Beispiel. Dort sehen Sie auch, wie viel Zeit im jeweiligen Schritt verbraucht wird (in Version 11 sieht das auch so aus). Diese Information wird in früheren Versionen nicht angezeigt, zum Vergleich das Ergebnis, wenn Sie den AUTOTRACE in Version 10.1 ausführen:

```
Ausführungsplan
----------------------------------------------------------
 0 SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=15 Bytes=495)
 1 0 TABLE ACCESS (FULL) OF 'EMP' (Cost=2 Card=15 Bytes=495)
```

Beachten Sie bitte: SET AUTOTRACE TRACEONLY! Ich hätte auch SET AUTOTRACE ON EXPLAIN schreiben können, dann wären die Daten ebenfalls gelesen worden, was je nachdem, was man gerade vorhat, ziemlich viel Zeit in Anspruch nehmen kann. Deshalb verwende ich bei solchen Untersuchungen im Regelfall immer die TRACEONLY-Option. Eine interessante Variante ist SET AUTOTRACE ON EXPLAIN STATISTICS. Damit werden auch ausgewählte statistische Informationen ausgegeben:

```
SQL> set autotrace on explain statistics
SQL> select * from dept

....

Execution Plan
----------------------------------------------------------
Plan hash value: 3383998547
```

```
-----------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     4 |    80 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| DEPT |     4 |    80 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------

Statistics
----------------------------------------------------------
        216  recursive calls
          0  db block gets
         46  consistent gets
          2  physical reads
          0  redo size
        647  bytes sent via SQL*Net to client
        381  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          6  sorts (memory)
          0  sorts (disk)
          4  rows processed
```

SET AUTOTRACE TRACEONLY EXPLAIN STATISTICS ergibt keinen Sinn, auch wenn Sie das Kommando absetzen können. Damit die Statistiken angezeigt werden können, muss die Anweisung zuerst ausgeführt werden. Man könnte auch nur STATISTICS verwenden, aber ohne Execution Plan ergibt das meiner Meinung nach keinen Sinn. Aufpassen müssen Sie, wenn Sie nur SET AUTOTRACE ON verwenden. Damit aktivieren Sie alles. Was bedeutet: das Statement wird ausgeführt, der Execution Plan angezeigt und die Statistiken ebenfalls. Das kann dann unter Umständen sehr lange dauern. Wenn Sie eine Anweisung optimieren wollen, die mehrere Stunden läuft, können Sie nur mit SET AUTOTRACE TRACEONLY EXPLAIN vernünftig arbeiten.

Unschön bei der ganzen AUTOTRACE-Geschichte ist die Tatsache, dass nach 80 Zeichen ein Zeilenumbruch erfolgt, was insbesondere bei komplexeren Plänen die Lesbarkeit stark beeinträchtigen kann. Allerdings lässt sich das seit Version 10.2 mittels SET LINESIZE beeinflussen. Hier ein einfaches Beispiel. Aus Darstellungsgründen wurde der Ausführungsplan editiert:

```
SQL>select e.ename,e.empno,d.dname
..2   from emp e, dept d
  3    where e.empno <2500
  4*   and e.deptno = d.deptno

Execution Plan
----------------------------------------------------------
Plan hash value: 351108634

-------------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     1 |    26 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS     |         |     1 |    26 |     4   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| EMP    |     1 |    13 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPT  ............
|*  4 |    INDEX UNIQUE SCAN         | PK_DEPT | ............
-------------------------------------------------------------------------------

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

   2 - filter("E"."EMPNO"<2500)
   4 - access("E"."DEPTNO"="D"."DEPTNO")
```

Bei diesem Plan sehen Sie am Anfang auch noch die Statement IDs und Levels der einzelnen Zeilen. Ganz oben bei 0 ist das Statement selbst, hier also ein SELECT. Die erste Zeile im Ausführungsplan ist noch relativ interessant. Die gibt über Cost einen Hinweis, wie „teuer“ Oracle das Statement einschätzt. Falls Sie bei Cost keine Zahl sehen, können Sie im Regelfall davon ausgehen, dass dieser Execution-Plan NICHT verwendet wurde.

In Version 9.2 oder früher ist die Darstellung der Ausführungspläne natürlich auch noch anders, dort wurde beispielsweise nicht die Zeit in jedem einzelnen Operationsschritt ausgewiesen. Die wesentlichen Informationen sind aber auch dort verfügbar.

Ausführungspläne werden von rechts nach links gelesen. Das bedeutet: die Schritte ganz rechts werden als Erstes ausgeführt; anschließend diejenigen, die davor und dann so weiter. Hier im obigen Beispiel werden also zuerst über den Index-Scan in Schritt 4 die Primärschlüssel der Tabelle DEPT bestimmt. Das Ergebnis dieser Operation ist der Input für den Nested Loop, innerhalb dessen die Schritte 2 und 3 abgearbeitet werden. Die beiden Tabellen DEPT und EMP werden dabei, wie im Prädikat 4 zu sehen ist, über einen Equijoin verbunden. Als Einschränkung wird der Filter in Prädikat 2 auf die Tabelle EMP angewandt. Hier sieht man auch, dass die Tabellen Statistiken haben. Wäre dies nicht der Fall, wären in der Anzeige zusätzlich folgende Zeilen erschienen:

```
Note
-----
   - dynamic sampling used for this statement
```

Ausschalten lässt sich das Tracing dann einfach mit SET AUTOTRACE OFF.

Um sich den aktuellen Plan aus der SGA anzeigen zu lassen, verwenden Sie V$SQL_PLAN oder besser noch DBMS_XPLAN. Wie bereits im zweiten Kapitel erwähnt, erhalten Sie mit dieser Abfrage den Ausführungsplan für die vorherige SQL-Anweisung:

```
select * from table(dbms_xplan.display_cursor(null,null, 'ALL'));
```

Den Ausführungsplan für eine beliebige SQL-Anweisung bekommen Sie, wenn Sie in diser Abfrage für die beiden ersten Parameter SQL_ID und die Nummer des Childs eingeben. Die entsprechenden Werte für diese Parameter finden Sie wiederum in V$SQL (SQL_ID und CHILD_NUMBER) sowie in V$SESSION (SQL_ID und PREV_SQL_ID, SQL_CHILD_NUMBER, und PREV_CHILD_NUMBER).

5.4 SQL_TRACE

SQL_TRACE gehört sicher zu den bekanntesten (und beliebtesten) Tuning-Utilities. Es lässt sich auf verschiedene Weise aktivieren. Die erste Variante besteht darin, SQL_TRACE = TRUE in die init.ora bzw. das spfile einzufügen und die Datenbank neu zu starten. Das ist aber nur in Spezialfällen zu empfehlen, weil dann alles getraced wird, was enorm viel Diskplatz benötigen kann. Sie können damit Tonnen von Trace-Dateien erzeugen, die meist im Verzeichnis abgelegt werden, was mit dem init.ora-Parameter USER_DUMP_DEST spezifiziert wird,. Ich sagte „meist", weil dies nur für User Traces gilt.

Traces, die von Hintergrundprozessen erzeugt werden (z.B. Parallel Query Slaves), werden im Verzeichnis, das mit BACKGROUND_DUMP_DEST bestimmt wird, abgelegt. Zumeist ist man aber an User Traces interessiert. Die Größe der Trace-Dateien können Sie über den Parameter MAX_DUMP_FILE_SIZE begrenzen, allerdings natürlich mit dem Risiko, dass nicht alles im Trace ist, was gewünscht wurde. Wobei man sich auch fragen kann, wer eine 1000 MB Trace-Datei lesen und auswerten soll. Die meisten Administratoren begrenzen die Größe. Ich empfehle das für den täglichen Normalbetrieb auch. Abgesehen davon kann MAX_DUMP_FILE_SIZE auch über ALTER SESSION verändert werden. Die erzeugten Trace-Dateien haben dann sehr aussagekräftige Namen, wie z.B. V92_ora_123456.trc. Sieht auf den ersten Blick ein wenig kryptisch aus, folgt aber einem bestimmten Format. Konkret ist das der Name der Datenbank (V$THREAD.INSTANCE) plus der Zeichenkette _ora_ plus der Prozess-ID (V$PROCESS.SPID). Seit Oracle 9*i* können Sie über den Parameter TRACEFILE_IDENTIFIER dann noch etwas aussagekräftigere Dateinamen für die erzeugten Trace-Dateien angeben. Dieser Parameter lässt sich auch für die eigene Session über ALTER SESSION setzen.

Der Parameter SQL_TRACE kann sogar über ALTER SYSTEM (ab 9*i*, wenn Sie mit einem spfile arbeiten) verändert werden. Zur Aktivierung von SQL_TRACE in der eigenen Session verwenden Sie wieder ALTER SESSION. Als Datenbankadministrator können Sie SQL_TRACE auch in anderen Sessions setzen; dafür existieren verschiedene Prozeduren, die teilweise erst in bestimmten Versionen verfügbar sind. Eine Variante, die bereits seit Version 7.3 existiert, ist die Prozedur SET_SQL_TRACE_IN_SESSION aus dem Package DBMS_SYSTEM. Bei der Verwendung dieser Prozedur müssen Sie zuerst aber die SID und die Serial# aus V$SESSION für die jeweilige Session ermitteln. Hier ein Beispiel:

```
SQL> select sid,serial# from v$session where username='SCOTT';

 SID SERIAL#
---------- ----------
 9 7

SQL> exec dbms_system.set_sql_trace_in_session(9,7,TRUE);
```

Bitte beachten Sie, dass die Prozedur per Default nur von SYS ausgeführt werden kann. Wollen Sie hier das Tracing wieder ausschalten, geben Sie einfach FALSE für den letzten Parameter an.

Werfen wir mal einen Blick auf die erzeugte Trace-Datei. In meiner SQL*Plus Session habe ich nur das Tracing aktiviert und dann SELECT * FROM DEPT ausgeführt. Im Trace sieht das so aus (die Trace-Datei wurde noch unter Version 10.2 erstellt):

```
Dump file c:\oracle\db\admin\ftest\udump\ftest_ora_5952.trc
Thu Jun 22 12:14:00 2006
ORACLE V10.2.0.1.0 - Production vsnsta=0
vsnsql=14 vsnxtr=3
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, Oracle Label Security, OLAP and Data Mining options
Windows XP Version V5.1 Service Pack 2
CPU                 : 1 - type 586
Process Affinity    : 0x00000000
Memory (Avail/Total): Ph:865M/2039M, Ph+PgF:2856M/3934M, VA:1573M/2047M
Instance name: ftest
```

```
Redo thread mounted by this instance: 1

Oracle process number: 19

Windows thread id: 5952, image: ORACLE.EXE (SHAD)

*** 2006-06-22 12:14:00.203
*** ACTION NAME:() 2006-06-22 12:14:00.193
*** MODULE NAME:(SQL*Plus) 2006-06-22 12:14:00.193
*** SERVICE NAME:(FTEST.ch.oracle.com) 2006-06-22 12:14:00.193
*** SESSION ID:(144.379) 2006-06-22 12:14:00.193
GetTraceVerbLevel: comp_id=1 event-level=0 user_cbm=0 user_vlevl=0 ret_vlevl=0
*** 2006-06-22 12:56:30.721
GetTraceVerbLevel: comp_id=1 event-level=0 user_cbm=0 user_vlevl=0 ret_vlevl=0
*** 2006-06-22 13:11:48.460
GetTraceVerbLevel: comp_id=1 event-level=0 user_cbm=0 user_vlevl=0 ret_vlevl=0
*** 2006-06-22 13:30:26.688
*** SERVICE NAME:(FTEST.ch.oracle.com) 2006-06-22 13:30:26.688
*** SESSION ID:(144.379) 2006-06-22 13:30:26.688
=====================
PARSING IN CURSOR #31 len=32 dep=0 uid=59 oct=42 lid=59 tim=11454181082 hv=1569151342
ad='1d6e700c'
alter session set sql_trace=true
END OF STMT
EXEC #31:c=10015,e=874,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=11454181073
=====================
PARSING IN CURSOR #6 len=52 dep=0 uid=59 oct=47 lid=59 tim=11454198548 hv=1029988163
ad='1d7a30cc'
BEGIN DBMS_OUTPUT.GET_LINES(:LINES, :NUMLINES); END;
END OF STMT
PARSE #6:c=0,e=222,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=11454198539
EXEC #6:c=10015,e=839,p=0,cr=3,cu=0,mis=0,r=1,dep=0,og=1,tim=11454210727
*** SESSION ID:(144.379) 2006-06-22 13:30:30.464
=====================
PARSING IN CURSOR #46 len=18 dep=0 uid=59 oct=3 lid=59 tim=11457960201 hv=3599690174
ad='1d6e7380'
select * from dept
END OF STMT
PARSE #46:c=0,e=39287,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=11457960190
EXEC #46:c=0,e=79,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=11457974634
FETCH #46:c=0,e=131,p=0,cr=5,cu=0,mis=0,r=1,dep=0,og=1,tim=11457977455
FETCH #46:c=0,e=64,p=0,cr=3,cu=0,mis=0,r=3,dep=0,og=1,tim=11457980642
*** SESSION ID:(144.379) 2006-06-22 13:30:30.514
...
```

In Version 11 sieht es sehr ähnlich aus. Wie man sieht, ist das noch nicht allzu aufschlussreich. Manche Informationen sind aber lediglich in diesen Dateien, nicht in den mit TKPROF formatierten. Das gilt vor allem für 10046 Traces, die mit Level 8 oder 12 erstellt wurden. So sind zum Beispiel die Werte von Bind-Variablen nur in diesem Trace zu sehen. Generell ist der Aufbau so, dass Sie für jede SQL-Anweisung, die ihrerseits zuerst unter der Sektion PARSING IN CURSOR protokolliert wird, die verschiedenen Phasen in der Abarbeitung des Cursor, hier also PARSE, EXEC und FETCH, sehen. Falls Sie auch Waits und Bind-Variable tracen, gibt es noch WAIT und BINDS zu sehen. Der Cursor wird immer über eine Nummer mit vorangestelltem Gartenhaken identifiziert. Die verschiedenen Bezeichner in der Zeile (beispielsweise c=, e= und r= sind die Abkürzungen für CPU-Zeit, Elapsed-Zeit, Anzahl der Rows) entsprechen dann den jeweiligen Ausführungsstatistiken. In den meisten Fällen werden Sie aber mit der TKPROF-formatierten Version der Trace-Datei arbeiten. Diese ist viel besser lesbar.

Die beste Beschreibung, wie diese Daten auszuwerten sind, finden Sie übrigens nach wie vor in [Millsap 2003], auch wenn dort die aktuelle Oracle-Version noch 9.2 ist.

5.5 TKPROF

TKPROF dient zum Formatieren von Trace-Dateien. Mit TKPROF allein können Sie aber nichts anfangen. Das Utility setzt zwingend voraus, dass es bereits eine Trace-Datei gibt. Diese wird dann mit TKPROF formatiert und in eine neue Datei geschrieben, optional können Sie sich den Execution-Plan mit anzeigen lassen. Wenn Sie TKPROF ohne Argumente aufrufen, werden Ihnen Informationen zur Verwendungsweise angezeigt:

```
Usage: tkprof tracefile outputfile [explain= ] [table= ]
 [print= ] [insert= ] [sys= ] [sort= ]
 table=schema.tablename Use 'schema.tablename' with 'explain=' option.
 explain=user/password Connect to ORACLE and issue EXPLAIN PLAN.
 print=integer List only the first 'integer' SQL statements.
 aggregate=yes|no
 insert=filename List SQL statements and data inside INSERT statements.
 sys=no TKPROF does not list SQL statements run as user SYS.
 record=filename Record non-recursive statements found in the trace file.
 waits=yes|no Record summary for any wait events found in the trace file.
 sort=option Set of zero or more of the following sort options:
 prscnt number of times parse was called
 prscpu cpu time parsing
 prsela elapsed time parsing
 prsdsk number of disk reads during parse
 prsqry number of buffers for consistent read during parse
 prscu number of buffers for current read during parse
 prsmis number of misses in library cache during parse
 execnt number of execute was called
 execpu cpu time spent executing
 exeela elapsed time executing
 exedsk number of disk reads during execute
 exeqry number of buffers for consistent read during execute
 execu number of buffers for current read during execute
 exerow number of rows processed during execute
 exemis number of library cache misses during execute
 fchcnt number of times fetch was called
 fchcpu cpu time spent fetching
 fchela elapsed time fetching
 fchdsk number of disk reads during fetch
 fchqry number of buffers for consistent read during fetch
 fchcu number of buffers for current read during fetch
 fchrow number of rows fetched
 userid userid of user that parsed the cursor
```

Das sieht jetzt seit Version 9.2 so aus. Die Option „waits“ gab es in 8.1.7 noch nicht, wir werden sie uns beim 10046 Event näher ansehen. Ich lasse TKPROF meist mit sys=no und der Option Explain laufen. Mit sys=no sehen Sie die internen, von Oracle selbst generierten Statements (Abfragen auf das Data Dictionary etc.) nicht mehr im Output. Da es im Regelfall aber ohnehin nicht möglich ist, diese Statements zu tunen, können Sie normalerweise auch nichts mit ihnen anfangen. Die Option table würden Sie benötigen, wenn Ihre Plan Table nicht PLAN_TABLE heißen würde. Ein Grund mehr also, die Plan Table nicht umzubenennen. Wenn Sie erst einmal nur am SQL der Applikation interessiert sind, können Sie die Record-Klausel verwenden. Damit wird das ganze SQL der Trace-Datei in der Datei abgelegt, die Sie unter Record spezifizieren. Allerdings auch die internen Statements. Das kann dann so aussehen:

```
SELECT PT.VALUE FROM SYS.V_$SESSTAT PT WHERE PT.SID=:1 AND PT.STATISTIC#
IN (7,40,41,42,115,236,237,238,242,243) ORDER BY PT.STATISTIC# ;
```

```
select distinct ename from big_emp where hiredate=to_date('17.12.80','DD.MM.YY') ;

SELECT PT.VALUE FROM SYS.V_$SESSTAT PT WHERE PT.SID=:1 AND PT.STATISTIC# IN
(7,40,41,42,115,236,237,238,242,243) ORDER BY PT.STATISTIC# ;


DELETE FROM PLAN_TABLE WHERE STATEMENT_ID=:1 ;

EXPLAIN PLAN SET STATEMENT_ID='PLUS645' FOR select distinct ename from big_emp
where hiredate=to_date('17.12.80','DD.MM.YY') ;
```

In der Praxis sind die verschiedenen Sort-Optionen noch ganz interessant. Falls CPU oder RAM Mangelware sind, dürften CPU und Elapsed-Zeit für Sie interessant sein (execpu und exeela), ansonsten noch die Consistent Reads (fchqry). Werfen wir mal einen Blick in solch eine formatierte Datei.

```
TKPROF: Release 11.1.0.7.0 - Production on Mon Jun 22 16:52:27 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


Trace file: v11107_ora_20517_10046nok.trc
Sort options: default

********************************************************************************
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
```

Am Anfang berichtet uns der TKPROF also netterweise immer, mit welchen Sort-Optionen er aufgerufen wurde und für welche Trace-Datei. Es folgt eine kurze Erläuterung der Spalten, danach die eigentlichen Anweisungen. In diesem Fall hatte ich nur ein SELECT * FROM DEPT abgesetzt und die Option EXPLAIN verwendet. Weiter unten in der TKPROF-Datei finde ich dann auch mein Statement.

```
select * from dept
```

Danach wird es richtig interessant. Jetzt kommt die Zerlegung in die drei Phasen Parse, Execute und Fetch. Hier im Beispiel gibt es keine CPU und Elapsed-Zeit, was bei dieser mickrigen Abfrage auch ein Wunder gewesen wäre. Falls Sie jedoch Zeiten erwarten, in TKPROF aber nichts zu sehen ist, überprüfen Sie den Parameter TIMED_STATISTICS. Der muss auf TRUE gesetzt sein (man kann ihn auch über ALTER SESSION verändern). Hier im Beispiel sehen wir auch, dass ich für die vier Rows vier Zugriffe auf Disk benötigte. Die Daten waren somit noch nicht im Buffer Cache.

```
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   2      0.00     0.00       0          4          0          4
------- ------ -------- ---------- ---------- ---------- ---------- ---
total   4      0.00     0.00       0          4          0          4
```

In „normalen“ OLTP-Applikationen sollte es relativ wenige Parses und sehr viele Executes geben. Falls dem nicht so ist, wissen Sie, dass keine oder kaum Bind-Variablen verwendet

werden und/oder viel dynamisches SQL ausgeführt wird. Das sind beides Bereiche, in denen sich viel optimieren lässt.

Beim Fetch ist noch interessant, wie oft es ausgeführt wurde und wie viele Rows insgesamt zurückkamen. Wünschenswert ist hier, dass möglichst viele Rows in möglichst wenig Fetches ausgegeben werden können. Optimieren lässt sich dies durch den Einsatz von Array Fetches. Array Fetches können Sie im SQL*Plus und im Precompiler mit der ARRAYSIZE Option konfigurieren. Dort teilen Sie Oracle mit, wie viel Rows auf einmal von der DB zurückgegeben werden sollen. Werte größer 100 sind hier allerdings selten sinnvoll. Wenn der Verkehr über SQL*Net-TCP/IP erfolgt, kann in der TNSNAMES.ORA noch der Parameter SDU gesetzt werden. Dort spezifizieren Sie, wie groß der Buffer für die Übertragung sein soll; 8192 Byte ist hier ein ganz guter Wert.

Danach sehen wir, ob das Statement selbst bereits im Cache war und welcher Optimizer verwendet wird, sowie unter welchem Benutzer es lief. Wir sehen auch den verwendeten Optimizer.

```
Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 59 (SCOTT)
```

Danach kommen Row Source Operation und Execution Plan. Die beiden sind oft identisch, müssen es aber nicht sein. Sie sehen sie auch nur, wenn Sie die Option EXPLAIN im TKPROF verwendet haben. In der Spalte „Row Source Operation“ sehen Sie den zur Laufzeit benutzten Ausführungsplan. In der Spalte „Execution Plan“ sehen Sie den Plan, wie er erzeugt wurde, als Sie TKPROF ausführten. Falls Sie also mal Traces generieren und diese erst Wochen später mit TKPROF formatieren, stehen die Chancen gut, dass Sie hier Unterschiede feststellen.

```
Rows     Row Source Operation
-------  ---------------------------------------------------
      4  TABLE ACCESS FULL DEPT (cr=8 pr=0 pw=0 time=126 us)


Rows    Execution Plan
------- ---------------------------------------------------
 0      SELECT STATEMENT MODE: ALL_ROWS
 4      TABLE ACCESS    MODE: ANALYZED (FULL) OF 'DEPT' (TABLE)
```

Das ist eine Darstellung mit TKPROF aus der Version 10.2 respektive Version 11, in der Sie in der Row Source Operation auch statistische Kennzahlen sehen. Beachten Sie bitte auch, dass in der Spalte „Rows“ die Rows-Anzahl in jedem Schritt aufgelistet wird. Bei einigen Plänen ist das sehr hilfreich, um zu beurteilen, ob der Plan gut oder schlecht ist. Ein Beispiel dafür sind Nested Loop Joins, dort sind die einzelnen Spalten zu multiplizieren. Sehen wir uns einen entsprechenden Ausschnitt an:

```
Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   MODE: ALL_ROWS
      6   NESTED LOOPS
      6    TABLE ACCESS   MODE: ANALYZED (FULL) OF 'EMP' (TABLE)
```

Hier sind es nur sechs Rows, und seit Oracle 10.2 wird die Anzahl bereits bei NESTED LOOPS direkt ausgewiesen; in frühren Versionen mussten Sie das noch selbst zusammenrechnen, also Anzahl Nested Loops mal Anzahl Rows im darunter liegenden Schritt. Andere Kandidaten, bei denen die Anzahl Rows schnell ins Gigantische anwachsen können, sind kartesische Produkte und Outer Joins.

Sie erhalten den Ausführungsplan hier auch nur, wenn im unformatierten Trace die entsprechenden STAT-Zeilen vorhanden sind. Diese STAT-Zeilen werden in die Trace-Datei geschrieben, wenn der Cursor geschlossen wird. Bitte beachten Sie, dass dies nicht der Fall ist, wenn Sie einfach SQL_TRACE ausschalten. Um sicherzugehen, dass der Cursor geschlossen wird, können Sie DBMS_SESSION.RESET_PACKAGE verwenden oder einfach eine Dummy-Abfrage nach der eigentlichen Anweisung starten, damit der vorherige Cursor geschlossen wird, und dann die SQL*PLus Session sauber verlassen. Das könnte dann so aussehen:

```
...
... Hier läuft die SQL-Anweisung, die Sie tracen ...

select * from dual;   -- die Dummy-Abfrage
Exit;                   -- die Session wird sauber beendet
```

Seit Oracle 9*i* werden standardmäßig auch Waits ausgegeben, falls ein 10046 Trace mit Level 8 aktiviert wurde (dazu mehr in Abschnitt 5.6). Diese Waits sehen Sie dann auch, wenn Sie das File mit TKPROF formatiert haben. Hier ein Beispiel:

```
select e.ename,d.dname
from
 emp e,dept d
...
call count cpu elapsed disk query current rows
....

Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   MODE: ALL_ROWS
     64   MERGE JOIN (CARTESIAN)
      4    TABLE ACCESS   MODE: ANALYZED (FULL) OF 'DEPT' (TABLE)
     64    BUFFER (SORT)
     16     TABLE ACCESS   MODE: ANALYZED (FULL) OF 'EMP' (TABLE)

Elapsed times include waiting on following events:
 Event waited on              Times Max.Wait Total Waited
 ---------------------------- ---- -------- ------------
 SQL*Net message to client     5    0.00     0.00
 db file sequential read       2    0.00     0.00
 SQL*Net message from client   5    1.99     2.21
```

Allerdings sind die Statistiken pro Cursor verdichtet. Sie sehen also nicht die einzelnen Waits. Dazu müssen Sie in der unformatierten Trace-Datei nachschauen. Hier sehen Sie etwa:

```
...
PARSE #1:c=0,e=172,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1556036945363
WAIT #1: nam='db file sequential read' ela= 6802 p1=8 p2=779 p3=1
WAIT #1: nam='db file sequential read' ela= 6855 p1=8 p2=778 p3=1
WAIT #1: nam='db file sequential read' ela= 383 p1=8 p2=777 p3=1
```

Die Parameter bei den Wait Events sind je nach Event unterschiedlich zu interpretieren. Näheres dazu in der Oracle Reference, wo Sie einen kompletten Anhang ausschließlich zu den diversen Wait Events finden. Hier im Beispiel ist Parameter p1 die relative Dateinummer, p2 die Blocknummer und p3 die Anzahl der Blöcke. Das ist generell der Fall bei folgenden Operationen:

- db file sequential read
- db file scattered read
- db file single write
- buffer busy waits
- free buffer waits

Gerade beim I/O Tuning kann das sehr interessant sein. Im obigen Beispiel haben wir also Waits beim Zugriff auf Dateinummer 8 und die Blöcke 777 bis 779. Es wird immer nur ein Block gelesen (p3=1). Letzteres ist wichtig, wenn man den Parameter DB_FILE_MULTI-BLOCK_READ_COUNT testet, da will man ja höhere Werte sehen. Verwirrenderweise gibt es bei Full Table Scans „db file scattered read" zu sehen.

Praktischerweise lässt sich aus relativer Dateinummer und Blocknummer das zugehörige Objekt aus DBA_EXTENTS ermitteln. Die entsprechende Query sieht auf den ersten Blick ein wenig sonderbar aus und wurde bereits im dritten Kapitel vorgestellt. Um den Lesefluss nicht zu unterbrechen, hier noch einmal die Wiederholung. Vollziehen wir das einmal beispielhaft für File 8 und Block 777 nach:

```
SQL> select owner, segment_name, segment_type
 2 from dba_extents
 3 where file_id = 8
 4 and 777 between block_id and (block_id + blocks) -1;

SYS AUD$ TABLE
```

Die Bedingung `FILE_ID = 8` ist ja noch verständlich, aber wozu `777 BETWEEN BLOCK_ID AND (BLOCK_ID + BLOCKS -1)`? Erinnern wir uns: Jedes Oracle Extent besteht aus mehreren Oracle-Blöcken. Deshalb kann es gut sein, dass Block 777 irgendwo im File ist, aber auf alle Fälle innerhalb der verschiedenen Block ID Ranges. Das Segment an sich hat noch einen Header-Block, deshalb wird –1 abgezogen. Wir hätten es auch anders machen können. Wir lassen uns die verschiedenen BLOCK_ID aus der DBA_EXTENTS sortiert ausgeben und schauen dann, wo 777 reinfällt. Etwas anderes macht die Query auch nicht.

Wie man hier sieht, verwende ich Auditing (sonst hätte ich nicht auf die Tabelle AUD$ zugegriffen) und habe offensichtlich einige Probleme damit. Die Lösung wäre hier das Auslagern der Tabelle AUD$ aus dem SYSTEM-Tablespace. Anschließend müsste ich genauer untersuchen, was ich da alles auditiere.

Ganz zum Schluss bringt der TKPROF noch eine Zusammenfassung. Das kann dann so aussehen:

```
...
OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count  cpu      elapsed    disk       query      current    rows
------- ------ -------- ---------- ---------- ---------- ---------- ---
Parse    131    3.56     3.86       0          3          1          0
Execute 131    70.62    278.32     91292      49685      1441       40
Fetch    258    75.34    516.75     176293     3997535    0          2382
------- ------ -------- ---------- ---------- ---------- ---------- ---
total    520   149.53    798.93     267585     4047223    1442      2422

Misses in library cache during parse: 34
Misses in library cache during execute: 1


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count  cpu      elapsed    disk       query      current    rows
------- ------ -------- ---------- ---------- ---------- ---------- ---
Parse    253    3.02     3.14       0          13         0          0
Execute 345    0.38     0.71       5          59         65         47
Fetch    880    40.88    674.63     33135      3848824    0          640
------- ------ -------- ---------- ---------- ---------- ---------- ---
total    1478   44.29    678.49     33140      3848896    65         687

Misses in library cache during parse: 11
Misses in library cache during execute: 1

 146 user SQL statements in session.
 239 internal SQL statements in session.
 385 SQL statements in session.
 26 statements EXPLAINed in this session.
********************************************************************************
Trace file: v92_ora_1220.trc
Trace file compatibility: 10.01.00
Sort options: default

 52 sessions in tracefile.
 3916 user SQL statements in trace file.
 5891 internal SQL statements in trace file.
 385 SQL statements in trace file.
 110 unique SQL statements in trace file.
 26 SQL statements EXPLAINed using schema:
 SCOTT.prof$plan_table
 Default table was used.
 Table was created.
 Table was dropped.
 4558 lines in trace file.
```

Das ist jetzt noch ein Beispiel aus der Version 10.2, in Version 11 werden zum Schluss zusätzlich die Wartezeiten aufgelistet. Hier wird noch zwischen rekursiven und nicht rekursiven Statements unterschieden. Rekursive Statements sind Statements, die Oracle intern selbst generiert. Wenn Sie beispielsweise das erste Mal die Abfrage SELECT * FROM DEPT ausführen, setzt Oracle zunächst einige Abfragen auf das Data Dictionary ab. Dort wird dann geprüft, ob es das Objekt überhaupt gibt, wie es mit den Berechtigungen aussieht, etc. Neben dem Parsing werden rekursive Anweisungen auch beim Einsatz von PL/SQL erzeugt. Dies bedeutet: Aus dem Verhältnis Rekursiv zu Nicht Rekursiv lässt sich ohne weitere Infos erst einmal nicht ableiten, ob es sich um „gutes“ oder „schlechtes“ SQL handelt.

5.6 Event 10046

Event 10046 ist für das Tuning das Event mit den besten Informationen. Interessant ist das Level. Level 1 ist gleichbedeutend mit SQL_TRACE = TRUE. Intern macht Oracle übrigens auch nichts anderes, wenn Sie SQL_TRACE auf TRUE setzen, es wird dann auch Event 10046 eingestellt. Insgesamt gibt es vier verwendbare Levels beim 10046 Trace:

- 1 entspricht SQL_TRACE
- 4 Ausgabe der Werte von Bind-Variablen
- 8 Ausgabe der Waits
- 12 Ausgabe von Bind-Variablen und Waits

Seien Sie vorsichtig, wenn Sie die höheren Levels verwenden, das kann die Trace-Dateien ungeahnt aufblasen. Nehmen wir mal an, dass in der Applikation die SQL-Anweisung SELECT * FROM EMP WHERE EMPNO = :b1 vorkommt. Dies bedeutet: In diesem Statement wird eine Bind-Variable verwendet. Angenommen, dieses Statement wird 10000 mal mit 10000 verschiedenen Werten ausgeführt, und Sie fahren einen Level 4 Trace, dann haben Sie auch 10000 Werte für die Bind-Variable in der Trace-Datei.

Andererseits haben Sie nur in diesen Traces die unter Umständen ausschlaggebenden Informationen. Wenn also das einfache Tracing mit Level 1 noch nicht aufschlussreich genug ist, kommen Sie um Level 4, 8 oder 12 nicht herum.

Wie schon erwähnt, sehen Sie Bind-Werte nicht in Trace-Dateien, die von TKPROF formatiert wurden, sondern nur in den unformatierten Traces. In der eigenen Session wird Event 10046 über ALTER SESSION aktiviert, hier ein kleines Beispiel:

```
SQL>alter session set events '10046 trace name context forever, level 4';

Session wurde geändert.

SQL> declare
 2 x number;
 3 val number;
 4 cursor c1 is select empno from emp where empno >x;
 5 begin
 6 x:=2500;
 7 for c1_rec in c1 loop
 8 val := c1_rec.empno;
 9 end loop;
    10 end;
    11 /

PL/SQL-Prozedur wurde erfolgreich abgeschlossen.
```

Blödsinniger Code, aber es geht ja nur darum, das zu veranschaulichen. Hier ist x unsere Bind-Variable. Im Trace sehen wir dann aber :b1 statt x (die Bind-Variablen werden einfach in jedem Statement hochgezählt):

```
PARSING IN CURSOR #4 len=38 dep=1 uid=59 oct=3 lid=59 tim=1569491954784
hv=1442032374 ad='6ebca594'
SELECT empno from emp where empno >:b1
END OF STMT
PARSE #4:c=0,e=1010,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=0,tim=1569491954766
BINDS #4:
```

```
bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=13 oacfl2=1 size=24 offset=0
bfp=0841fdd0 bln=22 avl=02 flg=05
value=2500
```

In der Spalte value= erhalten Sie dann den aktuellen Wert. Dty ist der Datentyp. 2 entspricht Number. Welche Zahl welchem Datentyp entspricht, können Sie dem Oracle Call Interface (OCI) Manual entnehmen ([OraOci 2008]). Dieser Trace kann übrigens auch ganz nützlich sein, wenn Sie mal ORA-1722 (invalid number) in Ihrer Applikation erhalten. Bei diesem Fehler sagt Oracle zwar schon, dass sich eine ungültige Zahl in einer numerischen Spalte befindet, aber leider nicht, um welche es sich handelt.

Level 8 ist, wie schon oben im TKPROF-Abschnitt erwähnt, der Trace auf Wait Events. Sehr nützlich bei Verdacht auf I/O-Probleme. Wenn Sie dort übrigens ungültige Dateinummern sehen, handelt es sich um (echte) Temporärdateien.

Level 12 – last but not least – vereint Level 4 und 8. Wie gesagt, dies sind alles sehr umfangreiche Traces, deshalb sollten sie nur beim detaillierten Tracing verwendet werden.

5.7 Ausführungspläne in der Vergangenheit

Mit EXPLAIN PLAN und SQL_TRACE können Sie sich zwar aktuelle Ausführungspläne anzeigen lassen, doch was machen Sie, wenn Sie wissen wollen, wie der Plan in der Vergangenheit aussah? Das kommt häufiger vor, als man gemeinhin annimmt. Angenommen, Sie kommen am Montag ins Büro, und Nutzer erzählen Ihnen, dass es am Samstag eine Störung gab; das System war extrem langsam. Aber nun sei alles wieder gut, und unternommen wurde nichts. Jetzt erklären Sie mal bitte schön, was da los war. Natürlich werden Sie zunächst untersuchen, ob es irgendwelche Vorkommnisse gab, die das erklären könnten, wenn Sie aber nichts finden, liegt in diesem Szenario die Vermutung nahe, dass sich aus irgendwelchen Gründen vielleicht der Ausführungsplan geändert hatte. Vielleicht aufgrund neuer Statistiken für die Applikation? Vielleicht wissen Sie ja aus der Vergangenheit, um welche SQL-Anweisung in der Applikation es sich handeln könnte. Ist dies der Fall, könnten Sie mit DBMS_XPLAN.DISPLAY_CURSOR (ab Version 10.2) oder V$-SQL_PLAN (vor Version 10.2) mal prüfen, ob Sie das entsprechende Child noch finden. Das geht aber nur, wenn die Datenbank zwischenzeitlich nicht neu gestartet wurde.

Viel wahrscheinlicher ist aber, dass Sie zunächst keine Ahnung haben. Sie werden also AWR oder Statspack (dazu später mehr) für die entsprechenden Zeitperiode laufen lassen, um einen Eindruck zu gewinnen, was dort schiefgelaufen ist. AWR und Statspack werden Ihnen auch zeigen, welche Ressourcen benötigt wurden. Was Sie dort aber nicht sehen, sind die Ausführungspläne. Um sich die anzeigen zu lassen, brauchen Sie das Script „Workload Repository SQL Report“ (im AWR), über das Script awrsqrpt.sql erstellt, bzw. den mit dem Script sprepsql.sql erstellten „Statspack SQL Report“. Beide Scripts sind in $ORACLE_HOME/rdbms/admin zu finden.

Im AWR-Bericht wird in den Abschnitten, die SQL-Anweisungen zeigen, immer die SQL_ID bei jedem SQL angegeben. Statspack benutzt hingegen den Hashwert der SQL-Anweisung und zeigt immer diesen an. Wenn Sie dann den Bericht laufen lassen, müssen

Sie zusätzlich zum Intervall die SQL_ID respektive den Hashwert angeben. Im Bericht sehen Sie dann Ausführungsstatistiken und Ausführungspläne. Hier ein Ausschnitt aus dem AWR SQL-Bericht:

```
Plan Statistics                          DB/Inst: V11107/v11107  Snaps: 6036-6040
-> % Total DB Time is the Elapsed Time of the SQL statement divided
   into the Total Database Time multiplied by 100

Stat Name                                   Statement   Per Execution % Snap
---------------------------------------- ---------- -------------- ------
Elapsed Time (ms)                            22,653        1,132.6    28.2
CPU Time (ms)                                 5,878          293.9    12.4
Executions                                       20            N/A     N/A
Buffer Gets                                 102,855        5,142.8    25.0
Disk Reads                                    1,469           73.5     3.6
Parse Calls                                       3            0.2     0.0
Rows                                      1,159,203       57,960.2     N/A
User I/O Wait Time (ms)                         234            N/A     N/A
Cluster Wait Time (ms)                            0            N/A     N/A
Application Wait Time (ms)                        0            N/A     N/A
Concurrency Wait Time (ms)                        0            N/A     N/A
Invalidations                                     1            N/A     N/A
Version Count                                     2            N/A     N/A
Sharable Mem(KB)                                 25            N/A     N/A
--------------------------------------------------------------------------------

Execution Plan
--------------------------------------------------------------------------------
| Id  | Operation        | Name  | Rows | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | INSERT STATEMENT |       |      |       |     3 (100)|          |
|   1 |  LOAD TABLE CONVENTIONAL |      |      |       |            |          |
|   2 |   TABLE ACCESS FULL      | EMP2 |   14 |  1218 |3   (0)| 00:00:01 |
```

Falls sich wirklich Änderungen im Ausführungsplan ergeben haben, muss dies natürlich näher untersucht werden (das wurde in Kapitel 2 bereits genauer beschrieben).

5.8 DBMS_MONITOR

Seit Oracle 10*g* können Sie auch das DBMS_MONITOR-Package zum Aktivieren des Tracing verwenden. Für einen bestimmten Client, den Sie zuerst mit der Prozedur DBMS_SESSION.SET_IDENTIFIER identifizieren müssen, erfolgt dies über DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE. Für Service, Modul und/oder Aktion erfolgt dies über DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE. Sie können dort auch spezifizieren, ob Sie Bind-Werte und Wait Events mit protokollieren wollen. Das ermöglicht Ihnen speziell in 3-Tier-Umgebungen das End-to-End-Tracing.

Beachten Sie bitte, dass diese Formen des Tracing auch nach einem Neustart der Datenbank noch gelten. Sie müssen also das Tracing ausdrücklich über DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE beziehungsweise über DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE wieder ausschalten. Tracing für eine bestimmte Session können Sie auch ein- und ausschalten (über SESSION_TRACE_ENABLE und SESSION_TRACE_DISABLE).

5.9 Event 10053

Event 10053 werden Sie wahrscheinlich selten brauchen. Damit können Sie die Entscheidungen des Costbased Optimizer verfolgen. Wir besprechen dieses Event zuerst generell, bevor wir zum Schluss noch die Auswertung anhand einer Beispielabfrage zeigen.

Interessant ist dieses Event, wenn Sie die Vermutung hegen, dass der Optimizer aufgrund der verfügbaren Statistiken falsch entschieden hat. Das Event ist nur sinnvoll, wenn Sie den Costbased Optimizer verwenden. Wenn Sie es mit dem RULE-based Optimizer versuchen, bekommen Sie keinen Output. Das Event wird immer auf Level 1 gesetzt:

```
ALTER SESSION SET EVENTS '10053 trace name context forever, level 1';
```

Danach muss EXPLAIN PLAN für die SQL-Anweisung ausgeführt werden. Die Query muss zwingend neu geparsed werden, sonst kommt dabei nichts heraus. Wenn Sie also beispielsweise vorher ein

```
EXPLAIN PLAN FOR SELECT * FROM EMP;
```

durchgeführt haben, müssen Sie dieses Statement verändern, so dass ein neuer Hard Parse erfolgt. (Sie erinnern sich: SQL kann nur gemeinsam genutzt werden, wenn der Text der SQL-Anweisung inklusive Groß-/Kleinschreibung und Leerzeichen gleich ist. Es reicht also vollkommen aus, ein Leerzeichen einzufügen oder einen vorher großgeschriebenen Buchstaben kleinzuschreiben.)

```
EXPLAIN PLAN FOR SELECT * FROM Emp;
```

Danach wird eine Trace-Datei in USER_DUMP_DEST geschrieben. Wenn Sie diese öffnen, sehen Sie unter dem Stichwort QUERY Ihre EXPLAIN PLAN-Anweisung:

```
QUERY
explain plan for select e.ename,d.loc from dept d, emp e
where e.deptno=d.deptno
```

Es folgt die Liste aller Parameter, die für den CBO interessant sind. Netterweise sehen Sie hier auch die undokumentierten (an den undokumentierten sollten Sie aber, wie gesagt, nur herumschrauben, wenn Sie wirklich wissen, was Sie gerade tun).

```
***************************************
PARAMETERS USED BY THE OPTIMIZER
********************************
optimizer_features_enable           = 10.2.0.1
_optimizer_search_limit             = 5
cpu_count                           = 1
...
```

Danach kommen die Grundstatistiken für die beteiligten Tabellen, Indizes und Spalten (falls Sie Histogramme verwenden). Viele dieser Statistiken sind aber nur für spezifische Probleme von Interesse. Beachten Sie bitte, dass, wie bereits im zweiten Kapitel näher beschrieben, diese Infos auch im Data Dictionary zu finden sind. Anbei ein Ausschnitt:

```
Table Stats::
  Table: EMP  Alias:  E
    #Rows: 14  #Blks:  5  AvgRowLen:  37.00
```

```
  Column (#8): DEPTNO(NUMBER)
    AvgLen: 3.00 NDV: 3 Nulls: 0 Density: 0.33333 Min: 10 Max: 30
Index Stats::
  Index: PK_EMP  Col#: 1
    LVLS: 0  #LB: 1  #DK: 14  LB/K: 1.00  DB/K: 1.00  CLUF: 1.00
```

Vor Version 10.2 finden Sie an dieser Stelle CDN für die Kardinalität, i.d. die Anzahl der Datensätze; ab 10.2 steht hier einfach #Rows. Es gibt weitere Versionsunterschiede, beispielsweise die Anzahl der Oracle-Blöcke #Blks in Version 10.2 und NBLKS vorher. Weil wir über deptno joinen, sehen wir auch die Statistiken für die Spalte deptno. NDV ist die Anzahl unterschiedlicher Werte für die Spalte. Einen NULL-Wert gibt's auch einmal. Min (früher LO) und Max (früher HI) sind der kleinste und größte Wert. Histogramme habe ich keines auf der Spalte. Für die Indizes sehen Sie LVLS – die Anzahl der Level. Falls diese größer als 3 oder 4 werden, könnten Sie sich ein ALTER INDEX ... REBUILD überlegen. #LB ist die Anzahl der Leaf-Blöcke, #DK die Anzahl unterschiedlicher Schlüsselwerte des Index. LB/K und DB/K bezeichnen die Anzahl der Leaf-Blöcke per Schlüsselwert und die Anzahl unterschiedlicher Blöcke per Schlüsselwert. CLUF ist der Clustering Factor. Da gilt ja, wie schon an anderer Stelle beschrieben, die Regel: Wenn CLUF <= Anzahl Blöcke der Tabelle ist, ist alles in Ordnung, andernfalls wäre ein Rebuild des Index vielleicht angebracht. Danach wird's richtig interessant: Jetzt kommen die Zugriffe auf die beteiligten Tabellen.

Seit Version 10.2 werden die Abkürzungen netterweise im Trace selbst erklärt. Hier mal ein entsprechender Ausschnitt aus einem 10053-Trace:

```
...
The following abbreviations are used by optimizer trace.
CBQT - cost-based query transformation
JPPD - join predicate push-down
FPD - filter push-down
…
LB - leaf blocks
DK - distinct keys
LB/K - average number of leaf blocks per key
DB/K - average number of data blocks per key
CLUF - clustering factor
NDV - number of distinct values
Resp - response cost
Card – cardinality
Resc - resource cost
NL - nested loops (join)
SM - sort merge (join)
...
```

Danach kommen die Kosten für die Zugriffe auf die einzelnen Tabellen. Falls Sie hier bereits für die Kosten des Zugriffs und die (berechnete) Kardinalität Schrott erhalten, können Sie auch nicht erwarten, dass der endgültige Plan in Ordnung ist.

```
***************************************
SINGLE TABLE ACCESS PATH
  Table: EMP  Alias: E
    Card: Original: 14  Rounded: 14  Computed: 14.00  Non Adjusted: 14.00
  Access Path: TableScan
    Cost:  3.00  Resp: 3.00  Degree: 0
      Cost_io: 3.00  Cost_cpu: 39667
      Resp_io: 3.00  Resp_cpu: 39667
  Best:: AccessPath: TableScan
         Cost: 3.00  Degree: 1  Resp: 3.00  Card: 14.00  Bytes: 0
```

Hier sehen Sie auch sehr schön die Einbeziehung von CPU und I/O in die Berechnung der Kosten. Diese Information war in früheren Versionen nicht ausgewiesen, wie man im folgenden Ausschnitt sieht:

```
***************************************
SINGLE TABLE ACCESS PATH
 TABLE: EMP ORIG CDN: 15 ROUNDED CDN: 14 CMPTD CDN: 14
 Access path: tsc Resc: 2 Resp: 2
 BEST_CST: 2.00 PATH: 2 Degree: 1
```

Danach kommen die eigentlichen Berechnungen für die möglichen Execution-Pläne. Die hier verwendeten Abkürzungen sind NL für Nested Loop, SM für Sort Merge und HA Join für den Hash Join. Typischerweise wird ein Join über eine dieser drei Methoden realisiert, weshalb Sie auch im 10053 Trace nacheinander auftauchen:

```
***************************************
OPTIMIZER STATISTICS AND COMPUTATIONS
***************************************
GENERAL PLANS
***********************
Join order[1]:  DEPT[D]#0  EMP[E]#1
***************
Now joining: EMP[E]#1
***************
NL Join
  Outer table: Card: 4.00  Cost: 3.00  Resp: 3.00  Degree: 1  Bytes: 13
  Inner table: EMP  Alias: E
  Access Path: TableScan
    NL Join:  Cost: 9.01  Resp: 9.01  Degree: 0
      Cost_io: 9.00  Cost_cpu: 194956
      Resp_io: 9.00  Resp_cpu: 194956
  Best NL cost: 9.01
         resc: 9.01 resc_io: 9.00 resc_cpu: 194956
         resp: 9.01 resp_io: 9.00 resp_cpu: 194956
Join Card:  14.00 = outer (4.00) * inner (14.00) * sel (0.25)
Join Card - Rounded: 14 Computed: 14.00
SM Join
  Outer table:
    resc: 3.00  card 4.00  bytes: 13  deg: 1  resp: 3.00
  Inner table: EMP  Alias: E
    resc: 3.00  card: 14.00  bytes: 9  deg: 1  resp: 3.00
    using dmeth: 2  #groups: 1
    SORT resource      Sort statistics
```

Dieser Teil kann extrem lang werden. Es gibt zwar einige Optimierungen, doch sind bis zu n! (also Fakultät von n) Zugriffspfade möglich. Je komplexer die Query, desto umfangreicher können die Zugriffspfade werden. Am Anfang startet der Optimizer damit, dass er die Tabellen nach Kardinalität ordnet und dann die Joins berechnet – außer wenn Sie den ORDERED-Hint verwenden, dann sollte natürlich die Reihenfolge von links nach rechts wie in der FROM-Klausel spezifiziert gültig sein. Jede Join-Berechnung wird durchgespielt und die mit den niedrigsten Kosten behalten. Bitte beachten Sie auch, dass der Optimizer jeweils nur Pläne berücksichtigt, die billiger sind als der jeweils vorhergehende. Ist der aktuelle Plan teurer als sein Vorgänger, sehen Sie ihn nicht. Das geht so durch bis zur Final Section:

```
Final - All Rows Plan:  Best join order: 2
  Cost: 3.0097  Degree: 1  Card: 14.0000  Bytes: 280
  Resc: 3.0097  Resc_io: 3.0000  Resc_cpu: 138153
  Resp: 3.0097  Resp_io: 3.0000  Resc_cpu: 138153
```

Hier sehen Sie nur noch die Kosten des Plans, der schließlich verwendet wurde, aber nicht mehr den Plan selbst. Der ist ja auch in der Trace-Datei. RESC (RSC vor 10.2) sind die Kosten für den seriellen Zugriff, RESP (RSP vor 10.2) die Zugriffskosten bei parallelem Zugriff.

Seit Version 10.2 sehen Sie in diesem Trace übrigens auch die Peek-Werte der Bind-Variablen, was sehr nützlich sein kann.

10053 Trace am Beispiel

Für unser Beispiel nehmen wir die folgende Abfrage:

```
select * from emp where empno=123;
```

Am Ende der 10053 Trace-Datei sehen wir für die Abfrage den folgenden Plan. Der Ausführungsplan ist auch unser Ausgangspunkt, bei der Arbeit mit 10053 Traces gehen wir immer von unten nach oben. Um eine bessere Lesbarkeit zu garantieren, wurde der Plan editiert; die TIME-Spalte fehlt (nicht erforderlich).

```
Plan Table
============

*** 2009-06-25 09:53:23.157
-------------------------------------------+----------------------
| Id  | Operation                    | Name    | Rows | Bytes | Cost  |
-------------------------------------------------------------------
| 0   | SELECT STATEMENT             |         |      |       |     1 |
| 1   |  TABLE ACCESS BY INDEX ROWID | EMP     |    1 |    37 |     1 |
| 2   |   INDEX UNIQUE SCAN          | PK_EMP  |    1 |       |     0 |
```

Wir sehen also, dass er die Abfrage über einen Index Unique Scan ausführt. Er geht davon aus, dass er nur einen Datensatz bekommt, die Kosten für den Plan betragen 1, und 37 Bytes werden für diesen einen Datensatz erwartet. 37 Bytes ist der Wert für AVG_ROW_LEN aus DBA_TABLES. Wenn wir von dieser Stelle weiter nach oben gehen, sehen wir die zugrunde liegende Kostenberechnung für den Plan:

```
Final cost for query block SEL$1 (#0) - All Rows Plan:
  Best join order: 1
  Cost: 1.0009  Degree: 1  Card: 1.0000  Bytes: 37
  Resc: 1.0009  Resc_io: 1.0000  Resc_cpu: 8461
  Resp: 1.0009  Resp_io: 1.0000  Resc_cpu: 8461
```

Die Kosten für den besten Plan betragen also 1.009, und die beste Join Order ist die erste. Ein paar Zeilen zurück finden wir die Join Order. Da wir hier nur auf eine Tabelle zugreifen, gibt es auch nur eine Join Order:

```
Join order[1]:  EMP[EMP]#0
**********************
Best so far:  Table#: 0  cost: 1.0009  card: 1.0000  bytes: 37
```

Wieder ein paar Zeilen weiter oben sehen wir dann auch, dass ein Index Unique Scan für PK_EMPNO in der Sektion SINGLE TABLE ACCESS PATH in der Zeile genommen wurde, die mit „Best:: Access Path:“ beginnt:

```
One row Card: 1.000000
  Best:: AccessPath: IndexUnique
```

```
 Index: PK_EMP
         Cost: 1.00  Degree: 1  Resp: 1.00  Card: 1.00  Bytes: 0
```

Noch in der gleichen Sektion weiter oben sehen wir die eigentliche Berechnung der Kardinalität:

```
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for EMP[EMP]
  Using prorated density: 0.035714 of col #1 as selectvity of out-of-range/non-
existent value pred
  Table: EMP  Alias: EMP
    Card: Original: 14.000000  Rounded: 1  Computed: 0.50  Non Adjusted: 0.50
```

Die Kardinalität beträgt ursprünglich 14, so viele Datensätze sind auch in der Tabelle; das entspricht also NUM_ROWS in DBA_TABLES. Woher kommt jetzt aber die Selektivität von 0.035714? In der Abfrage haben wir ja WHERE EMPNO=123 als Prädikat. Der Wert 123 liegt jedoch außerhalb des Wertebereichs für die Spalte EMPNO.

```
SQL> select min(empno),max(empno) from emp;

MIN(EMPNO) MAX(EMPNO)
---------- ----------
      7369       7934
```

Der Optimizer muss also die Selektivität anpassen, das geschieht in der Zeile, die mit „Using prorated density:“ beginnt; dafür nimmt er hier den Wert aus DENSITY und teilt ihn durch zwei. Weil wir keine Histogramme auf der Spalte EMPNO haben, wird DENSITY als 1/NUM_DISTINCT berechnet, wie ein kurzer Check zeigt.

```
SQL> select 1/14 from dual;

      1/14
----------
.071428571
```

Wenn wir dann die ursprüngliche Kardinalität mit der Selektivität multiplizieren, erhalten wir 14 * 0.035714 = 0.5, was auf 1 aufgerundet wird. Das ist dann also die berechnete Kardinalität. Danach sehen wir die möglichen Zugriffspfade:

```
Access Path: TableScan
    Cost:  3.00  Resp: 3.00  Degree: 0
      Cost_io: 3.00  Cost_cpu: 38547
      Resp_io: 3.00  Resp_cpu: 38547
  Using prorated density: 0.035714 of col #1 as selectvity of out-of-range/non-
existent value pred
  Access Path: index (UniqueScan)
    Index: PK_EMP
    resc_io: 1.00  resc_cpu: 8461
    ix_sel: 0.071429  ix_sel_with_filters: 0.071429
    Cost: 1.00  Resp: 1.00  Degree: 1
  Using prorated density: 0.035714 of col #1 as selectvity of out-of-range/non-
existent value pred
  Access Path: index (AllEqUnique)
    Index: PK_EMP
    resc_io: 1.00  resc_cpu: 8461
    ix_sel: 0.035714  ix_sel_with_filters: 0.035714
    Cost: 1.00  Resp: 1.00  Degree: 1
```

Die besten Kosten hat der Zugriff über den Index, dort betragen die Kosten 1.00, während es beim Full Table Scan 3.00 sind. Bei den Indexberechnungen ist die erste Variante über

den Index Unique Scan die günstigere, weil dort die Selektivität des Index mit 0.071429 doppelt so hoch ist wie in der zweiten Variante. Deshalb entscheidet sich der Optimizer folgerichtig für diese Variante.

Eine ausführliche Diskussion von 10053 Traces (Stand: 10.1) finden Sie im Anhang B von [Lewis 2005].

5.10 AWR ASH, Statspack und Bstat/Estat

Seit Version 10*g* ist eigentlich nur noch das Automatic Workload Repository, kurz AWR, mit seinem Bericht die Auswertung der Wahl, wenn es um einen ersten Blick auf die Performance der gesamten Datenbank geht. Die von diesen Utilities zur Verfügung gestellte Infrastruktur erlaubt eine Vielzahl weiterer Auswertungen.

AWR baut seinerseits auf Statspack auf und Statspack seinerseits wieder auf Bstat/Estat. Die Diskussion hier folgt auch der Geschichte. Sie beginnt also mit Bstat/Estat, fährt mit den Neuerungen in Statspack fort und schließt mit den Ergänzungen, die AWR brachte, ab. Active Session History, kurz ASH, ist vollständig neu in Version 10*g* und stellt eine erweiterte und historisierte Variante von V$SESSION_WAIT bereit.

Bstat/Estat

Am Anfang gab es nur Bstat/Estat. Das steht für Beginn Statistics und End Statistics und umschreibt recht gut, worum es geht. Seit Oracle 8 heißen die entsprechenden Scripts im $ORACLE_HOME/rdbms/admin utlbstat.sql und utlestat.sql. Normalerweise lassen Sie die unter SYS laufen (bzw. CONNECT INTERNAL bis zur 8*i*) – weil Sie auf einige V$-Views zugreifen müssen – zu einer repräsentativen Zeit. Repräsentativ meint natürlich den durchschnittlichen Workload, also beispielsweise morgens um 8:00 utlbstat und abends dann gegen 18:00 utlestat, wenn alle außer dem armen DBA nach Hause gegangen sind.

Utlbstat kreiert einige Zwischentabellen. Wenn Sie dann utlestat laufen lassen, werden die aktuellen Werte mit dem Stand aus den Zwischentabellen verglichen. Damit lässt sich schön beobachten, was in der Zwischenzeit passiert ist. Utlestat druckt noch einen Performance-Report in der Datei report.txt im aktuellen Verzeichnis aus, bevor die Zwischentabellen wieder gelöscht werden. Das ist gleichzeitig auch das größte Manko bei Bstat/Estat. Man hat lediglich zwei Zeitpunkte, an denen Daten gesammelt werden, und vergleicht die Unterschiede zwischen ihnen, was meiner Meinung nach ein bisschen wenig ist. Es kann einem aber einen groben Einblick verschaffen, zumal Statspack, das einiges mehr bietet, erst ab 8.1.6 unterstützt wird. (Mit Tricks bekommt man es aber auch mit früheren Versionen zum Laufen.)

Estat/Bstat wurde unter ServerManager entwickelt, Sie brauchen also nicht zwingend SQL*Plus. An erster Stelle im Bstat/Estat Performance-Report kommt der Library Cache:

```
LIBRARY      GETS GETHITRATIO PINS PINHITRATIO RELOAD INVALIDATIONS
------------ ---- ----------- ---- ----------- ------ -------------
BODY         0    1           0    1            0     0
CLUSTER      0    1           0    1            0     0
INDEX        7    1           7    1            0     0
JAVA DATA    0    1           0    1            0     0
JAVA RESOURC 0    1           0    1            0     0
JAVA SOURCE  0    1           0    1            0     0
OBJECT       0    1           0    1            0     0
PIPE         0    1           0    1            0     0
SQL AREA     28   857         9    0            90    0
TABLE/PROCED 50   1           58   1            0     0
TRIGGER      0    1           0    1            0     0
```

Interessant sind hier schlechte Ratios oder Invalidations/Reloads. Die Kur besteht normalerweise darin, den Shared Pool zu erhöhen. Schlechte Ratios hängen auch von der Applikation ab. Je statischer die Datenbank ist, desto höher sollten die Ratios sein. Statisch ist hier so zu verstehen, dass die Strukturen (oder Metadaten) der Datenbank sich nicht ändern, kein dynamisches SQL ausgeführt wird und SQL gemeinsam genutzt werden kann. Wir erinnern uns: SQL kann gemeinsam genutzt werden, wenn Bind-Variablen verwendet werden. Falls Sie ein Data Warehouse betreiben, in dem ständig dynamisch SQL erzeugt wird, brauchen Sie sich hier also nicht über schlechte Ratios zu wundern. Danach wird die Anzahl der Benutzer zu Beginn und am Ende sowie die durchschnittliche Anzahl der Benutzer ausgegeben, was normalerweise nicht so spannend ist. Interessanter wird es da bei den Systemstatistiken, die auf v$sysstat basieren. Hier ein kleiner Auszug:

```
Statistic                     Total       Per Transact Per Logon Per Second
----------------------------- ----------- ------------ ----------- ------
background timeouts           47          47           5,22        1,04
buffer is not pinned count    186         186          20,67       4,13
…
execute count                 56          56           6,22        1,24
…
parse count (hard)            5           5            ,56         ,11
parse count (total)           28          28           3,11        ,62
parse time cpu                3           3            ,33         ,07
parse time elapsed            3           3            ,33         ,07
```

Danach werden die Wait Events ausgewertet, auch hier ein kleiner Auszug:

```
Event Name                     Count          Total Time    Avg Time
------------------------------ -------------- ------------- ------------
dispatcher timer               1              6001          6001
pmon timer                     16             4205          262,81
SQL*Net message from client    12             4017          334,75
```

Aufgrund der Two-Task-Architektur sieht man eigentlich immer irgendwelche SQL*Net-Events, auch wenn Sie gar nicht mit SQL*Net arbeiten. Die ganzen SQL*Net Events sind dann lediglich die zwischen Ihnen und dem Kernel übertragenen Daten. In diesem Beispiel gibt es aber schon eine echte Verbindung über SQL*Net: Am Dispatcher Event sieht man, dass meine Datenbank mit Shared Server konfiguriert ist. Die Events kommen zweimal, zuerst für die Benutzerprozesse und das zweite Mal für die Hintergrundprozesse. Danach kommen die Latches, auch hier ein kurzer Blick darauf:

```
LATCH_NAME         GETS        MISSES      HIT_RATIO   SLEEPS SLEEPS/MISS
------------------ ----------- ----------- ----------- ------ ------
active checkpoint  15          0           1           0      0
cache buffer handl 2           0           1           0      0
cache buffers chai 1564        0           1           0      0
```

Hier gilt: Hits sollten hoch sein, Sleeps niedrig. Manche dieser Latches lassen sich über Parameter tunen. Die Latches werden noch mal ausgewertet, diesmal No-Wait Latches, die nicht in Sleep gehen, sondern sofort einen Timeout erhalten. Auch hier sollte die Ratio hoch sein. Das Event „buffer busy waits“ wird anschließend separat ausgewiesen. Hier dient als Basis V$WAITSTAT. Anschließend werden Rollback-Segment-Statistiken ausgewiesen, Non-Default-Parameter, I/O über Tablespaces und Dateien, Beginn- und Endzeit des Reports und schließlich die verwendeten Versionen. Die meisten Ratios müssen explizit ausgerechnet werden.

STATSPACK

Seit 8.1.6. und bis Oracle 10*g* ist Statspack das Tool der Wahl. Von Nachteil bei Statspack ist der ein wenig größere Aufwand für die Konfiguration. Statspack verwendet einen dedizierten Benutzer, der sinnigerweise PERFSTAT heißt, und einen dedizierten Tablespace zur Speicherung der Performance-Daten. Der sollte mindestens 64 MB groß sein, besser noch, Sie starten gleich mit 128 MB. Im Unterschied zu Bstat/Estat erlaubt Statspack beliebig viele Zeitpunkte, zu denen ein Blick aufs System geworfen werden kann, nicht nur Beginn und Ende.

Die Auswertung erfolgt bei Statspack separat, weshalb die gesammelten Performance-Daten nicht gleich gelöscht werden. Statspack verwendet so genannte Snapshots, die aber mit den Oracle Snapshots (neuerdings Materialized Views genannt) nichts gemein haben. Ein Statspack Snapshot ist einfach eine „Aufnahme“ des Systems zu einem bestimmten Zeitpunkt (so als würden Sie mit Ihrem Fotoapparat einen Schnappschuss der Datenbank aufnehmen). Statspack berechnet netterweise viele Ratios selbst, man muss sie dem Bericht also nicht mehr separat entnehmen. Ganz vorteilhaft bei Statspack ist, dass sich das Sammeln der Daten prima über DBMS_JOB automatisieren lässt. Bstat/Estat und Statspack dürfen nicht unter demselben Benutzer laufen, da beide die Tabelle STATS$WAITSTAT verwenden. Falls Sie übrigens mal auf einem System arbeiten, bei dem Statspack zwar aufgesetzt ist, Sie aber das Passwort für den Benutzer PERFSTAT nicht kennen, können Sie sich mit folgendem Trick behelfen:

```
ALTER SESSION SET CURRENT_SCHEMA=PERFSTAT;
```

Dieses Statement bewirkt, dass Oracle bei der Namensauflösung diesen Benutzer zuerst verwendet. Wenn Sie also danach die Abfrage SELECT * FROM EMP absetzen, wird Oracle zuerst im PERFSTAT-Schema nach der Tabelle EMP suchen. Aufgepasst, die ganze Sicherheit bleibt davon unberührt. Wenn Sie also kein SELECT-Recht auf die Tabelle EMP haben, bekommen Sie immer noch ORA-942. Wenn Sie es aber mit einem DBA-Account machen, klappt es im Regelfall.

Statspack wird mit dem Script spcreate.sql (in $ORACLE_HOME/rdbms/admin) aufgesetzt. Das muss unter einem DBA-Account erfolgen. Sie müssen bereits über einen Temporary und einen Default Tablespace für den Benutzer PERFSTAT verfügen. Für das Intervall, in dem die Daten dann gesammelt werden, empfehle ich eine oder eine halbe Stunde. Im spauto.sql ist ein Beispiel für die Konfiguration über DBMS_JOB, Sie müssen

es nur noch anpassen. Die Auswertung geschieht dann über spreport.sql. Dort müssen Sie nur noch Beginn und Ende der gewünschten Auswertung angeben. Während dieser Zeit darf die Datenbank nicht gestartet worden sein, sonst ist die Auswertung ungültig. Statspack wertet die bei jedem Neustart der Datenbank neu initialisierten V$-Views aus. Wenn Sie bereits mit Bstat/Estat gearbeitet haben, wird Ihnen die Statspack-Auswertung einigermaßen bekannt vorkommen. Sie können auch eine Auswertung machen, die sich nur auf ein spezifisches SQL-Statement bezieht, dann müssen Sie noch den HASH_VALUE für das SQL mitgeben. Dazu verwenden Sie das Script sprepsql.sql. HASH_VALUE wird in der Auswertung im Abschnitt „Top SQL Statements“ mit ausgegeben.

Sehen wir uns diese Auswertung an: Statt der Spielzeugdatenbank von vorhin kommt sie diesmal von einer „richtigen“ Datenbank (VLDB unter Solaris mit EMC, Größe > 4,5 TB). Werfen wir einen Blick hinein; am Anfang finden Sie wieder einige allgemeine Angaben zur Datenbank selbst:

```
STATSPACK report for

DB Name      DB Id       Instance     Inst Num Release     Cluster Host
------------ ----------- ------------ -------- ----------- ------- ------
GUGUS        2510587121  GUGUS        1        9.2.0.4.0   NO      server03

           Snap Id Snap Time           Sessions Curs/Sess Comment
           ------- ------------------ -------- --------- -------------------
Begin Snap: 3722   03-Mar-04 23:10:10 162      776.1     STATSPACK Plus
End Snap:   3746   04-Mar-04 11:10:58 141      927.5     STATSPACK Plus
Elapsed:    720.80 (mins)

Cache Sizes (end)
~~~~~~~~~~~~~~~~~
 Buffer Cache: 8,192M Std Block Size: 16K
 Shared Pool Size: 1,024M Log Buffer: 1,024K
```

Danach bringt Statspack ein grobes Lastprofil. Hier sieht man, dass dynamisches SQL eingesetzt wird (Ratio Parse – Execute, aber nur sehr wenige Hard Parses, d.h. Bind-Variablen werden verwendet) und viel PL/SQL (Recursive Calls):

```
Load Profile
~~~~~~~~~~~~

                      Per Second      Per Transaction
                      --------------- ---------------
Redo size:            343,775.85      864,798.63
Logical reads:        8,724.33        21,946.83
Block changes:        489.55          1,231.49
Physical reads:       10,167.79       25,577.98
Physical writes:      3,315.84        8,341.30
User calls:           7.98            20.08
Parses:               13.94           35.07
Hard parses:          0.70            1.75
Sorts:                1.70            4.28
Logons:               0.05            0.14
Executes:             18.23           45.85
Transactions:         0.40

% Blocks changed per Read: 5.61 Recursive Call %: 95.29
Rollback per transaction %: 3.40 Rows per Sort: ########
```

Anschließend bringt Statspack wichtige Ratios und Shared Pool-Statistiken:

```
Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 Buffer Nowait %: 99.98 Redo NoWait %: 99.99
 Buffer Hit %: 91.89 In-memory Sort %: 99.12
 Library Hit %: 97.38 Soft Parse %: 95.00
 Execute to Parse %: 23.51 Latch Hit %: 99.96
Parse CPU to Parse Elapsd %: 2.44 % Non-Parse CPU: 99.87

 Shared Pool Statistics Begin End
 ------ ------
 Memory Usage %: 100.00 100.00
 % SQL with executions>1: 87.28 73.01
 % Memory for SQL w/exec>1: 61.39 37.96
```

Sieht nicht schlecht aus hier, sieht man mal von „Execute to Parse“ ab, aber das war ja zu erahnen. Danach kommen die Top-5-Wait-Events: Hier können manchmal auch Idle Events wie beispielsweise 'PX Deq Credit: send blkd' auftauchen. Idle Events sind Events, mit denen Sie nichts anfangen können und die Sie auch nicht tunen können. Sie können das Statspack allerdings abgewöhnen. Alle Idle Events in Statspack werden in der Tabelle STATS$IDLE_EVENT gespeichert. Wir müssen also nur noch dieses Event hinzufügen, dann wird der nächste Bericht dieses Event nicht mehr anzeigen:

```
insert into stats$idle_event values('PX Deq Credit: send blkd');
```

Nach diesem Abschnitt kommen die übrigen Wait Events und die Background Wait Events, gefolgt vom Abschnitt „Top SQL Statements“, zuerst geordnet nach Buffer Gets. „Top SQL Statements“ kommen dann noch in verschiedenen Reihenfolgen: geordnet nach Physical Reads, danach geordnet nach Executions, anschließend nach Parse Calls, Sharable Memory und schließlich auch nach Execution Count. Im Anschluss bringt Statspack noch die Auswertungen, basierend auf V$SYSSTAT. Unschön finde ich hier, dass die Auswertung per Default alphabetisch sortiert, aber das kann man ja im Script anpassen. Danach kommt das I/O pro Tablespace, dort sehen Sie auch die Buffer Waits. Anschließend zeigt der Bericht das File I/O. Danach kommen die Buffer Wait-Statistiken, die auf V$WAITSTAT basieren:

```
Buffer wait Statistics for DB: GUGUS Instance: GUGUS Snaps: 3722 -3746
-> ordered by wait time desc, waits desc

                     Tot       Wait        Avg
Class              Waits       Time (s)   Time (ms)
------------------ ----------- ---------- ---------
bitmap index block 7,051       6,891      977
data block         69,009      543        8
segment header     9,949       190        19
file header block  472         10         20
undo block         340         1          2
undo header        462         0          0
1st level bmb      10          0          8
2nd level bmb      8           0          8
```

Anschließend berichtet Statspack Enqueues und Rollback-Statistiken. Bei den Latch-Statistiken sollte die Prozentzahl der Misses sehr klein sein. Idealerweise sollte sie gegen den Wert 0 tendieren. Die Latch-Statistiken werden dann auch wieder detaillierter, Latches mit Sleeps werden noch mal explizit dargestellt. Danach kommt noch Dictionary und Library Cache, gefolgt vom Shared Pool Advisory. Abschließend kommen ein SGA Breakdown und allfällige Ressource Limits, bevor die Auswertung mit den Parametern endet. Im

Unterschied zu Bstat/Estat listet Statspack alle Parameter auf und auch, ob sie sich im Auswertungszeitraum änderten:

```
...
Parameter Name                 Begin value (if different)
------------------------------ --------------------------------- ---------
O7_DICTIONARY_ACCESSIBILITY    FALSE
archive_lag_target             0
audit_trail                    FALSE
background_dump_dest           /dbdata/GUGUS/admin/bg
```

Statspack benutzt verschiedene Schwellwerte, insbesondere bei den Auswertungen der SQL Statements, die über die Prozedur MODIFY_STATSPACK verändert werden können. Die Defaults finde ich persönlich meist ausreichend. Es gibt auch Scripts zum Aufräumen der Statspack-Tabellen, das sind dann sppurge.sql und sptrunc.sql. Raten Sie mal, welches Script DELETE mit einer WHERE-Klausel und welches TRUNCATE verwendet?

Man kann natürlich auch machen den Benutzer PERFSTAT exportieren und die Daten dann in eine andere Datenbank für Auswertungszwecke importieren. Statspack existiert nach wie vor auch in Version 10 oder 11 und kann dort auch eingesetzt werden. Das ist natürlich vor allem dann interessant, wenn Sie keine Lizenz für AWR haben. Allerdings würde ich davon abraten, sowohl Statspack wie AWR zu verwenden, das wäre dann Overkill.

Automatic Workload Repository (AWR)

Oracle 10*g* und Oracle 11 schließlich automatisieren das Tuning über AWR und ADDM. Hier werden Sie auch öfters mit den Advisories arbeiten, aber von Zeit zu Zeit wird es hilfreich sein, einen Blick auf die AWR-Schnappschüsse zu werfen. Wenn Sie bereits mit Statspack gearbeitet haben, dürfte Ihnen der AWR-Bericht sehr bekannt vorkommen. AWR baut auf Statspack auf. AWR ist eine Weiterentwicklung von Statspack und Statspack seinerseits eine Weiterentwicklung von Bstat/Estat.

Im Unterschied zu Statspack müssen Sie AWR nicht explizit aufsetzen, da AWR ja bereits im Kernel vorhanden ist. Sie müssen nur sicherstellen, dass STATISTICS_LEVEL zumindest auf TYPICAL steht, was aber der Voreinstellung entspricht. Das Script für das Erstellen des AWR-Berichts ist awrrpt.sql und unter Unix in $ORACLE_HOME/rdbms/admin zu finden. Sie können auswählen, ob Sie den Bericht als Text oder im HTML-Format wollen. Der Bericht enthält zusätzliche Informationen, die in Statspack nicht berichtet wurden, wie beispielsweise verschiedene Metriken. Statspack kann in 10*g* auch noch verwendet werden. Da Statspack aber seit 9.2 nicht mehr erweitert wurde und AWR/ADDM eine weit umfangreichere Funktionalität anbieten, ist davon eher abzuraten.

Es folgen einige Beispiele für Informationen, die nur im AWR-Bericht sichtbar sind, wie beispielsweise die Time Model-Statistiken. Hier ein entsprechender Ausschnitt:

```
FTEST/ftest  Snaps:                                          599-622
-> Total time in database user-calls (DB Time): 172.5s
-> Statistics including the word "background" measure background process
   time, and so do not contribute to the DB time statistic
-> Ordered by % or DB time desc, Statistic name
```

```
Statistic Name                                       Time (s) % of DB Time
-------------------------------------------- ----------------- ------------
sql execute elapsed time                               151.5         87.8
DB CPU                                                 145.3         84.2
PL/SQL execution elapsed time                           93.8         54.4
parse time elapsed                                      22.9         13.3
hard parse elapsed time                                 20.8         12.0
PL/SQL compilation elapsed time                          2.4          1.4
connection management call elapsed time                  1.3           .8
hard parse (sharing criteria) elapsed time               1.0           .6
failed parse elapsed time                                0.4           .2
repeated bind elapsed time                               0.1           .1
sequence load elapsed time                               0.0           .0
hard parse (bind mismatch) elapsed time                  0.0           .0
DB time                                                172.5          N/A
background elapsed time                                169.8          N/A
background cpu time                                     73.3          N/A
```

Sehr schön an dieser Darstellung finde ich, dass hier auf den ersten Blick ersichtlich ist, in welchem Bereich wie viel Zeit jeweils verbraucht wird. SQL versus PL/SQL, Parse, Bind, Execute, Benutzerprozess versus Oracle-Hintergrundprozesse – man sieht alles.

Auch die Zuordnung der Zeit zur Wait-Klasse ist nur aus dem AWR-Bericht ersichtlich. So sehen Sie gleich, ob die Wartezeit in der Datenbank oder außerhalb verbracht wird:

```
Wait Class                              DB/Inst: FTEST/ftest  Snaps: 599-622
-> s  - second
-> cs - centisecond -     100th of a second
-> ms - millisecond -    1000th of a second
-> us - microsecond - 1000000th of a second
-> ordered by wait time desc, waits desc

                                                          Avg
                                          Total Wait     wait    Waits
Wait Class                 Waits   %Time      Time (s)    (ms)   /txn
-------------------- ------------ ------ ---------- ------- -------
System I/O               34,718     .0         87       2    13.3
User I/O                  2,483     .0         21       9     1.0
Commit                    1,299     .1          5       4     0.5
Concurrency                 247     .0          4      18     0.1
Application               1,391     .0          1       1     0.5
Other                       491     .0          1       2     0.2
Configuration                 2     .0          0     142     0.0
Network                  28,628     .0          0       0    11.0
```

Im AWR-Bericht sehen Sie auch die Betriebssystemstatistiken; die Namen der Statistiken sollten zum großen Teil selbsterklärend sein, VM bedeutet Virtual Memory, und RSRC steht für den Oracle Resource Manager:

```
Statistic                                     Total
-------------------------------- --------------------
AVG_BUSY_TIME                              214,076
AVG_IDLE_TIME                            1,192,738
AVG_SYS_TIME                                94,783
AVG_USER_TIME                              119,293
BUSY_TIME                                  214,076
IDLE_TIME                                1,192,738
SYS_TIME                                    94,783
USER_TIME                                  119,293
RSRC_MGR_CPU_WAIT_TIME                           0
VM_IN_BYTES                          2,842,198,016
VM_OUT_BYTES                             1,769,472
PHYSICAL_MEMORY_BYTES                2,138,427,392
NUM_CPUS                                         1
```

Active Session History (ASH)

ASH steht wie AWR erst ab Version 10*g* zur Verfügung. ASH ist kurz gesagt eine historisierte und ausgebaute Version von V$SESSION_WAIT. Sie sehen dort neben den Waits z.B. auch, welches Ihre Top-SQL-Anweisungen waren, wann was passierte und mehr.

Für das Erstellen des ASH-Berichts wird das Script ashrpt.sql verwendet. Nach dem Starten des Scripts müssen Sie auswählen, ob Sie den Bericht als Text oder in HTML erstellen möchten. Als weitere Parameter müssen Sie angeben, welchen Zeitraum Sie untersuchen und mit welcher Startzeit Sie beginnen wollen.

Sehen wir uns das mal am Beispiel an, der erzeugte Bericht beginnt mit den allgemeinen Informationen:

```
ASH Report For FTEST/ftest


DB Name         DB Id    Instance     Inst Num Release     RAC Host
------------ ----------- ------------ -------- ----------- --- ------------
FTEST         2933034983 ftest               1 10.2.0.1.0  NO  fhaas-ch

CPUs        SGA Size       Buffer Cache        Shared Pool    ASH Buffer
---- ------------------ ------------------ ------------------ --------------
1        276M (100%)       188M (68.1%)        52M (18.8%)     2.0M (0.7%)

          Analysis Begin Time:   22-Jun-06 02:43:35
            Analysis End Time:   24-Jun-06 14:43:38
                 Elapsed Time:     3,600.1 (mins)
                 Sample Count:         747
     Average Active Sessions:        0.03
  Avg. Active Session per CPU:        0.03
```

Danach wird es interessanter, es folgen die Top Events für Benutzer und Hintergrundprozesse:

```
Top User Events                  DB/Inst: FTEST/ftest  (Jun 22 02:43 to 14:43)
                                                          Avg Active
Event                               Event Class     % Activity   Sessions
----------------------------------- --------------- ---------- ----------
enq: TX - row lock contention       Application          39.63       0.01
CPU + Wait for CPU                  CPU                  13.39       0.00
db file scattered read              User I/O             10.58       0.00
db file sequential read             User I/O             10.44       0.00
control file sequential read        System I/O            1.61       0.00
```

Es folgen die Waits für diese Events. Sehr schön ist hier, dass auch die Bedeutung der Parameter für jedes Wait Event ausgegeben wird.

```
Top Event P1/P2/P3 Values        DB/Inst: FTEST/ftest  (Jun 22 02:43 to 14:43)

Event                          % Event  P1 Value,P2 Value,P3 Value  % Activity
------------------------------ ------- ------------------------------ -------
Parameter 1                Parameter 2                Parameter 3
-------------------------- -------------------------- ----------------------
enq: TX - row lock contention    39.63  "1415053318","393245","1574"   39.63
name|mode                  usn<<16 | slot             sequence

db file sequential read          16.20                "1","6153","1"    0.27
file#                      block#                     blocks

db file scattered read           10.84               "1","55447","2"    0.13
file#                      block#                     blocks
```

Es folgen die Top SQL Command Types für die Berichtsperiode:

```
                                          Distinct            Avg Active
SQL Command Type                            SQLIDs % Activity   Sessions
----------------------------------------- ---------- ---------- ----------
DELETE                                             4      45.38       0.02
PL/SQL EXECUTE                                    10       6.69       0.00
```

Es geht auch noch genauer, der Bericht liefert uns auch (wenn möglich, d.h. falls noch vorhanden) die Top-SQL-Anweisungen im Detail:

```
Top SQL Statements              DB/Inst: FTEST/ftest  (Jun 22 02:43 to 14:43)
       SQL ID    Planhash % Activity Event                          % Event
------------- ----------- ---------- ------------------------------ --------
3wsk3s8f8q33p   204855851      39.63 enq: TX - row lock contention     39.63
 delete from scott.emp where empno=7934

d8u9c31yw9r9h   442337976       5.89 db file scattered read             2.68
** SQL Text Not Available **
```

Es geht dann weiter über Top SQL using Literals, Top Sessions, Top Blocking Sessions, Top Sessions mit Parallel Query, Top DB Objects, Top DB Files und Top Latches zum Abschnitt Activity over Time, mit dem der Bericht endet:

```
Activity Over Time              DB/Inst: FTEST/ftest  (Jun 22 02:43 to 14:43)
-> Analysis period is divided into smaller time slots
-> Top 3 events are reported in each of those slots
-> 'Slot Count' shows the number of ASH samples in that slot
-> 'Event Count' shows the number of ASH samples waiting for
   that event in that slot
-> '% Event' is 'Event Count' over all ASH samples in the analysis period
                        Slot                                  Event
Slot Time (Duration)  Count Event                            Count % Event
-------------------- ------ ------------------------------ ------ ------
06:00:00 (360.0 min)    308 enq: TX - row lock contention     264  35.34
                            db file sequential read            26   3.48
                            CPU + Wait for CPU                 12   1.61
12:00:00 (360.0 min)    213 CPU + Wait for CPU                 56   7.50
                            db file scattered read             42   5.62
                            db file sequential read            41   5.49
18:00:00 (360.0 min)     14 CPU + Wait for CPU                  8   1.07
```

Sie sehen hier auf den ersten Blick, zu welcher Zeit welche Top Events maßgeblich waren. Wie man hier erkennt, ist morgens zwischen 6:00 und 12:00 am meisten passiert, und zwar applikatorisch. Das Event „enq:TX – row lock contention“ bedeutet, dass zwei (oder mehr) Prozesse die gleichen Daten verändern wollten.

5.11 Das Tracing von PL/SQL

Für das Tracing von PL/SQL stehen Ihnen zwei Wege offen. In Version 11 können Sie den hierarchischen Profiler benutzen. In früheren Versionen gibt es nur DBMS_PROFILER, das schauen wir uns zuerst an.

DBMS_PROFILER

Bis zur Version 8.1.5 war das Tracen von PL/SQL recht schwierig. Zwar konnte man das im PL/SQL verwendete SQL rausbekommen und untersuchen, aber der PL/SQL-Code

selbst konnte nicht direkt analysiert werden. Dafür gibt's seit 8.1.5 glücklicherweise den PL/SQL Profiler. Der Profiler besteht aus dem DBMS_PROFILER Package, das zuerst unter einem DBA-Account mit dem Script profload.sql installiert werden muss. Das Script ist wie immer in $ORACLE_HOME/rdbms/admin zu finden. Der Benutzer, unter dem Sie dann das Profiling laufen lassen, muss auch die Profiling-Tabellen installiert haben. Das erfolgt über das Script proftab.sql. DBMS_PROFILER verwendet man so:

- Profiling aktivieren
- Test ausführen
- Profiling stoppen (damit werden die Profilingdaten gespeichert)
- Profiling-Daten auswerten

Das Aktivieren des Profilers geschieht über die Funktionen im DBMS_PROFILER-Package. Um ihn zu starten, verwenden Sie START_PROFILER, zum Stoppen STOP_PROFILER. Typischerweise baut man das in LOGON- und LOGOFF-Trigger ein. Ein Beispiel:

```
create or replace trigger on_logon after logon on frank.schema
 declare
 err number;
 begin
 err:=DBMS_PROFILER.START_PROFILER (to_char(sysdate,'dd-Mon-YYYY hh:mi:ss'));
 end;
 /

 create or replace trigger on_logoff before logoff on frank.schema
 declare
 err number;
 begin
 err:=DBMS_PROFILER.STOP_PROFILER ;
 end;
 /
```

Für die Auswertung benötigen Sie die so genannte RUNID. Die finden Sie in der Tabelle PLSQL_PROFILER_RUNS. Beachten Sie hier auch, dass das beim Starten mitgegebene Datum, im RUN_COMMENT taucht es auf:

```
select runid, run_date, RUN_COMMENT from plsql_profiler_runs order by 1;

 RUNID     RUN_DATE  RUN_COMMENT
 --------- --------- ----------------------------------------
 8         25-JAN-00 25-Jan-2003 08:46:07
 9         25-JAN-00 25-Jan-2003 08:47:16
 10        25-JAN-00 25-Jan-2003 09:16:54
```

Danach können Sie mit profsum.sql – skurrilerweise im Verzeichnis $ORACLE_HOME/plsql/demo – die Daten auswerten. Die Auswertung enthält immer auch die Zeilen des Source Code:

```
...
================Results for run #9 made on 25-JAN-03 08:47:16============

 (25-Jan-2003 08:47:16) Run total time: 1777.59 seconds
 Unit #1: <anonymous>.<anonymous> - Total time: .00 seconds
 Unit #2: <anonymous>.<anonymous> - Total time: .00 seconds
 Unit #3: SYS.DBMS_APPLICATION_INFO - Total time: .00 seconds
...
```

```
Unit #6: FRANK.MYTEST - Total time: .10 seconds
 1 PACKAGE BODY MYTEST AS
 2   PROCEDURE doit (dept# IN number, cnt OUT number) AS
 3 BEGIN
 4 201 .09392746 .00046730 SELECT count(*) INTO cnt
...
```

Statt profsum.sql kann man auch seine eigene Auswertung machen. In Metalink (http://metalink.oracle.com) sollten Sie hierzu einige Anregungen finden.

Noch ein Hinweis: Es besteht die Möglichkeit, PL/SQL Source Code zu „wrappen". Per Default wird PL/SQL-Source Code ja lesbar (in ASCII) abgelegt. Damit kann der Code natürlich auch leicht modifiziert werden. Um dies zu verhindern und zum Schutz des geistigen Eigentums besteht in Oracle die Möglichkeit, mit dem so genannten PL/SQL Wrapper Utility den Code zu „wrappen". Wenn der Code gewrapped ist, ist er in binärer Form abgelegt und somit nicht mehr lesbar. Solche PL/SQL-Module können also mit dem PL/SQL Profiler zwar noch analysiert werden, ohne Zugriff auf den blanken Source Code können Sie damit aber nicht allzu viel anfangen. Immerhin können Sie in diesem Fall dem Programmierer noch mitteilen, in welchen Prozeduren/Funktionen etc. die meiste Zeit verbraucht wird.

DBMS_HPROF

In Version 11 wurde das Profiling noch erweitert, dort existiert DBMS_HPROF. Generell ist der Ablauf aber der gleiche wie mit DBMS_PROFILER, das heißt: Sie aktivieren das Profiling, führen dann Ihren PL/SQL Code aus und stoppen das Profiling dann wieder. Das Profiling erzeugt eine Trace-Datei, die sich weiter auswerten lässt.

Sie benötigen auch wieder spezielle Profiling-Tabellen, die mit Hilfe des Scripts $ORACLE_HOME/rdbms/admin/dbmshptab.sql im jeweiligen Schema angelegt werden.

Für das Profiling müssen Sie als Parameter das Verzeichnis, in dem die Trace-Datei dann geschrieben wird, also das korrespondierende Oracle DIRECTORY, und den Namen der Trace-Datei angeben. Hier ein Beispiel:

```
exec dbms_hprof.start_profiling('TEMP_DIR','new_way.trc');
exec new_way;
exec dbms_hprof.stop_profiling;
```

Im nächsten Schritt muss die resultierende Trace-Datei weiter analysiert werden, was über DBMS_HPROF.ANALYZE gschieht:

```
exec :runid := dbms_hprof.analyze('TEMP_DIR','new_way.trc',run_comment=>'New way
Run');
```

Nach der Analyse können Sie die Profiling-Ergebnisse entweder den Profiling-Tabellen entnehmen, oder Sie lassen sich über das Kommandozeilen-Utility plshprof einen Bericht im HTML-Format generieren:

```
plshprof -output /tmp/new_way /tmp/new_way.trc
```

Die erste Seite des Berichts enthält dann summarische Informationen und die Hyperlinks zu den anderen Seiten; folgende Auswertungen sind verfügbar:

- Function Elapsed Time (microsecs) Data sorted by Mean Subtree Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Function Name
- Function Elapsed Time (microsecs) Data sorted by Total Descendants Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Total Function Call Count
- Function Elapsed Time (microsecs) Data sorted by Mean Subtree Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Mean Function Elapsed Time (microsecs)
- Function Elapsed Time (microsecs) Data sorted by Mean Descendants Elapsed Time (microsecs)
- Module Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)
- Module Elapsed Time (microsecs) Data sorted by Module Name
- Module Elapsed Time (microsecs) Data sorted by Total Function Call Count
- Namespace Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)
- Namespace Elapsed Time (microsecs) Data sorted by Namespace
- Namespace Elapsed Time (microsecs) Data sorted by Total Function Call Count

Für weitere Details verweise ich auf Metalink Note 763944: „How to tune plsql applications and identify hot spots“.

5.12 Performance und SQL*Net

5.12.1 SQL*Net Tracing

Bei SQL*Net Tracing denkt man zunächst nicht an Performance-Auswertungen. Aber auch das ist möglich und sinnvoll, insbesondere seit Oracle 8*i*. Damals wurde die Möglichkeit eingeführt, SQL*Net Trace-Dateien mit Timestamps zu versehen. In Verbindung mit dem stärksten Tracelevel (SUPPORT oder 16) erlaubt uns dies zu sehen, wann was passiert(e). Dazu müssen wir zuerst die Konfigurationsdatei sqlnet.ora anpassen. Diese Datei finden Sie per Default im $ORACLE_HOME/network/admin. Das Verzeichnis kann aber über die Umgebungsvariable TNS_ADMIN gesetzt werden. Dort setzen wir die Trace-Parameter für den Client:

```
TRACE_LEVEL_CLIENT = SUPPORT
TRACE_FILE_CLIENT = sqlnet.trc
TRACE_DIRECTORY_CLIENT = C:\TEMP
TRACE_TIMESTAMP_CLIENT = ON
```

Sind die Parameter erst mal gesetzt, wird beim nächsten Start eines Client-Programms die Trace-Datei sqlnet.trc (und eventuell noch mehr Trace-Dateien je nach SQL*Net-Konfiguration) in das Verzeichnis C:\TEMP geschrieben. Ich habe mich hier für den Test mit SQL*Plus als Benutzer SCOTT bei der Datenbank angemeldet und dann nur ein `SELECT * FROM DEPT` ausgeführt. Im Trace sehe ich dann:

```
...
[05-MÄR-2004 11:19:43:640] nspsend: 01 00 00 00 00 12 73 65 |......se|
[05-MÄR-2004 11:19:43:640] nspsend: 6C 65 63 74 20 2A 20 66 |lect.*.f|
[05-MÄR-2004 11:19:43:640] nspsend: 72 6F 6D 20 64 65 70 74 |rom.dept|
....
```

10 Mikrosekunden später werden die Header der Spalten von der Datenbank zurückgeliefert:

```
...
[05-MÄR-2004 11:19:43:650] nsprecv: 06 06 00 00 00 06 44 45 |......DE|
[05-MÄR-2004 11:19:43:650] nsprecv: 50 54 4E 4F 00 00 00 00 |PTNO....|
[05-MÄR-2004 11:19:43:650] nsprecv: 00 00 00 00 01 01 80 00 |........|
.....
```

Gleich darauf kommen auch die Werte:

```
...
[05-MÄR-2004 11:19:43:650] nsprecv: 07 07 02 C1 15 08 52 45 |......RE|
[05-MÄR-2004 11:19:43:650] nsprecv: 53 45 41 52 43 48 06 44 |SEARCH.D|
[05-MÄR-2004 11:19:43:650] nsprecv: 41 4C 4C 41 53 15 03 00 |ALLAS...|
```

Jetzt sehen Sie auch, warum Sie hier einen Level 16 Trace benötigen. Ohne den würden Sie die zurückgelieferten Werte in der Trace-Datei nicht sehen. Allerdings nützt Ihnen das auch nichts, falls Sie den SQL*Net-Verkehr verschlüsseln. Das ist über Oracles Advanced Security-Option möglich. Verwenden Sie diese Art des Tracing, falls SQL*Net verwendet wird und es Hinweise darauf gibt, dass zu viel Zeit im Netzwerk verbracht wird.

5.12.2 Event 10079

Der Vollständigkeit halber sei hier auch Event 10079, das bereits mit Oracle 7.3 eingeführt wurde, erwähnt. Es ist ähnlich wie das normale SQL*Net Tracing. Mit Event 10079 teilen Sie Oracle mit, dass auch der SQL*Net-Verkehr in den Trace-Dateien mit getraced werden soll. Sie können hier vier Level angeben:

- 1 – damit wird das Tracing für Netzwerkoperationen aktiviert;
- 2 – damit werden auch die über SQL*Net übertragenen Daten in die Trace-Datei geschrieben;
- 4 – damit wird das Tracing für Datenbank-Links aktiviert;
- 8 – damit werden auch die über Datenbank-Links übertragenen Daten in die Trace-Datei geschrieben.

Eingesetzt wird dieses Event meines Wissens eher selten, das normale SQL*Net Tracing ist im Regelfall vollkommen ausreichend. Interessant dürfte der Einsatz wohl vor allem sein, wenn man den Verkehr über Datenbanklinks genauer inspizieren will.

5.12.3 Trace Assistant

Der Level 16 SQL*Net Trace ist auch die Grundvoraussetzung für den SQL*Net Trace Assistant trcasst. Damit lassen sich Level 16 Traces noch besser formatieren. Hier ein kurzer Vorgeschmack aus 10.2, was man damit anstellen kann. Der Aufruf des Utilities ohne weitere Parameter zeigt uns, wozu es in der Lage ist:

```
C:\Oracle\db\admin\FTEST\udump>trcasst

Dienstprogramm Trace-Assistent: Version 10.2.0.1.0 Production am 18. Juni 2006 20:34:04

Copyright (c) 2001, 2005, Oracle.  All rights reserved. Alle Rechte vorbehalten.

TNS-04302: Fehler bei Verwendung des Trace-Assistenten: Fehlender Dateiname.

Verwendung: trcasst [options] <filename>
      [options]  Standardwerte sind -odt -e0 -s
      <filename>  immer das letzte Argument
    -o[c|d][u|t][q]  Net Services- und TTC-Informationen
      [c]  Zusammenfassung der Net Services-Informationen
      [d]  Detaillierte Net Services-Informationen
      [u]  Zusammenfassung der TTC-Informationen
      [t]  Detaillierte TTC-Informationen
      [q]  SQL-Befehle
    -s  Statistiken
    -e[0|1|2]  Fehlerinformationen, Standard ist 0
      [0]  NS-Fehlernummern ³bersetzen
      [1]  Fehler³bersetzung
      [2]  Fehlernummern ohne Uebersetzung
    -l[a|i <connection_id>]  Verbindungsinformationen
      [a]  Auflisten aller Verbindungen in einer Trace-Datei
      [i <connection_id>]  Decodieren einer angegebenen Verbindung
```

Die <connection_id> existiert im SQL*Net Trace für jedes NS Connect-Paket. Falls Sie die verwenden wollen, müssen Sie sie erst einmal über trcasst –la ermitteln, bevor Sie die <connection-id> spezifisch über trcasst –li untersuchen. Der Trace Assistant ist insbesondere bei Traces für Sessions, die über Shared Server verbunden sind, sehr nützlich.

Untersuchen wir anhand eines Beispiels, wie eine solche Auswertung mit den Voreinstellungen aussieht. Am Anfang sehen Sie allgemeine Informationen zum Verbindungsaufbau:

```
---> Send 261 bytes - Connect packet  timestamp=03-JUL-2006 11:28:12:784
Current NS version number is: 313.
Lowest NS version number can accommodate is: 300.
       Maximum SDU size: 2048
       Maximum TDU size: 32767
       NT protocol characteristics:
              Asynchronous mode
              Callback mode
              Test for more data
              Full duplex I/O
              Urgent data support
              Handoff connection to another
              Grant connection to another
       Line turnaround value: 0
       Connect data length: 203
```

```
    Connect data offset: 58

    Connect data maximum size: 512
           Native Services wanted
           Authentication is linked and specify
           NAU doing O3LOGON - DH key foldedin
           Native Services wanted
           Authentication is linked and specify
           NAU doing O3LOGON - DH key foldedin
 (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=fhaas-ch.ch.oracle.com)(PORT
 =1521))(CONNECT_DATA=(SERVICE_NAME=FTEST.ch.oracle.com)(CID=(PROGRAM=C
 :\Oracle\db\10.2\bin\sqlplus.exe)(HOST=fhaas-ch)(USER=fhaas))))
```

Danach wird der Verkehr zwischen Client und Server aufgelistet. Hier ein entsprechender Ausschnitt aus der Trace-Datei:

```
<--- Received 17 bytes - Data packet  timestamp=03-JUL-2006 11:28:12:894
V6 Oracle func complete (TTISTA)

---> Send 319 bytes - Data packet  timestamp=03-JUL-2006 11:28:12:894
Start of user function (TTIFUN)
    New v8 bundled call (OALL8) Cursor # 0 Parse Fetch

<--- Received 526 bytes - Data packet  timestamp=03-JUL-2006 11:28:12:904
Describe information (TTIDCB)
```

Das zieht sich so durch bis zum Schluss, dort gibt es dann noch summarische Informationen. Diese sind vor allem für das Einstellen der SDU interessant, dort suchen Sie nach „Maximale Byte“:

```
Trace-Datei-Statistiken:
------------------------
Start-Zeitstempel : 03-JUL-2006 11:28:12:784
End-Zeitstempel   : 03-JUL-2006 11:28:21:757
Gesamtanzahl von Sessions: 2

DATABASE:
  Vorgangsanzahl:    0 OPEN-Vorgänge,     7 PARSE-Vorgänge,
                     7 EXECUTE-Vorgänge,     7 FETCH-Vorgänge

    Parse-Anzahl:
       0 PL/SQL,      0 SELECT,       0 INSERT,      0 UPDATE,      0 DELETE,
       0 LOCK,        0 TRANSACT,     0 DEFINE,      0 SECURE,      7 OTHER

    Ausführungsanzahl mit SQL-Daten:
       0 PL/SQL,      0 SELECT,       0 INSERT,      0 UPDATE,      0 DELETE,
       0 LOCK,        0 TRANSACT,     0 DEFINE,      0 SECURE,      7 OTHER

  Paketrate: 4.285714285714286 Pakete pro Vorgang gesendet
  Aktuell geöffnete Cursor: 0
  Maximal geöffnete Cursor: 0

ORACLE NET SERVICES:
  Gesamte Aufrufe  :        30 gesendet,         28 empfangen,        14 oci
  Anzahl der Byte:      4521 gesendet,      5211 empfangen

  Durchschnittliche Byte:   150 pro Paket gesendet,  186 pro Paket empfangen
  Maximale Byte:      1111 gesendet,      1020 empfangen
  Pakete gesamt:     30 gesendet,     28 empfangen
```

5.12.4 Trcsess Utility

Dieses Utility hat nur indirekt etwas mit SQL*Net Tracing zu tun, sieht man einmal davon ab, dass Shared Server eine entsprechende SQL*Net-Konfiguration voraussetzt, weshalb es hier auch aufgeführt ist. Das Trcsess Utility dient der Zusammenfassung verschiedener Trace-Dateien in einer einzigen Trace-Datei. Dies dient zwei Zwecken: Zum einen können Sie die mit DBMS_MONITOR erstellten Traces zusammenfassen. Das sehen Sie beim Aufruf des Utilities. Beachten Sie, wie Sie als Parameter Client Identifier, Service, Action oder auch Module angeben können:

```
C:\Oracle\db\admin\FTEST\udump>trcsess

oracle.ss.tools.trcsess.SessTrcException: SessTrc-00002: Fehler bei Verwendung von
Session Trace: Falsche Parameter uebergeben.

trcsess [output=<output file name >]  [session=<session ID>] [clientid=<clientid>]
[service=<service name>] [action=<action name>] [module=<module name>] <trace file
names>

output=<output file name> output destination default being standard
output.session=<session Id> session to be traced.
Session id is a combination of session Index & session serial number e.g. 8.13.

clientid=<clientid> clientid to be traced.
service=<service name> service to be traced.
action=<action name> action to be traced.
module=<module name> module to be traced.
<trace_file_names> Space separated list of trace files with wild card '*'
supported. C:\Documents and Settings\fhaas>trcasst
```

Das zweite Einsatzgebiet für Trcsess sind über Shared Server angemeldete Traces von Sessions. In diesem Fall können die Informationen ja über mehrere Trace-Dateien verstreut sein, was die Analyse zunächst ziemlich erschwert.

Trcsess erzeugt aus allen (angegebenen) Dateien wieder eine einzige Trace-Datei, die wie gewohnt von TKPROF etc. weiterverarbeitet werden kann.

5.13 Tuning mit dem Enterprise Manager

Der Enterprise Manager bietet eine graphische Benutzeroberfläche, mit deren Hilfe sich viele Arbeiten, insbesondere im DBA-Bereich, komfortabel erledigen lassen. Der Enterprise Manager existiert in zwei Varianten: Grid Control und Database Control. Grid Control ist sozusagen die Luxusfassung. Damit können Sie alle Datenbanken in Ihrer Systemumgebung zentral über eine Konsole verwalten. Für Grid Control sollten Sie deshalb eine dedizierte Datenbank anlegen. Im Unterschied dazu ist Database Control (=dbconsole) eine abgespeckte Variante von Grid Control, die auf einer einzigen Datenbank installiert wird. Database Control läuft immer nur gegen eine Datenbank, nicht gegen mehrere. Seit Version 10*g* wird beim Anlegen der Datenbank mit Hilfe des Datenbankassistenten standardmäßig auch Database Control installiert. Deshalb gehen wir im Folgenden auf das Tuning mit Database Control ein. Das ist auch für Grid Control gültig, mit dem Unterschied, dass Sie damit noch mehr „machen“ können.

Sie greifen auf den Enterprise Manager über Ihren Browser zu, die URL ist http://<<hostname>>:<<portname>>/em; ersetzen Sie //<<hostname>> durch den Namen des Datenbankservers und <<portname>> durch den konkreten HTTP-Port. Die Port-Nummer finden Sie in der Datei $ORACLE_HOME/install/portlist.ini. Die notwendigen Privilegien für die Arbeit mit dem Enterprise Manager können Sie einrichten, wenn Sie von der Startseite aus zuerst auf den Reiter Setup und dann im Reiter auf der linken Seite „Navigation Administrators“ anwählen. Der Benutzer muss bereits in der Datenbank vorhanden sein. Nach dem Aufsetzen haben nur die Benutzer SYS, SYSTEM, DBSNMP und SYSMAN die entsprechenden Privilegien. Es wird empfohlen, hier speziell eingerichtete Benutzer zu verwenden.

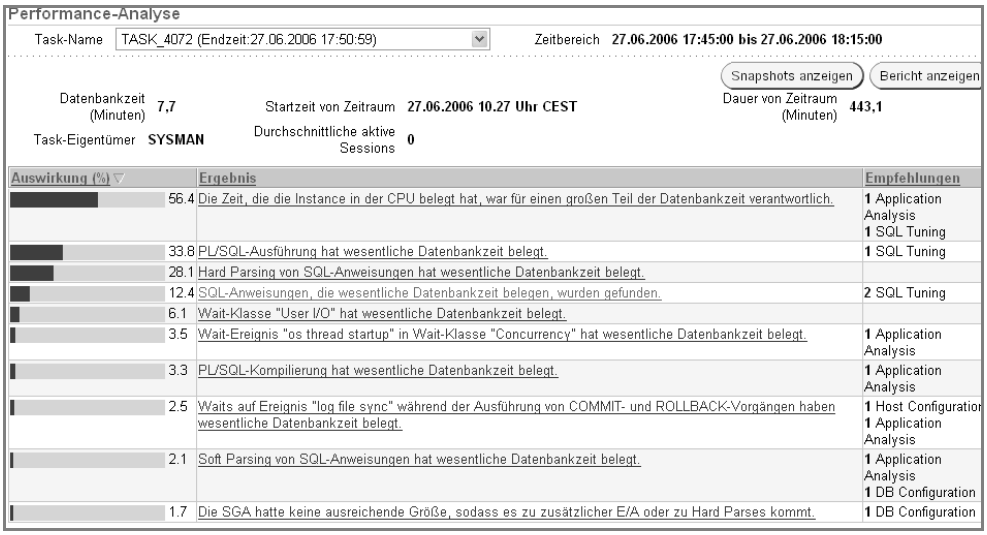
Die wesentlichen Kontrollen für das Tuning erreichen Sie im Database Control, wenn Sie auf der Startseite auf den Link „Performance“ klicken:

Auf der Seite „Performance“ finden Sie dann die Grafiken zur CPU Utilization, dem Festplattendurchsatz, den durchschnittlich aktiven Sessions und dem Datenbankdurchsatz. Im Abschnitt „Additional Monitoring Links“ finden Sie weitere Links wie „Top Activity“ oder „Top Consumers“.

Auf die Performance-Analysen des ADDM greifen Sie typischerweise über den Button „ADDM jetzt ausführen“ zu. Das Ergebnis kann dann zum Beispiel wie im folgenden Bild aussehen.

Damit erhalten Sie nicht nur einen repräsentativen Überblick über die angegebene Zeit und sehen auf einen Blick, wo die meiste Zeit verbraucht wird, sondern können für jede Wait-Klasse über die verschiedenen Empfehlungen auch gleich die (hoffentlich) richtigen Verbesserungsmaßnahmen einleiten.

In jedem Fall ist bei der Arbeit mit dem Enterprise Manager das Anlegen von Baseline-Metriken zu empfehlen. Dies geschieht über den Link „Normalisierte Baseline-Metriken“ auf der Performance-Seite bzw. „Metrik-Baselines“ auf der Startseite. Sind Sie auf der Metrik-Baselines-Seite, können Sie zwischen verschiedenen Einstellungen wählen; Standardeinstellung ist keine aktive Baseline. Eine Baseline ist einfach ein Satz von gespeicherten Einstellungen und Statistiken innerhalb eines festgelegten Zeitraums. Diese Baseline kann


Performance-Analyse

Task-Name TASK_4072 (Endzeit:27.06.2006 17:50:59) Zeitbereich **27.06.2006 17:45:00 bis 27.06.2006 18:15:00**

Snapshots anzeigen | Bericht anzeigen

Datenbankzeit (Minuten) **7,7** Startzeit von Zeitraum **27.06.2006 10.27 Uhr CEST** Dauer von Zeitraum (Minuten) **443,1**

Task-Eigentümer **SYSMAN** Durchschnittliche aktive Sessions **0**

Auswirkung (%)	Ergebnis	Empfehlungen
56.4	Die Zeit, die die Instance in der CPU belegt hat, war für einen großen Teil der Datenbankzeit verantwortlich.	**1** Application Analysis **1** SQL Tuning
33.8	PL/SQL-Ausführung hat wesentliche Datenbankzeit belegt.	**1** SQL Tuning
28.1	Hard Parsing von SQL-Anweisungen hat wesentliche Datenbankzeit belegt.	
12.4	SQL-Anweisungen, die wesentliche Datenbankzeit belegen, wurden gefunden.	**2** SQL Tuning
6.1	Wait-Klasse "User I/O" hat wesentliche Datenbankzeit belegt.	
3.5	Wait-Ereignis "os thread startup" in Wait-Klasse "Concurrency" hat wesentliche Datenbankzeit belegt.	**1** Application Analysis
3.3	PL/SQL-Kompilierung hat wesentliche Datenbankzeit belegt.	**1** Application Analysis
2.5	Waits auf Ereignis "log file sync" während der Ausführung von COMMIT- und ROLLBACK-Vorgängen haben wesentliche Datenbankzeit belegt.	**1** Host Configuration **1** Application Analysis
2.1	Soft Parsing von SQL-Anweisungen hat wesentliche Datenbankzeit belegt.	**1** Application Analysis **1** DB Configuration
1.7	Die SGA hatte keine ausreichende Größe, sodass es zu zusätzlicher E/A oder zu Hard Parses kommt.	**1** DB Configuration

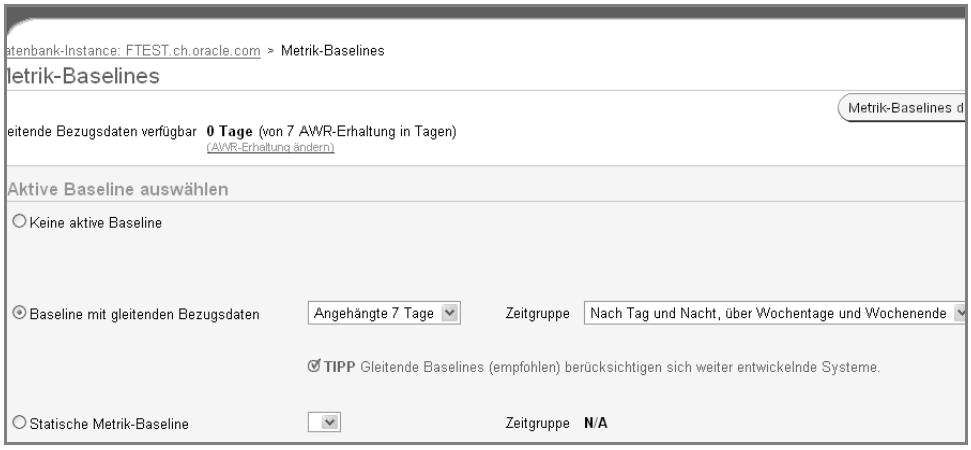
dann als Ausgangspunkt für Vergleiche genommen werden. Angenommen, Sie haben eine Verarbeitung, die täglich wiederkehrt. Gestern war noch alles in Ordnung, aber heute beklagen sich die Benutzer über schlechte Antwortzeiten. Kein Problem – wenn Sie eine Baseline haben, können Sie die heutige Verarbeitung einfach mit dieser Baseline vergleichen und sehen, wo die Unterschiede sind.

Zwei Arten von Baselines lassen sich erstellen: gleitende und statistische. Statistische Baselines sind interessant, wenn Sie sehr unterschiedliche Verwendungsmuster zu verschiedenen Zeiten und/oder für verschiedene Programme haben. Demgegenüber berücksichtigt eine gleitende Baseline fortlaufend immer die Daten der letzten Woche. Bei einer Baseline mit gleitenden Bezugsdaten können Sie außerdem angeben, wie weiter unterteilt werden soll, und dabei die folgenden Zeitgruppen auswählen:

- Keine
- Nach Tag und Nacht
- Nach Wochentagen und Wochenende
- Nach Tag und Nacht, über Wochentage und Wochenende
- Nach Wochentag

Wählen Sie die Zeitgruppe, die den Verarbeitungsmustern in Ihrer Datenbank am besten entspricht. Über den Button „Adaptive Schwellenwerte festlegen“ können Sie außerdem die Schwellenwerte für die verschiedenen Metriken wie z.B. die System-Antwortzeit (Hundertstelsekunden) verändern. Das erfordert allerdings eine sehr genaue Kenntnis des Systems. Die folgende Abbildung zeigt die entsprechende Seite.

Sie sehen dort auch, dass die Baseline mit gleitenden Bezugsdaten die empfohlene Variante ist. Sie sehen im Bild auch den Link „AWR-Erhaltung ändern“. Wenn Sie ihm folgen, kommen Sie auf die Seite „Einstellungen bearbeiten“, wo Sie festlegen können, für wie lange die AWR-Daten gespeichert bleiben sollen, welchen Wert STATISTICS_LEVEL hat



und wie lange das Intervall zwischen zwei AWR-Schnappschüssen sein soll. Speziell Letzteres ist sehr wichtig. Falls Sie eine Performance-Analyse vornehmen sollen und noch nicht bekannt ist, wo das Problem denn nun genau liegt, sollten Sie hier starten und erst einmal das Intervall verkürzen. Voreingestellt sind 60 Minuten, dieser Wert ist zu grob. Gehen Sie runter auf 10 Minuten. Sie brauchen dann auch eventuell mehr Platz im SYS-AUX Tablespace, sollten also auch ein wachsames Auge darauf werfen.

In Version 11 wurde dieser Bereich weiter ausgebaut, die verschiedenen Baselines werden jetzt unter dem Begriff „AWR Baseline“ zusammengefasst. Eine AWR Baseline enthält einen Bereich von AWR-Schnappschüssen für eine bestimmte Periode. Dabei kann die Baseline statisch sein, beispielsweise die letzten drei Tage des Quartalsende, oder sich wiederholend wie beispielsweise immer der letzte Freitag im Monat. Oracle 11 führte auch Moving Window Baselines – Baselines, die ständig nachgeführt werden. Vorgegeben in dieser Version ist das SYSTEM_MOVING_WINDOW Baseline, das die AWR-Daten der letzten acht Tage enthält. Das kann selbstverständlich angepasst werden, allerdings müssen Sie dann eventuell auch die Vorhaltezeit für AWR-Daten über das GUI bzw. DBMS_STATS.ALTER_STATS_HISTORY_RETENTION anpassen. Die Vorhaltezeit für AWR-Daten beträgt voreingestellt acht Tage in Version 11 und sieben Tage in früheren Versionen.

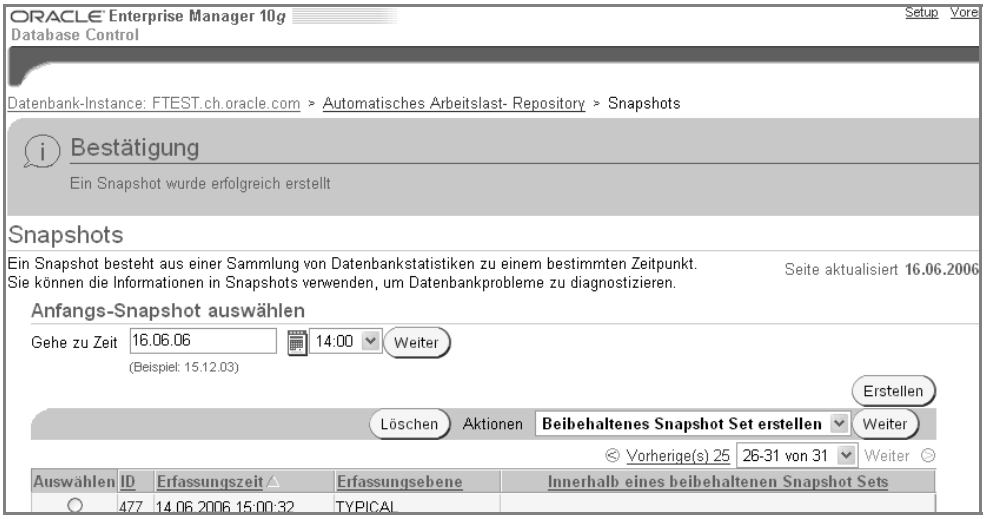
Sie können in Version 11 auch Baselines für die Zukunft erstellen. Dazu dienen Baseline Templates. Hier ein Beispiel für den nächsten Jahresabschluss. Bitte beachten Sie, dass keine Schnappschüsse angegeben werden (die kennen wir ja noch nicht!):

```
begin
dbms_workload_repository.create_baseline_template(
start_time => to_date('30.12.2009','DD.MM.YYYY',
end_time => to_date('02.01.2010','DD.MM.YYYY',
baseline_name => 'Jahresabschluss09',
template_name => 'Jahresabschluss09',
expiration => NULL);
end;
```

Schnappschüsse lassen sich in einem so genannten beibehaltenen Snapshot Set zusammenfassen. Navigieren Sie zuerst über den Link Snapshots von der Performance-Seite auf die

Snapshots-Seite. Dort wählen Sie den Anfangs-Snapshot aus, unter den Aktionen dann „Beibehaltenes Snapshot Set erstellen“, und klicken auf den Button „Weiter“. Auf der nächsten Seite wählen Sie den End-Snapshot, optional können Sie dem Snapshot Set auch noch einen Namen geben. Dies ist sehr zu empfehlen, da die systemgenerierten Namen nicht sehr aussagekräftig sind. Wenn Sie dann auf OK klicken, sollte das Snapshot Set erstellt werden und Sie sehen auf der Seite „Beibehaltene Snapshot-Sets“, welche Snapshot Sets existieren. Sie können dort auch verschiedene Aktionen auf diese Snapshot Sets auswählen. Die Aktion „Bericht anzeigen“ erzeugt einen AWR-Bericht, „ADDM ausführen“ ruft den ADDM für das Snapshot Set aus, „Zeiträume vergleichen“ erlaubt den Vergleich zweier Snapshot Sets, und „SQL Tuning Set erstellen“ erstellt ein SQL Tuning Set.

Manuell können Sie jederzeit einen Schnappschuss direkt erstellen. Klicken Sie dazu auf der Performance-Seite auf den entsprechenden Button:

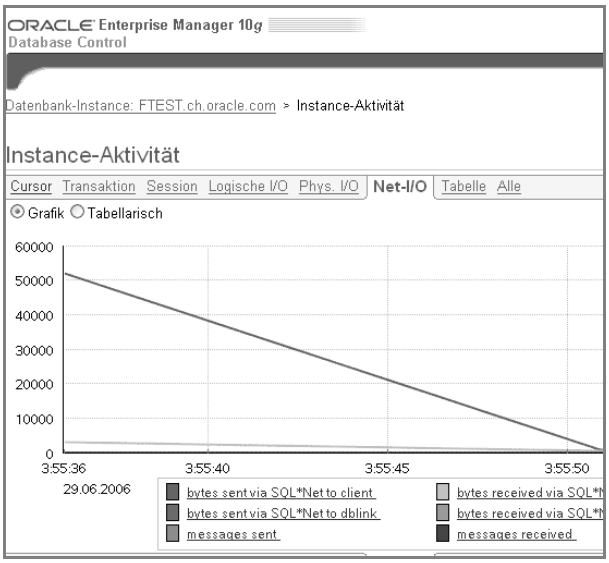


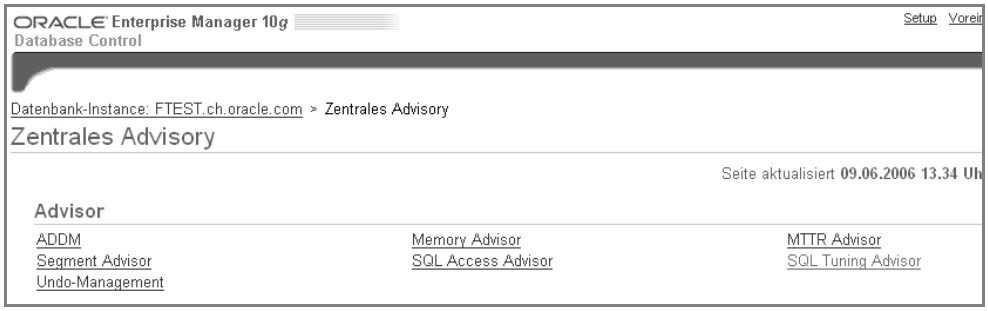
Statistiken sehen Sie im Enterprise Manager an verschiedenen Stellen; die Werte aus V$SYSSTAT – vor allem auf der Seite Instance-Aktivität – zeigt das Bild auf der nächsten Seite oben.

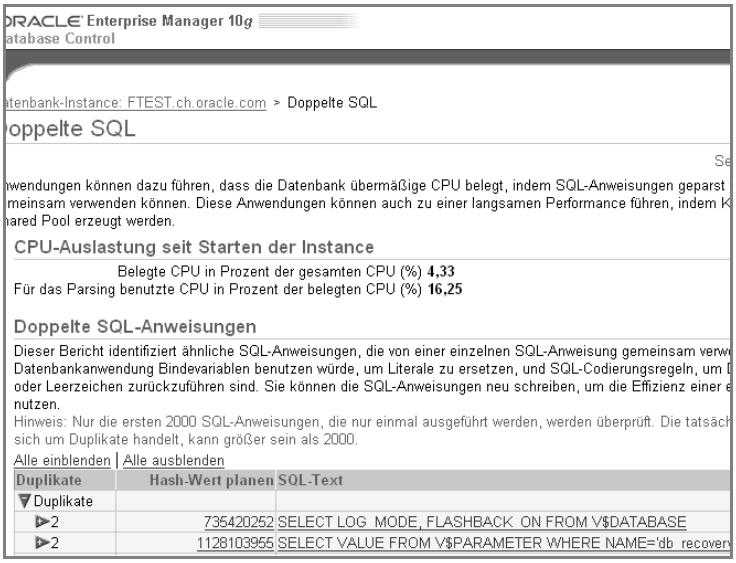
Wie Sie dort sehen, können Sie über die verschiedenen Reiter wie Cursor, Transaktion, Session etc. gezielt die entsprechenden Statistiken auswählen.

Die verschiedenen Advisories inklusive ADDM und SQL Tuning Advisor /SQL Access Advisor können Sie alle über den Link „Zentrales Advisory“ erreichen. Sie haben dort alle Advisories auf einer Seite versammelt (Bild auf der nächsten Seite mitte):

Sehr nützlich für das Tuning kann auch die Seite „Doppelte SQL“ sein. Dort sehen Sie allerdings auch Oracle-interne Anweisungen, wie aus der Abbildung ersichtlich (Bild auf der nächsten Seite unten):

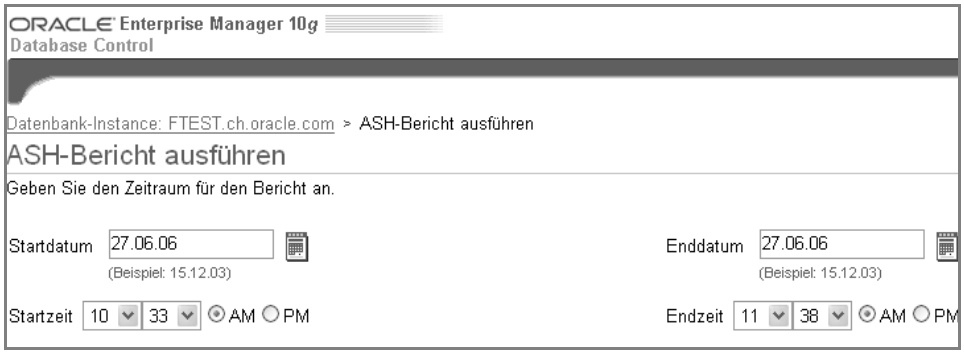

ORACLE Enterprise Manager 10g
Database Control
Datenbank-Instance: FTEST.ch.oracle.com > Instance-Aktivität
Instance-Aktivität
Cursor Transaktion Session Logische I/O Phys. I/O Net-I/O Tabelle Alle
Grafik Tabellarisch
60000
50000
40000
30000
20000
10000
0
3:55:36
3:55:40
3:55:45
3:55:50
29.06.2006
bytes sent via SQL*Net to client
bytes sent via SQL*Net to dblink
messages sent
bytes received via SQL*N
bytes received via SQL*N
messages received


ORACLE Enterprise Manager 10g
Database Control
Setup Voreir
Datenbank-Instance: FTEST.ch.oracle.com > Zentrales Advisory
Zentrales Advisory
Seite aktualisiert 09.06.2006 13.34 Uh
Advisor
ADDM
Segment Advisor
Undo-Management
Memory Advisor
SQL Access Advisor
MTTR Advisor
SQL Tuning Advisor


ORACLE Enterprise Manager 10g
atabase Control
atenbank-Instance: FTEST.ch.oracle.com > Doppelte SQL
oppelte SQL
nwendungen können dazu führen, dass die Datenbank übermäßige CPU belegt, indem SQL-Anweisungen geparst
meinsam verwenden können. Diese Anwendungen können auch zu einer langsamen Performance führen, indem K
hared Pool erzeugt werden.
CPU-Auslastung seit Starten der Instance
Belegte CPU in Prozent der gesamten CPU (%) 4,33
Für das Parsing benutzte CPU in Prozent der belegten CPU (%) 16,25
Doppelte SQL-Anweisungen
Dieser Bericht identifiziert ähnliche SQL-Anweisungen, die von einer einzelnen SQL-Anweisung gemeinsam verw
Datenbankanwendung Bindevariablen benutzen würde, um Literale zu ersetzen, und SQL-Codierungsregeln, um
oder Leerzeichen zurückzuführen sind. Sie können die SQL-Anweisungen neu schreiben, um die Effizienz einer
nutzen.
Hinweis: Nur die ersten 2000 SQL-Anweisungen, die nur einmal ausgeführt werden, werden überprüft. Die tatsäch
sich um Duplikate handelt, kann größer sein als 2000.
Alle einblenden | Alle ausblenden
Duplikate
Hash-Wert planen
SQL-Text
Duplikate
2
735420252
SELECT LOG_MODE, FLASHBACK_ON FROM V$DATABASE
2
1128103955
SELECT VALUE FROM V$PARAMETER WHERE NAME='db_recover

Interessant ist hier natürlich vor allem applikatorisches SQL. Falls Sie ein Performance-Problem haben, weil eine Session durch Locks blockiert wird, sollten Sie das über die Seite „Blockierende Sessions“ sehen können. Dort werden die entsprechenden Infos aus DBA_BLOCKERS und DBA_WAITERS dargestellt. Locks sehen Sie allgemein auf der Seite Instance-Sperren, dort können Sie neben blockierenden Locks auch benutzerdefinierte Locks bzw. generell alle Locks sich anzeigen lassen.

Den ASH-Bericht erzeugen Sie im Enterprise Manager, in dem Sie auf der Performance-Seite den Button „ASH-Bericht ausführen“ anklicken. Sie werden auf die nächste Seite weitergeleitet, wo Sie Start- und Enddatum für den Bericht eingeben müssen:

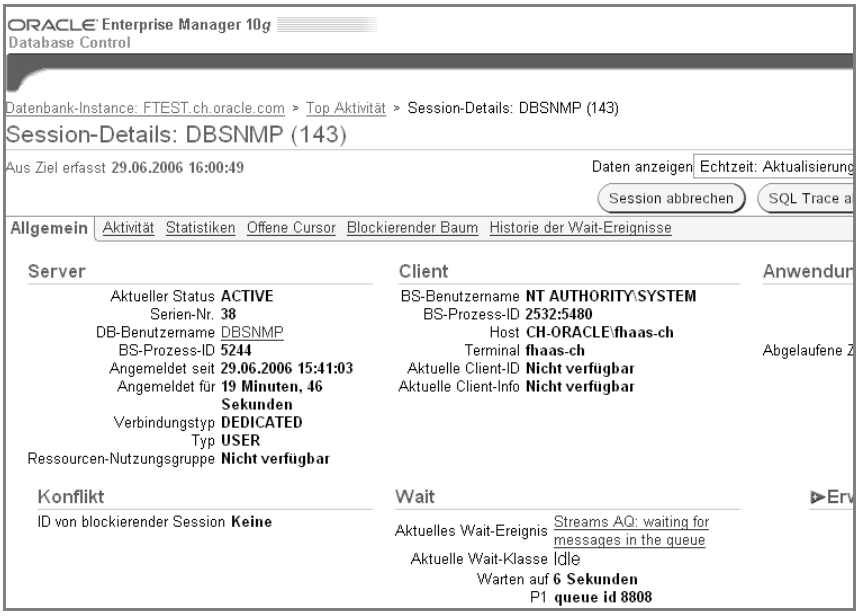


Für die Analyse einzelner Sessions ist die Seite „Sessions suchen“ hilfreich. Auf dieser Seite können Sie anhand verschiedener Suchkriterien wie SID, DB-Benutzername, Service, Modul etc. oder auch direkt über die WHERE-Klausel die Sie interessierende Session in V$SESSION suchen. Bitte beachten Sie, dass Kriterien wie Service oder Modul natürlich nur funktionieren, wenn die Session entsprechend über DBMS_MONITOR bzw. DBMS_SESSION und/oder DBMS_APPLICATION_INFO instrumentiert wurde.

Die Untersuchung der einzelnen Session startet mit der Session-Auswahl. Auf der Seite mit den Session-Details kann dann auch der SQL Trace aktiviert werden:

Wie man hier sieht, ist nicht nur der einfache SQL Trace, sondern auch ein 10046 Level 8 oder 12 Trace ist ohne Weiteres möglich.

Abgesehen davon haben Sie bereits auf der Seite mit den Session-Details über die diversen Reiter Zugriff auf wichtige Tuning-Informationen:



Sie haben, wie man hier sieht, eine Fülle von Informationen, auf die Sie zugreifen können. Für das Tuning sind natürlich vor allem die Waits inklusive der Historie der Wait-Ereignisse, die Statistiken und die jeweiligen Ausführungspläne von Bedeutung. Den Ausführungsplan für die aktuelle SQL-Anweisung erhalten Sie, in dem Sie im Abschnitt „Anwendung“ auf den Link unter „Aktuelle SQL“ klicken. Auf der Seite SQL-Details klicken Sie dann auf den Reiter „Plan“, und der Ausführungsplan wird angezeigt. Hier ein Beispiel für einen Ausführungsplan: