HANSER

Oracle Datenbankadministration mit SQL-Skripten

Alexander Kick

ISBN 3-446-40727-8

Leseprobe

Weitere Informationen oder Bestellungen unter
http://www.hanser.de/3-446-40727-8 sowie im Buchhandel

8

Performance

8 Performance

Entwickler oder Endbenutzer einer Datenbank treten gelegentlich an den Datenbankadministrator mit der Aussage heran, die Datenbank sei langsam. Diese Information ist leider wenig hilfreich. Sind das Betriebssystem und die Datenbankinstanz professionell aufgesetzt worden, so liegt das Problem an der Abfrage selbst. Man muss daher nachfragen und versuchen herauszufinden, welche Session, welches Programm, welches SQL-Statement langsam ist. Auch wenn das Problem nicht die SQL-Abfrage, sondern in der Datenbankinstanz oder dem Betriebssystem zu suchen ist, kann man so der eigentlichen Ursache auf die Spur kommen.

Informationen über den langsam laufenden Prozess lassen sich herausfinden, indem man z.B. die Aktivitätsskripte (*Kapitel 7*) benutzt. Kennt man den Benutzer, so kennt man auch sehr schnell die Session, deren Abfrage zu langsam läuft.

Ausgehend von der Session, kann man versuchen, das Problem herauszufinden. Schaut man sich den Ausführungsplan und den Text des SQL-Statements der gerade laufenden problematischen Session an, so ist man schon einen wesentlichen Schritt weitergekommen. Diesem Zweck dient das Skript in *Abschnitt 8.1*.

Hat man den Ausführungsplan, so weiß man, welche Tabellen und Indizes benutzt werden. Um abschätzen zu können, ob der von Oracle gewählte Ausführungsplan in Ordnung ist, möchte man möglichst viel über diese Objekte herausfinden. Dazu gehört die Größe der entsprechenden Tabellen- und Indexsegmente und die Größe der entsprechenden Partitionen oder Subpartitionen. Diese Aufgabe erfüllt das Skript in *Abschnitt 8.2*.

Ebenso stellt sich die Frage: Werden die richtigen Indizes benutzt oder nicht? Das Skript `tabInds.sql` liefert daher einen Überblick über die zu einer gegebenen Tabelle gehörenden Indizes und zeigt auf, welche Spalten indexiert sind.

Eine Fülle an Informationen liefert das Skript in *Abschnitt 8.4*. Da fehlende oder falsche Statistiken eine häufige Ursache für einen nicht optimalen Ausführungsplan sind, kann man basierend auf dessen Ausgabe sehr schnell feststellen, ob Objektstatistiken gar nicht berechnet oder nicht mehr aktuell sind. In diesem Fall empfiehlt es sich, die Statistiken neu und korrekt zu berechnen. Sind sie hingegen aktuell, verfügt man über viele weitere Informationen zu den Daten und beteiligten Objekte und ist der Ursache des Problems einen weiteren Schritt näher gekommen.

8.1 Ausführungsplan einer Session

8.1.1 Zweck

Oft tritt die Frage auf, warum eine Abfrage oder ein SQL-Statement langsam laufen. Ein sehr hilfreicher Anhaltspunkt ist der Ausführungsplan des gerade ausgeführten SQL-Statements. Hier sieht man auch schnell, auf welche Basistabellen oder Indizes zugegriffen wird.

Erkennt man, dass der Ausführungsplan zu umständlich ist und zu viele Ressourcen verbraucht, kann man schon früh die Entscheidung treffen, das Statement abzubrechen. Während mit den von Oracle zur Verfügung gestellten Skripten für das Automatic Workload Repository (AWR), dem Automatic Database Diagnostic Monitor (ADDM) und der Auswertung von Trace Files die Performance im nachhinein untersucht werden kann, ist es möglich, basierend auf dem Ausführungsplan eines laufenden SQL-Statements, sehr früh einzugreifen.

8.1.2 Beispiel-Report

Der Beispiel-Bericht zeigt für die Session mit SID 28 den SQL-Text und den Ausführungsplan.

```
SQL> @plan 28

SQL Text and Execution Plan for Session with SID 28

SELECT sql_text FROM   sys.v_$sqlarea t, v$session s WHERE  t.address =
s.sql_address AND    t.hash_value = s.sql_hash_value AND    s.SID = 28
AND    s.sql_hash_value != 0



---------------------------------------------------------------------------
---------------
| Id  | Operation                      |  Name            | Rows  | Bytes
| Cost (%CPU)|
---------------------------------------------------------------------------
---------------
|   0 | INSERT STATEMENT               |                  |       |
|     |  |
|   1 |  NESTED LOOPS                  |                  |     1 |  6619
|    13   (8)|
|   2 |   NESTED LOOPS OUTER           |                  |     1 |  6548
|     8  (13)|
|   3 |    NESTED LOOPS OUTER          |                  |     1 |  6533
|     7  (15)|
|*  4 |     FIXED TABLE FULL           | X$KQLFXPL        |     1 |  6506
|     |  |
|   5 |     TABLE ACCESS BY INDEX ROWID| OBJ$             |     1 |    27
|     2  (50)|
|*  6 |      INDEX UNIQUE SCAN         | I_OBJ1           |     1 |
|     |  |
|   7 |    TABLE ACCESS CLUSTER        | USER$            |     1 |    15
|     2  (50)|
|*  8 |     INDEX UNIQUE SCAN          | I_USER#          |     1 |
|     |  |
|*  9 |   FIXED TABLE FIXED INDEX      | X$KSUSE (ind:1)  |     1 |    71
|     |  |
---------------------------------------------------------------------------
---------------

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

   4 - filter("P"."KQLFXPL_HASH"<>0 AND "P"."INST_ID"=:B1 AND
              "P"."KQLFXPL_HADD"<>"P"."KQLFXPL_PHAD")
   6 - access("P"."KQLFXPL_OBJN"="O"."OBJ#"(+))
   8 - access("O"."OWNER#"="U"."USER#"(+))
   9 - filter("X$KSUSE"."INDX"=28 AND "X$KSUSE"."KSUSESQH"<>0 AND
              "X$KSUSE"."INST_ID"=:B1 AND
              BITAND("X$KSUSE"."KSSPAFLG",1)<>0 AND
```

```
                BITAND("X$KSUSE"."KSUSEFLG",1)<>0 AND
               "P"."KQLFXPL_PHAD"="X$KSUSE"."KSUSESQL" AND
               "P"."KQLFXPL_HASH"="X$KSUSE"."KSUSESQH")
SQL>
```

Unter der Spalte „Name“ werden die Tabellen und Indizes und unter „Operation“ die ausgeführten Operationen angezeigt, die zur Ausführung des SQL-Statements verwendet werden.

8.1.3 Skript

Der SQL-Text zu einer Session ID kann aus v$sqlarea ausgelesen werden. Zur Anzeige des Ausführungsplans genügt es, die entsprechenden Informationen aus v$sql_plan (und v$session) zu extrahieren und in die Tabelle plan_table einzufügen, da Oracle schon PL/SQL-Pakete (dbms_xplan) und Skripte (utlxplan, utlxpls, …) zur Anzeige eines Ausführungsplans zur Verfügung stellt. Voraussetzung ist jedoch, dass ein plan_table mit utlxplan.sql angelegt wurde.

Da ein hash_value mit Wert 0 nicht gültig ist, muss eine entsprechende Einschränkung in der Where-Clause vorgenommen werden. Damit verschiedene Anwendungen den gleichen plan_table benutzen können, sollte man diese mit einer statement_id unterscheiden. Um die Wahrscheinlichkeit zu reduzieren, dass in der plan_table schon ein Plan mit der gleichen Statement_ID enthalten ist, wird eine Zufallszeichenkette erzeugt, die dann als Statement_ID verwendet wird. Mit dbms_random.string('A', 20) wird ein String erzeugt, das 20 Zeichen lang ist und aus Groß- oder Kleinbuchstaben besteht.

Listing 8.1 plan.sql

```
SET verify OFF lines 90 pages 400 echo OFF feedback OFF termout ON
set heading off

column sql_text format a73 word_wrapped

variable stmt_id varchar2(20)
begin
   :stmt_id := dbms_random.string('A', 20);
end;
/

ttitle left "SQL Text and Execution Plan for Session with SID &&1" skip 2

SELECT sql_text
FROM   sys.v_$sqlarea t, v$session s
WHERE  t.address = s.sql_address
AND    t.hash_value = s.sql_hash_value
AND    s.SID = &&1
AND    s.sql_hash_value != 0;

INSERT INTO plan_table
  SELECT :stmt_id,
         SYSDATE,
         'REMARKS',
         operation,
         options,
         object_node,
         object_owner,
         object_name,
         0,
```

```
          'OBJECT_TYPE',
          optimizer,
          search_columns,
          ID,
          parent_id,
          position,
          COST,
          cardinality,
          bytes,
          other_tag,
          partition_start,
          partition_stop,
          partition_id,
          other,
          distribution,
          cpu_cost,
          io_cost,
          temp_space,
          access_predicates,
          filter_predicates
  FROM    sys.v_$sql_plan p, v$session s
  WHERE   p.address = s.sql_address
  AND     p.hash_value = s.sql_hash_value
  AND     s.SID = &&1
  AND     s.sql_hash_value != 0;

ttitle OFF

SELECT plan_table_output
FROM   TABLE(DBMS_XPLAN.display('plan_table', :stmt_id, 'serial'));

DELETE FROM plan_table
      WHERE STATEMENT_ID = :stmt_id;
```

Dieses Skript funktioniert nur für Oracle 9, da `v$sql_plan` in Oracle 10 einige Spalten mehr hat.

Aufgabe:

Ändern Sie das Skript so, dass es sowohl für Oracle 9 als auch Oracle 10 funktioniert. Damit es nur für Oracle 10 funktioniert, müssten lediglich beim Insert ein paar Spaltennamen hinzugefügt werden. Damit es für Oracle 9 und 10 eingesetzt werden kann, verwenden Sie bitte die in *Kapitel 6* vorgestellte Methode.

8.2 Tabellen- und Indexsegmente

8.2.1 Zweck

Fürs Performance-Tuning ist es nützlich zu wissen, wie groß die zu einer Tabelle, deren Indizes oder deren Partitionen gehörenden Segmente sind. Ebenso kann man auf diese Weise leichter herausfinden, ob die Partitionierung gut gewählt wurde.

8.2.2 Beispiel-Report

Der folgende Bericht zeigt die Namen und Größen der zur Tabelle Account und deren Indizes gehörenden Partitionen an.

```
SQL> @tabSegs
Enter table name: account

Segments of table ACCOUNT

SEGMENT_NAME                   PARTITION_NAME                        MB
------------------------------ ------------------------------ ----------
ACCOUNT                        ACCOUNTS_2006_A                       .1
                               ACCOUNTS_2006_C                      12.0
                               ACCOUNTS_2006_O                     688.0
                               ACCOUNTS_MAX_A                      360.0
                               ACCOUNTS_MAX_C                        .1
                               ACCOUNTS_MAX_O                        .1


Segments of indexes belonging to table ACCOUNT

SEGMENT_NAME                   PARTITION_NAME                        MB
------------------------------ ------------------------------ ----------
ACCOUNT_IDX1                   ACCOUNTS_2006_A                       .1
                               ACCOUNTS_2006_C                      3.0
                               ACCOUNTS_2006_O                    144.0
                               ACCOUNTS_MAX_A                      72.0
                               ACCOUNTS_MAX_C                        .1
                               ACCOUNTS_MAX_O                        .1

ACCOUNT_PK                     ACCOUNTS_2006_A                       .1
                               ACCOUNTS_2006_C                      8.0
                               ACCOUNTS_2006_O                    448.0
                               ACCOUNTS_MAX_A                     231.0
                               ACCOUNTS_MAX_C                        .1
                               ACCOUNTS_MAX_O                        .1

SQL>
```

8.2.3 Skript

Die entsprechenden Informationen können aus `dba_segments` extrahiert werden. Im Unterschied zum `segs.sql`-Skript in *Abschnitt 5.7.2*, das Informationen zu den in einem Tablespace befindlichen Segmenten ausgibt, zeigt das `tabSegs.sql`-Skript die Informationen zur eingegebenen Tabelle und deren Indizes.

Listing 8.2 tabSegs.sql

```
SET verify OFF linesize 74 pages 200 echo OFF feedback OFF termout ON

col segment_name FORM a30
col partition_name FORM a30
col mb heading 'MB' format 999G999D0
break ON segment_name SKIP 1

ACCEPT tabName prompt 'Enter table name: '

@tabName

ttitle left "Segments of table &&tabName"
SELECT   segment_name,
         partition_name,
         ROUND(bytes / 1024 / 1024, 1) mb
FROM     sys.dba_segments
WHERE    segment_name = '&&tabName'
ORDER BY 1, 2;

ttitle "Segments of indexes belonging to table &&tabName"
```

```
SELECT   segment_name,
         partition_name,
         ROUND(bytes / 1024 / 1024, 1) mb
FROM     sys.dba_segments s, (SELECT index_name
                              FROM   sys.dba_indexes
                              WHERE  table_name = '&&tabName') i
WHERE    i.index_name = s.segment_name
ORDER BY 1, 2;
```

Da die Umwandlung in Großbuchstaben auch in anderen Skripten benötigt wird, wird diese in eine separate Datei `tabName.sql` ausgelagert.

Listing 8.3 tabName.sql

```
SET termout OFF

COLUMN tabName new_value tabName
SELECT UPPER('&&tabName') tabName
FROM   DUAL;

SET termout ON
```

8.3 Zu einer Tabelle gehörende Indizes

8.3.1 Zweck

Es kann vorkommen, dass einerseits Indizes fehlen, andererseits aber zu viele oder unnötige Indizes angelegt wurden. Zum Zwecke des Performance-Tuning (aber auch der Bereinigung unnötiger Indizes) möchte man daher wissen, welche Indizes zu einer bestimmten Tabelle vorhanden sind und insbesondere, auf welchen Spalten diese definiert sind.

8.3.2 Beispiel-Report

Die beiden folgenden Berichte zeigen zu den eingegebenen Tabellen die Indizes und die Index-Spalten in der korrekten Reihenfolge.

```
SQL> @tabInds
Enter Table Name: sales
Enter Table Owner: acc

                         Indexes for table SALES

Index Name       Loc Type           Uni Table Name     Column Name
---------------- --- -------------- --- -------------- ------------------
ACC.SCBJIX           BITMAP         NO  ACC.CUSTOMERS  CUST_GENDER

SQL> @tabInds
Enter Table Name: account
Enter Table Owner: acc

                        Indexes for table ACCOUNT

Index Name       Loc Type           Uni Table Name     Column Name
---------------- --- -------------- --- -------------- ------------------
ACC.ACCOUNT_IDX1 LOC FUNCTION-BASED NO  ACC.ACCOUNT    SUBSTR("CUSTOMER_I
                     NORMAL                            D",1,3)
```

```
ACC.ACCOUNT_PK   LOC NORMAL          YES ACC.ACCOUNT     ACCOUNT_ID
                                                         VALID_UNTIL
                                                         STATUS
```

Der Index `acc.scbjix` ist global (`loc null`), nicht einzigartig (`uni no`) und ein Bitmap-Index. Der Index `account_idx1` ist lokal und ein funktionsbasierter Index mit dem Index-Ausdruck `SUBSTR("CUSTOMER_ID",1,3)`, wohingegen `account_pk` ein eindeutiger (normaler) B*-Baum-Index ist.

Zum besseren Verständnis geben wir hier den Constraint-Ausschnitt (beim Create-Table-Statement) und Create-Index-Statement für die Indizes auf der Tabelle Account an (die vollständigen Statements sind auch in *Abschnitt 4.6.1* abgedruckt).

```
CONSTRAINT account_PK PRIMARY KEY (account_id, valid_until, status)
USING INDEX local

CREATE INDEX acc.account_idx1 ON acc.ACCOUNT(SUBSTR(customer_id,1,3))
LOCAL;
```

Der Bitmap-Index auf der Tabelle Sales wird im folgenden Abschnitt erläutert.

8.3.3 Skript

Informationen über Indexspalten sind in `dba_ind_columns` abgelegt. Für funktions-basierte Indizes muss man jedoch `dba_ind_expressions` zu Rate ziehen. Im folgenden Test sieht man, dass bei einem funktionsbasierten Index in beiden Tabellen Einträge vorhanden sind.

```
SQL> create index dualind on sys.dual (upper(dummy),lower(dummy),dummy);

Index created.

SQL> col index_owner form a11
col index_name form a10
col column_expression form a17
col column_name form a17
select index_owner, index_name, column_position, column_expression
from dba_ind_expressions where index_name='DUALIND';

INDEX_OWNER INDEX_NAME COLUMN_POSITION COLUMN_EXPRESSION
----------- ---------- --------------- -----------------
SYS         DUALIND                 1 UPPER("DUMMY")
SYS         DUALIND                 2 LOWER("DUMMY")

SQL> select index_owner, index_name, column_position, column_name
from dba_ind_columns where index_name='DUALIND';

INDEX_OWNER INDEX_NAME COLUMN_POSITION COLUMN_NAME
----------- ---------- --------------- -----------------
SYS         DUALIND                 1 SYS_NC00002$
SYS         DUALIND                 2 SYS_NC00003$
SYS         DUALIND                 3 DUMMY
```

Für die funktionsbasierten Spalten eines Index möchte man daher `dba_ind_expressions`, für die anderen Spalten hingegen `dba_ind_columns` verwenden.

Das innere Select im Skript in *Listing 8.4* kombiniert daher diese beiden Tabellen (Join über `column_position`) und stellt somit für alle Indexarten die entsprechenden Spalteninformationen zur Verfügung. Da die Spalte `column_expression` in `dba_ind_expressions` jedoch den Typ Long hat, muss dieser Typ in einen Char-Typ umgewandelt werden, wenn man

die nvl-Funktion verwenden will. Andernfalls würde man eine Fehlermeldung wie folgt erhalten:

```
SQL> select nvl(column_expression,'sdfl') from dba_ind_expressions where
index_name='UD';
select nvl(column_expression,'sdfl') from dba_ind_expressions where
index_name='UD'
                           *
ERROR at line 1:
ORA-00932: inconsistent datatypes: expected LONG got CHAR
```

Diesen Zweck (der Umwandlung von Long in Char) erfüllt die programmierte Funktion `get_ind_col_expr`, die vor der Ausführung der Hauptabfrage erzeugt und danach wieder gelöscht wird, da der `Cast`-Operator leider nicht für den Typ Long verwendet werden kann.

Mit diesem Wissen können wir uns nun an das eigentliche Skript wagen:

Listing 8.4 tabInds.sql

```
SET linesize 73 pages 80 verify OFF echo OFF feedback OFF termout ON
set recsep off

COLUMN index_name       format a16 word_wrapped heading 'Index Name'
COLUMN locality         format a3 trunc         heading 'Loc'
COLUMN table_name       format a14 word_wrapped heading 'Table Name'
COLUMN column_name      format a18 word_wrapped heading 'Column Name'
COLUMN index_TYPE       format a14 word_wrapped heading 'Type'
COLUMN uni              format a3               heading 'Uni'

break ON index_name SKIP 1 on locality ON index_TYPE ON uni on table_name

ACCEPT tabName CHAR -
               PROMPT "Enter Table Name: "
ACCEPT tabOwner CHAR -
               PROMPT "Enter Table Owner: "

@tabName

ttitle center 'Indexes for table &&tabName' SKIP 2

CREATE OR REPLACE FUNCTION get_ind_col_expr(
  p_table   IN   VARCHAR2,
  p_index        VARCHAR2,
  p_col          VARCHAR2)
  RETURN VARCHAR2
IS
  v_col_expr   DBA_IND_EXPRESSIONS.column_expression%TYPE;
BEGIN
  SELECT column_expression
  INTO   v_col_expr
  FROM   DBA_IND_EXPRESSIONS
  WHERE  table_name = p_table
  AND    index_name = p_index
  AND    column_position = p_col;

  RETURN v_col_expr;
END;
/

SELECT   i.owner || '.' || i.index_name index_name,
         pi.locality,
         i.index_type,
         DECODE(i.uniqueness, 'UNIQUE', 'YES', 'NONUNIQUE', 'NO') uni,
         (c.table_owner || '.' || c.table_name) table_name,
         c.column_name
FROM     sys.DBA_INDEXES i,
         sys.DBA_PART_INDEXES pi,
```

```
          (SELECT c.table_name,
                  c.index_name,
                  c.table_owner,
                  c.index_owner,
                  NVL(get_ind_col_expr(e.table_name,
                                       e.index_name,
                                       e.column_position),
                      c.column_name) column_name,
                  c.column_position
           FROM   sys.DBA_IND_COLUMNS c, sys.DBA_IND_EXPRESSIONS e
           WHERE  c.index_name = e.index_name(+)
           AND    c.index_owner = e.index_owner(+)
           AND    c.column_position = e.column_position(+)) c
WHERE    i.index_name = c.index_name
AND      i.owner = c.index_owner
AND      i.owner = pi.owner(+)
AND      i.index_name = pi.index_name(+)
AND      i.table_name = '&&tabName'
AND      i.table_owner = UPPER('&&tabOwner')
ORDER BY i.index_name, i.owner, c.column_position;

DROP FUNCTION get_ind_col_expr;
```

Ebenso zu beachten ist die unterschiedliche Bedeutung der Spalten `table_owner` und `table_name` in den Views `dba_ind_columns` und `dba_ind_expressions`, verglichen mit z.B. `dba_indexes`. In den beiden ersten Views geben diese Auskunft, zu welcher Tabelle die indexierte Spalte gehört, in Letzterer dagegen, auf welcher Tabelle der Index erzeugt wurde. Bei Join-Indizes können diese (Tabellen) unterschiedlich sein, wie das folgende Beispiel zeigt:

```
SQL> CREATE TABLE SALES (cust_id NUMBER, amount_sold NUMBER);

Table created.

SQL> CREATE TABLE customers (cust_id NUMBER PRIMARY KEY,
NAME VARCHAR(20), cust_gender CHAR);

Table created.

SQL> CREATE BITMAP INDEX acc.scbjix
  2  ON SALES(customers.cust_gender)
  3  FROM SALES, customers
  4  WHERE SALES.cust_id = customers.cust_id;

Index created.

SQL> col index_owner form a15
SQL> col owner form a15
SQL> col index_name form a15
SQL> col table_owner form a15
SQL> col table_name form a15
SQL> select owner, index_name, table_owner, table_name
from dba_indexes where index_name='SCBJIX';

OWNER           INDEX_NAME      TABLE_OWNER     TABLE_NAME
--------------- --------------- --------------- ---------------
ACC             SCBJIX          ACC             SALES

SQL> select index_owner, index_name, table_owner, table_name
from dba_ind_columns where index_name='SCBJIX';  2

INDEX_OWNER     INDEX_NAME      TABLE_OWNER     TABLE_NAME
--------------- --------------- --------------- ---------------
ACC             SCBJIX          ACC             CUSTOMERS

SQL>
```

Der Bitmap-Index wurde also auf der Tabelle `sales` erzeugt, wohingegen die indexierte Spalte aus der Tabelle `customers` stammt. Diesbezügliche Informationen sind in der View `dba_join_ind_columns` enthalten:

```
SQL> select inner_table_owner, inner_table_name, inner_table_column,
  2         outer_table_owner, outer_table_name, outer_table_column
  3  from dba_join_ind_columns where index_name='SCBJIX';

INNER_TABLE_OWNER    INNER_TABLE_NAME     INNER_TABLE_COLUMN
-------------------- -------------------- --------------------
OUTER_TABLE_OWNER    OUTER_TABLE_NAME     OUTER_TABLE_COLUMN
-------------------- -------------------- --------------------
ACC                  SALES                CUST_ID
ACC                  CUSTOMERS            CUST_ID


SQL>
```

8.4 Tabellen- und Index-Statistiken

8.4.1 Zweck

Der Oracle-Optimizer entwickelt bei einer Abfrage verschiedene Ausführungspläne. Deren Kosten werden anhand der Objekt- und System-Statistiken berechnet. Der günstigste Ausführungsplan kommt dann zur Ausführung.

Fehlende oder nicht aktuelle Objektstatistiken können daher zu Performance-Problemen führen. Auch wenn diese nicht Ursache für eine schlechte Performance von Abfragen sind, so helfen sie doch, die Ursache der Performance-Probleme aufzuspüren.

Greift eine Abfrage auf eine Tabelle zu, möchte man nicht nur die globalen Statistikdaten zusammen mit der Spaltenstatistik, sondern auch die Statistiken auf Partitions- und Subpartitionsebene und die entsprechenden Indexstatistiken sehen. Das vorgestellte Skript generiert daher alle vorhandenen Statistiken, die etwas mit der betroffenen Tabelle zu tun haben, und speichert diese Informationen in einer Datei, da der Output recht umfangreich werden kann.

Die folgende Tabelle gibt einen Überblick über die Statistikspalten in den einzelnen Dictionary Views.

Tabelle 8.1 Bedeutung der Statistik-Spalten

Spaltenname	Bedeutung
NUM_ROWS	Zahl der Zeilen im Objekt
BLOCKS	Anzahl benutzter Blöcke
EMPTY_BLOCKS	Anzahl leerer Blöcke
AVG_SPACE	durchschnittlich freier Platz
CHAIN_CNT	Anzahl verketteter Zeilen
AVG_ROW_LEN	durchschnittliche Zeilengröße

Spaltenname	Bedeutung
SAMPLE_SIZE	Größe des zur Analyse benutzten Sample
LAST_ANALYZED	Datum, wann die letzte Statistikanalyse erfolgte
GLOBAL_STATS	für partitionierte Indizes: ob Statistiken durch Analyse des ganzen Index gesammelt (YES) oder auf Basis der (Sub-) Partitionsstatistiken geschätzt wurden (NO)
USER_STATS	ob Statistiken durch den Benutzer gesetzt wurden (YES) oder nicht (NO)
BLEVEL	B*-Tree Level: Tiefe des Baums vom Wurzel-Block bis zu den Blatt-Blöcken
LEAF_BLOCKS	Anzahl der Blatt-Blöcke im Index
DISTINCT_KEYS	Anzahl der unterschiedlichen indizierten Werte
AVG_LEAF_BLOCKS_PER_KEY	durchschnittliche Anzahl an Blatt-Blöcken, in denen ein Index-Wert auftritt
AVG_DATA_BLOCKS_PER_KEY	durchschnittliche Anzahl Data-Blöcke, auf die ein Index-Eintrag zeigt
CLUSTERING_FACTOR	gibt einen Hinweis auf die Ordnung der Tabellenzeilen bzgl. der Indexwerte: Wenn der Wert ungefähr der Blockanzahl entspricht, dann ist die Tabelle gut geordnet. In diesem Fall zeigen die Index-einträge eines Blocks größtenteils auf Zeilen im gleichen Datenblock. Wenn der Wert nahe bei der Zeilenanzahl ist, dann ist die Tabelle ungeordnet, sodass nur wenige Indexeinträge im gleichen Block auf einen gleichen Datenblock zeigen.

8.4.2 Beispiel-Report

Der folgende recht umfangreiche Bericht zeigt die Statistiken für die Tabelle Account, deren Partitionen und Subpartitionen und deren Indizes.

```
SQL> @stats
Show Statistics for Table: account
Enter Table Owner: accounting

Table Statistics for ACCOUNT

                              avg
                 emp  avg     row
   rows  blks     ty  spa  cha  len Sampled Last
  1000s 1000s blks   ce   in  gth   1000s Analyzed
------- ----- ---- ---- ---- ---- ------- --------------
 13'149    67    0    0    0   73   3'287 03.02.06 15:17

Partition Statistics for ACCOUNT

                                        avg
                          emp  avg       row
                  rows  blks   ty  spa  cha  len Sampled Last
Partition Name    1000s 1000s blks   ce   in  gth   1000s Analyzed
```

```
-------------- ------- ----- ---- ---- ---- ---- ------- --------------
ACCOUNTS_2006     8'685    45    0    0    0   73   8'685 03.02.06 15:06
ACCOUNTS_MAX      4'462    23    0    0    0   73   4'462 03.02.06 15:10

Subpartition Statistics for ACCOUNT

                                      avg
                            emp  avg       row
Subpartition       rows  blks   ty  spa  cha len Sampled Last
Name              1000s 1000s blks   ce   in gth   1000s Analyzed
--------------- ------- ----- ---- ---- ---- ---- ------- --------------
ACCOUNTS_2006_A       0     0    0    0    0    0         03.02.06 14:39
ACCOUNTS_2006_C     142     1    0    0    0   73     142 03.02.06 14:39
ACCOUNTS_2006_O   8'545    44    0    0    0   73     566 03.02.06 14:41
ACCOUNTS_MAX_A    4'462    23    0    0    0   73   4'462 03.02.06 14:51
ACCOUNTS_MAX_C        0     0    0    0    0    0         03.02.06 14:51
ACCOUNTS_MAX_O        0     0    0    0    0    0         03.02.06 14:51

Column Statistics for ACCOUNT

                         num     num
                    distinct   Nulls         Sampled Last
Column Name            1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID             2'956       0       1   3'287 03.02.06 15:17
CURRENCY                   0       0       1       6 03.02.06 15:17
CUSTOMER_ID              867       0       1     591 03.02.06 15:17
OTHER_DATA                 0  13'149       1         03.02.06 15:17
STATUS                     0       0       1       6 03.02.06 15:17
TECH_INS_DATE              0       0       1       6 03.02.06 15:17
TECH_UPD_DATE              0       0       1       6 03.02.06 15:17
VALID_FROM                 0       0       1       6 03.02.06 15:17
VALID_UNTIL                0       0       1       6 03.02.06 15:17

Partition Column Statistics for Partition: ACCOUNTS_2006

                         num     num
                    distinct   Nulls         Sampled Last
Column Name            1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID             2'862       0       1   8'685 03.02.06 15:06
CURRENCY                   0       0       1       5 03.02.06 15:06
CUSTOMER_ID              648       0       1     527 03.02.06 15:06
OTHER_DATA                 0   8'685       1         03.02.06 15:06
STATUS                     0       0       1       5 03.02.06 15:06
SYS_NC00010$               0       0       1       5 03.02.06 15:06
TECH_INS_DATE              0       0       1       5 03.02.06 15:06
TECH_UPD_DATE              0       0       1       5 03.02.06 15:06
VALID_FROM                 0       0       1       5 03.02.06 15:06
VALID_UNTIL                0       0       1       5 03.02.06 15:06

Partition Column Statistics for Partition: ACCOUNTS_MAX

                         num     num
                    distinct   Nulls         Sampled Last
Column Name            1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID             4'462       0       1     407 03.02.06 15:10
CURRENCY                   0       0       1       4 03.02.06 15:10
CUSTOMER_ID            2'441       0       1   4'462 03.02.06 15:10
OTHER_DATA                 0   4'462       1         03.02.06 15:10
STATUS                     0       0       1       4 03.02.06 15:10
SYS_NC00010$               0       0       1       4 03.02.06 15:10
TECH_INS_DATE              0       0       1       4 03.02.06 15:10
TECH_UPD_DATE              0       0       1       4 03.02.06 15:10
VALID_FROM                 0       0       1       4 03.02.06 15:10
VALID_UNTIL                0       0       1       4 03.02.06 15:10

Subpartition Column Statistics for Subpartition: ACCOUNTS_2006_A
```

```
                        num     num
                   distinct   Nulls         Sampled Last
Column Name           1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID                0       0       1         03.02.06 14:39
CURRENCY                  0       0       1         03.02.06 14:39
CUSTOMER_ID               0       0       1         03.02.06 14:39
OTHER_DATA                0       0       1         03.02.06 14:39
STATUS                    0       0       1         03.02.06 14:39
SYS_NC00010$              0       0       1         03.02.06 14:39
TECH_INS_DATE             0       0       1         03.02.06 14:39
TECH_UPD_DATE             0       0       1         03.02.06 14:39
VALID_FROM                0       0       1         03.02.06 14:39
VALID_UNTIL               0       0       1         03.02.06 14:39

Subpartition Column Statistics for Subpartition: ACCOUNTS_2006_C

                        num     num
                   distinct   Nulls         Sampled Last
Column Name           1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID              141       0       1      35 03.02.06 14:39
CURRENCY                  0       0       1       5 03.02.06 14:39
CUSTOMER_ID              98       0       1     142 03.02.06 14:39
OTHER_DATA                0     142       1         03.02.06 14:39
STATUS                    0       0       1       5 03.02.06 14:39
SYS_NC00010$              0       0       1       5 03.02.06 14:39
TECH_INS_DATE             0       0       1       5 03.02.06 14:39
TECH_UPD_DATE             0       0       1       5 03.02.06 14:39
VALID_FROM                0       0       1       5 03.02.06 14:39
VALID_UNTIL               0       0       1       5 03.02.06 14:39

Subpartition Column Statistics for Subpartition: ACCOUNTS_2006_O

                        num     num
                   distinct   Nulls         Sampled Last
Column Name           1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID            1'177       0       1     566 03.02.06 14:41
CURRENCY                  0       0       1       6 03.02.06 14:41
CUSTOMER_ID             661       0       1     566 03.02.06 14:41
OTHER_DATA                0   8'545       1         03.02.06 14:41
STATUS                    0       0       1       6 03.02.06 14:41
SYS_NC00010$              0       0       1       6 03.02.06 14:41
TECH_INS_DATE             0       0       1       6 03.02.06 14:41
TECH_UPD_DATE             0       0       1       6 03.02.06 14:41
VALID_FROM                0       0       1       6 03.02.06 14:41
VALID_UNTIL               0       0       1       6 03.02.06 14:41

Subpartition Column Statistics for Subpartition: ACCOUNTS_MAX_A

                        num     num
                   distinct   Nulls         Sampled Last
Column Name           1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID            4'458       0       1     542 03.02.06 14:51
CURRENCY                  0       0       1       6 03.02.06 14:51
CUSTOMER_ID           2'441       0       1   4'462 03.02.06 14:51
OTHER_DATA                0   4'462       1         03.02.06 14:51
STATUS                    0       0       1       6 03.02.06 14:51
SYS_NC00010$              0       0       1       6 03.02.06 14:51
TECH_INS_DATE             0       0       1       6 03.02.06 14:51
TECH_UPD_DATE             0       0       1       6 03.02.06 14:51
VALID_FROM                0       0       1       6 03.02.06 14:51
VALID_UNTIL               0       0       1       6 03.02.06 14:51

Subpartition Column Statistics for Subpartition: ACCOUNTS_MAX_C

                        num     num
                   distinct   Nulls         Sampled Last
```

```
Column Name             1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID                 0       0       1         03.02.06 14:51
CURRENCY                   0       0       1         03.02.06 14:51
CUSTOMER_ID                0       0       1         03.02.06 14:51
OTHER_DATA                 0       0       1         03.02.06 14:51
STATUS                     0       0       1         03.02.06 14:51
SYS_NC00010$               0       0       1         03.02.06 14:51
TECH_INS_DATE              0       0       1         03.02.06 14:51
TECH_UPD_DATE              0       0       1         03.02.06 14:51
VALID_FROM                 0       0       1         03.02.06 14:51
VALID_UNTIL                0       0       1         03.02.06 14:51

Subpartition Column Statistics for Subpartition: ACCOUNTS_MAX_O

                        num     num
                   distinct   Nulls         Sampled Last
Column Name           1000s   1000s Buckets   1000s Analyzed
-------------------- -------- ------- ------- ------- --------------
ACCOUNT_ID                 0       0       1         03.02.06 14:51
CURRENCY                   0       0       1         03.02.06 14:51
CUSTOMER_ID                0       0       1         03.02.06 14:51
OTHER_DATA                 0       0       1         03.02.06 14:51
STATUS                     0       0       1         03.02.06 14:51
SYS_NC00010$               0       0       1         03.02.06 14:51
TECH_INS_DATE              0       0       1         03.02.06 14:51
TECH_UPD_DATE              0       0       1         03.02.06 14:51
VALID_FROM                 0       0       1         03.02.06 14:51
VALID_UNTIL                0       0       1         03.02.06 14:51

Index Statistics for ACCOUNT

                                 leaf   dist.             avg   Clust
                                 blks    keys     avg     Data  factor
Index                Uni  Lvl 1000s   1000s LF/key blks/key   1000s
-------------------- --- ---- ----- ------- ------ -------- -------
ACCOUNT_PK           UNI    2    42  12'624      1        1  10'929
ACCOUNT_IDX1         NON    2    13       0    162    7'372     605

Partition Index Statistics for Index ACCOUNT_IDX1

                                      leaf   dist.             avg   Clust
                                      blks    keys     avg     Data  factor
Partition Name  Uni Loc Ali  Lvl 1000s   1000s LF/key blks/key   1000s
--------------- --- --- --- ---- ----- ------- ------ -------- -------
ACCOUNTS_2006   NON LOC NON    2     9       0    105    2'255     192
ACCOUNTS_MAX    NON LOC NON    2     5       0     52    4'296     378

Partition Index Statistics for Index ACCOUNT_PK

                                      leaf   dist.             avg   Clust
                                      blks    keys     avg     Data  factor
Partition Name  Uni Loc Ali  Lvl 1000s   1000s LF/key blks/key   1000s
--------------- --- --- --- ---- ----- ------- ------ -------- -------
ACCOUNTS_2006   UNI LOC NON    2    30   8'943      1        1   8'380
ACCOUNTS_MAX    UNI LOC NON    2    15   4'434      1        1   3'233

Subpartition Index Statistics for Index ACCOUNT_IDX1

                                      leaf   dist.             avg   Clust
Subpartition                          blks    keys     avg     Data  factor
Name            Uni Loc Ali  Lvl 1000s   1000s LF/key blks/key   1000s
--------------- --- --- --- ---- ----- ------- ------ -------- -------
ACCOUNTS_2006_A NON LOC NON    0     0       0      0        0       0
ACCOUNTS_2006_C NON LOC NON    1     0       0      1      464      39
ACCOUNTS_2006_O NON LOC NON    2     9       0    106    1'907     162
ACCOUNTS_MAX_A  NON LOC NON    2     5       0     56    4'620     379
ACCOUNTS_MAX_C  NON LOC NON    0     0       0      0        0       0
ACCOUNTS_MAX_O  NON LOC NON    0     0       0      0        0       0
```

```
Subpartition Index Statistics for Index ACCOUNT_PK

                                      leaf   dist.             avg   Clust
Subpartition                          blks    keys     avg     Data  factor
Name            Uni Loc Ali  Lvl 1000s   1000s LF/key blks/key   1000s
--------------- --- --- --- ---- ----- ------- ------ -------- -------
ACCOUNTS_2006_A UNI LOC NON    0     0       0      0        0       0
ACCOUNTS_2006_C UNI LOC NON    1     0     142      1        1     141
ACCOUNTS_2006_O UNI LOC NON    2    29   8'559      1        1   8'018
ACCOUNTS_MAX_A  UNI LOC NON    2    16   4'883      1        1   3'544
ACCOUNTS_MAX_C  UNI LOC NON    0     0       0      0        0       0
ACCOUNTS_MAX_O  UNI LOC NON    0     0       0      0        0       0
SQL>
```

Auf der Basis derartig umfangreicher Informationen kann man z.B. erkennen, ob ein `Rebuild` eines Index erfolgen sollte (weil z.B. die Level-Zahl zu groß ist) oder ob ein Segment mit `move` verschoben werden sollte, weil z.B. der Chain Count zu hoch ist.

Die Spalte `SYS_NC00010$`, die in den Partitions- und Subpartitionsstatistiken erscheint, ist keine Spalte der Tabelle, sondern entspricht der Spalte des Function-based Index `account_idx1`:

```
SQL> SELECT index_owner, index_name, table_owner, table_name,
  2         column_name, column_position
  3  FROM   DBA_IND_COLUMNS
  4  WHERE  index_name = 'ACCOUNT_IDX1';

INDEX_OWNER    INDEX_NAME      TABLE_OWNER    TABLE_NAME
-------------- --------------- -------------- ---------------
COLUMN_NAME               COLUMN_POSITION
------------------------- ---------------
ACC            ACCOUNT_IDX1    ACC            ACCOUNT
SYS_NC00010$                             1


SQL> SELECT index_owner, index_name, table_owner, table_name,
  2         column_expression, column_position
  3  FROM   DBA_IND_EXPRESSIONS
  4  WHERE  index_name = 'ACCOUNT_IDX1';

INDEX_OWNER    INDEX_NAME      TABLE_OWNER    TABLE_NAME
-------------- --------------- -------------- ---------------
COLUMN_EXPRESSION         COLUMN_POSITION
------------------------- ---------------
ACC            ACCOUNT_IDX1    ACC            ACCOUNT
SUBSTR("CUSTOMER_ID",1,3)                1


SQL>
```

8.4.3 Skript

Statistikdaten sind in verschiedenen Tabellen zu finden. Auf diese wird im folgenden Skript zugegriffen. Um die Liste der Spaltennamen nicht ständig zu wiederholen, werden dafür vier Variablen definiert.

Anstatt bei den partitionsweisen Spaltenstatistiken die Partitionsnamen in einer Spalte auszugeben, werden diese im Titel angezeigt. Dieses Verhalten erhält man durch den folgenden Code-Teil:

```
COLUMN partition_name new_value part_var noprint
ttitle left 'Partition Column Statistics for Partition: ' part_var skip 2
break on partition_name skip page
```

Ähnliches wurde auch bei den Subpartitions-Spaltenstatistiken und bei verschiedenen Indizes verwirklicht. Auf diese Weise ist die Ausgabe übersichtlicher.

Listing 8.5 stats.sql

```
set linesize 74 pages 200 verify off echo off feedback off termout on
set recsep off

ACCEPT tabName CHAR PROMPT "Show Statistics for Table: "
ACCEPT tabOwner CHAR PROMPT "Enter Table Owner: "

set termout off

col tabName new_value tabName
col tabOwner new_value tabOwner

SELECT UPPER('&tabName') tabName
FROM   DUAL;
SELECT UPPER('&tabOwner') tabOwner
FROM   DUAL;

ALTER SESSION SET NLS_DATE_FORMAT = 'DD.MM.YY HH24:MI'
                  NLS_NUMERIC_CHARACTERS = '.''';

set termout on

define large_num = "form 999G999G999 head"
define med_num = "form 99G999 head"
define small_num = "form 999 head"
define thousand = "1000"

define tab_stat_cols ="num_rows/&&thousand num_rows,-
blocks/&&thousand blocks,-
empty_blocks, avg_space, chain_cnt, avg_row_len,-
sample_size/&&thousand sample_size, last_analyzed"

define col_stat_cols ="column_name,-
num_distinct/&&thousand num_distinct,-
num_nulls/&&thousand num_nulls, num_buckets,-
sample_size/&&thousand sample_size, last_analyzed"

define ind_stats_cols="i.blevel, i.leaf_blocks/&&thousand leaf_blocks,-
i.distinct_keys/&&thousand distinct_keys, i.avg_leaf_blocks_per_key,-
i.avg_data_blocks_per_key,-
i.clustering_factor/&&thousand clustering_factor"

define ind_stats_cols_uni = "SUBSTR(idx.uniqueness, 1, 3) uniqueness,-
SUBSTR(pi.locality, 1, 3) locality,-
SUBSTR(pi.alignment, 1, 3) alignment,"


col num_rows &&med_num 'rows|1000s'
col blocks &&small_num 'blks|1000s'
col empty_blocks &&small_num 'emp|ty|blks'
col avg_space &&small_num 'avg|spa|ce'
col chain_cnt &&small_num 'cha|in'
col avg_row_len &&small_num 'avg|row|len|gth'
col last_analyzed form a14 head 'Last|Analyzed'
col sample_size &&med_num 'Sampled|1000s'
col partition_name form a15 head 'Partition Name'
col subpartition_name form a15 head 'Subpartition|Name'

col column_name form a20 trunc head 'Column Name'
col num_distinct &&med_num 'num|distinct|1000s'
```

```
col num_nulls &&med_num 'num|Nulls|1000s'
col num_buckets &&small_num 'Buckets'

col index_name form a20 head 'Index'
col uniqueness form a3 head 'Uni'
col blevel &&small_num  'Lvl'
col leaf_blocks &&small_num 'leaf|blks|1000s'
col distinct_keys &&med_num 'dist.|keys|1000s'
col avg_leaf_blocks_per_key  &&small_num 'avg|LF/key'
col avg_data_blocks_per_key  &&med_num 'avg|Data|blks/key'
col clustering_factor &&med_num 'Clust|factor|1000s'
col locality form a3 head 'Loc'
col alignment form a3 head 'Ali'

spool stats_&tabOwner-&tabName

ttitle left "Table Statistics for &tabName" skip 2

SELECT &&tab_stat_cols
FROM   dba_all_tables
WHERE  table_name = '&tabName'
AND    owner = '&tabOwner';

ttitle left "Partition Statistics for &tabName" skip 2

SELECT   partition_name,
         &&tab_stat_cols
FROM     dba_tab_partitions
WHERE    table_name = '&tabName'
AND      table_owner = '&tabOwner'
ORDER BY partition_name;

ttitle left "Subpartition Statistics for &tabName" skip 2

SELECT   subpartition_name,
         &&tab_stat_cols
FROM     dba_tab_subpartitions
WHERE    table_name = '&tabName'
AND      table_owner = '&tabOwner'
ORDER BY subpartition_name;

ttitle left "Column Statistics for &tabName" skip 2

SELECT   &&col_stat_cols
FROM     dba_tab_columns utc
WHERE    table_name = '&tabName'
AND      owner = '&tabOwner'
ORDER BY column_name;

COLUMN partition_name new_value part_var noprint
ttitle left 'Partition Column Statistics for Partition: ' part_var skip 2
break on partition_name skip page

SELECT   partition_name,
         &&col_stat_cols
FROM     dba_part_col_statistics
WHERE    table_name = '&tabName'
AND      owner = '&tabOwner'
ORDER BY partition_name, column_name;

COLUMN subpartition_name new_value part_var noprint
ttitle left 'Subpartition Column Statistics for Subpartition: ' part_var
skip 2
break on subpartition_name skip page

SELECT   subpartition_name,
         &&col_stat_cols
FROM     dba_subpart_col_statistics
WHERE    table_name = '&tabName'
AND      owner = '&tabOwner'
```

```
ORDER BY subpartition_name, column_name;

clear breaks
ttitle left "Index Statistics for &tabName" skip 2

SELECT i.index_name,
       SUBSTR(i.uniqueness, 1, 3) uniqueness,
       &&ind_stats_cols
FROM   dba_indexes i
WHERE  i.table_name = '&tabName'
AND    i.table_owner = '&tabOwner';

COLUMN index_name new_value idx_var noprint
ttitle left 'Partition Index Statistics for Index ' idx_var skip 2
break on index_name skip page
COLUMN partition_name print

SELECT   idx.index_name,
         i.partition_name,
         &&ind_stats_cols_uni
         &&ind_stats_cols
FROM     dba_ind_partitions i, dba_part_indexes pi, dba_indexes idx
WHERE    idx.table_name = '&tabName'
AND      idx.table_owner = '&tabOwner'
AND      i.index_name = pi.index_name
AND      i.index_name = idx.index_name
ORDER BY idx.index_name, i.partition_name;

ttitle left 'Subpartition Index Statistics for Index ' idx_var skip 2
COLUMN subpartition_name print

SELECT   idx.index_name,
         i.subpartition_name,
         &&ind_stats_cols_uni
         &&ind_stats_cols
FROM     dba_ind_subpartitions i, dba_part_indexes pi, dba_indexes idx
WHERE    idx.table_name = '&tabName'
AND      idx.table_owner = '&tabOwner'
AND      i.index_name = pi.index_name
AND      i.index_name = idx.index_name
ORDER BY idx.index_name, i.subpartition_name;

spool off
```

Es gibt viele Skripte im Internet, die z.B. anzeigen, welche Tabellen keine Statistiken haben. Diese sind nützlich, wenn man für die gesamte Datenbank herausfinden will, welche Objekte keine oder alte Statistiken haben. Mit Oracle 10g gibt es jedoch einen automatischen Job, der alle Statistiken für die gesamte Datenbank berechnet, sodass alle Objekte Statistiken haben sollten.

In sehr großen Datenbanken muss man sich jedoch selbst spezielle Maßnahmen für die Statistikberechnung überlegen, insbesondere bei Datenbanken mit regelmäßigen Ladevorgängen, da umfangreiche Statistikberechnungen mittels `dbms_stats` sehr viel Zeit und Ressourcen in Anspruch nehmen (ein Konzept dafür wurde in [Kic052] vorgestellt). Erst dort würde sich ein solches Skript eventuell auszahlen.

Während die Statistiken in Oracle 9i nur in den in *Tabelle 8.1* aufgeführten Spalten der im Skript verwendeten Dictionary Views vorhanden sind, gibt es in Oracle 10g zusätzlich die beiden Views `dba_tab_statistics` und `dba_ind_statistics`, die die Statistikinformationen für Tabellen, deren Partitionen und Subpartitionen beziehungsweise für Indizes und deren Partitionen und Subpartitionen zusammenfassen (Für die Spalten-Statistiken gibt es jedoch

keine neue View). Verwendet man (in 10g) die beiden neuen Views, kann man im obigen Skript vier Abfragen einsparen.

Häufig ist es schwierig, anhand der Oracle Datenbank-Referenz die Unterschiede oder Bedeutung der einzelnen Dictionary Views herauszufinden. Hier hilft es manchmal, im Verzeichnis `$ORACLE_HOME/rdbms/admin/` nach der View zu suchen und sich das entsprechende SQL-Skript anzusehen. Oft tragen die dort vorhandenen Kommentare zum Verständnis bei.

```
$ oracle@maschine: pwd
/oracle/product/10.2.0/rdbms/admin/
$ oracle@maschine: grep -i dba_ind_statistics * | grep -v comment
catpart.sql:create or replace view DBA_IND_STATISTICS
catpart.sql:create or replace public synonym DBA_IND_STATISTICS for
DBA_IND_STATISTICS
catpart.sql:grant select on DBA_IND_STATISTICS to select_catalog_role
e0902000.sql:DROP VIEW DBA_IND_STATISTICS;
e0902000.sql:DROP PUBLIC SYNONYM DBA_IND_STATISTICS;
```

Aufgabe:

Verändern Sie obiges Skript so, dass die neuen Views dba_tab_statistics und dba_ind_statistics statt der sechs anderen Views abgefragt werden. Das Skript würde dann nur noch für Oracle 10 funktionieren.

8.5 Automatic Workload Repository Report

Oracle generiert viele verschiedene Arten von Statistiken (System, Sessions, SQL-Statements, Segmente, Services). Wenn man Performance-Probleme analysiert, kann man für die Zeitperiode, die man untersuchen möchte, die Statistikunterschiede anschauen. Das Automatic Workload Repository in Oracle 10g macht regelmäßig Schnappschüsse sehr vieler Statistiken und speichert diese in entsprechenden Tabellen.

Auf das Automatic Workload Repository kann man über den Oracle Enterprise Manager zugreifen. Zudem gibt es viele Views, die Daten über das AWR enthalten. Oracle stellt jedoch auch entsprechende Skripte zur Verfügung, mit denen man sich Reports erzeugen lassen kann.

- `awrrpt.sql`: generiert einen HTML- oder Text-Bericht, der Statistiken für eine Reihe von Snapshot Ids anzeigt.
- `awrsqrpt.sql`: Damit lässt sich die Performance eines bestimmten SQL-Statements in einer Zeitperiode untersuchen.
- `awrddrpt.sql`: Vergleicht die Performance-Attribute und -Konfigurationen zweier auszuwählender Zeitabschnitte.

Der folgende Ausschnitt (*Abbildung 8.1*) zeigt den Inhalt eines mit `awrrpt.sql` erzeugten Reports an. Derartige Berichte sind sehr hilfreich, um eine Instanz zu tunen (z.B. Speicher-Parametereinstellungen), aber auch, um herauszufinden, welche SQL-Statements am meisten Ressourcen verbrauchen.

Abbildung 8.1 Automatic Workload Repository Report

Im Internet und in vielen anderen Büchern gibt es haufenweise Skripte, die nur Teilaspekte der Oracle-Statistiken beleuchten. Hier stellt sich die Frage, ob derartige Skripte nötig sind, da der AWR-Report alles zusammen liefert. Als Beispiel betrachten wir den Abschnitt im AWR-Report über das Buffer Pool Advisory (*Abbildung 8.2*).

Buffer Pool Advisory

- Only rows with estimated physical reads >0 are displayed
- ordered by Block Size, Buffers For Estimate

P	Size for Est (M)	Size Factor	Buffers for Estimate	Est Phys Read Factor	Estimated Physical Reads
D	1,024	0.10	64,448	1.00	11,764
D	2,048	0.20	128,896	1.00	11,764
D	3,072	0.30	193,344	1.00	11,764
D	4,096	0.40	257,792	1.00	11,764
D	5,120	0.50	322,240	1.00	11,764
D	6,144	0.60	386,688	1.00	11,764
D	7,168	0.70	451,136	1.00	11,764
D	8,192	0.80	515,584	1.00	11,764
D	9,216	0.90	580,032	1.00	11,764
D	10,240	1.00	644,480	1.00	11,764
D	11,264	1.10	708,928	1.00	11,764
D	12,288	1.20	773,376	1.00	11,764
D	13,312	1.30	837,824	1.00	11,764
D	14,336	1.40	902,272	1.00	11,764
D	15,360	1.50	966,720	1.00	11,764
D	16,384	1.60	1,031,168	1.00	11,764
D	17,408	1.70	1,095,616	1.00	11,764
D	18,432	1.80	1,160,064	1.00	11,764
D	19,456	1.90	1,224,512	1.00	11,764
D	20,480	2.00	1,288,960	1.00	11,764

Abbildung 8.2 Buffer Pool Advisory im AWR-Report

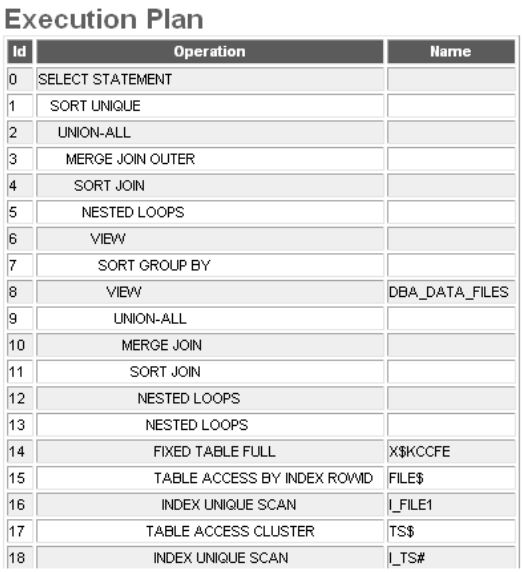
Eine ähnliche tabellarische Ausgabe kann man auch erhalten, indem man sich ein Skript schreibt, das auf die `v$db_cache_advice`-View zugreift. In Oracle 10g gibt es sehr viele dieser Advice-Tabellen.

```
SQL> @dict advice

TABLE_NAME                     COMMENTS
------------------------------ ----------------------------------------
DBA_HIST_DB_CACHE_ADVICE       DB Cache Advice History Information
DBA_HIST_JAVA_POOL_ADVICE      Java Pool Advice History
DBA_HIST_MTTR_TARGET_ADVICE    Mean-Time-To-Recover Target Advice History
DBA_HIST_PGA_TARGET_ADVICE     PGA Target Advice History
DBA_HIST_SGA_TARGET_ADVICE     SGA Target Advice History
DBA_HIST_SHARED_POOL_ADVICE    Shared Pool Advice History
DBA_HIST_STREAMS_POOL_ADVICE   Streams Pool Advice History
V$DB_CACHE_ADVICE              Synonym for V_$DB_CACHE_ADVICE
V$JAVA_POOL_ADVICE             Synonym for V_$JAVA_POOL_ADVICE
V$MTTR_TARGET_ADVICE           Synonym for V_$MTTR_TARGET_ADVICE
V$PGA_TARGET_ADVICE            Synonym for V_$PGA_TARGET_ADVICE
V$PGA_TARGET_ADVICE_HISTOGRAM  Synonym for V_$PGA_TARGET_ADVICE_HISTOGRAM
V$PX_BUFFER_ADVICE             Synonym for V_$PX_BUFFER_ADVICE
V$SGA_TARGET_ADVICE            Synonym for V_$SGA_TARGET_ADVICE
V$SHARED_POOL_ADVICE           Synonym for V_$SHARED_POOL_ADVICE
V$STREAMS_POOL_ADVICE          Synonym for V_$STREAMS_POOL_ADVICE
SQL>
```

Wozu sollte man sich jedoch die Mühe machen, für all diese Views ein eigenes Skript zu schreiben, wenn der AWR-Report das Ganze schön formatiert in HTML ausgibt?

Um mehr Informationen über ein bestimmtes SQL-Statement zu erhalten, das z.B. sehr viel CPU verbraucht, kann man nach dem AWR-Report das Skript `awrsqrpt.sql` laufen lassen. Im erzeugten Bericht sieht man neben Statistiken für das Statement (Elapsed Time, Buffer Gets, etc.) den Ausführungsplan und den gesamten SQL-Text. *Abbildung 8.3* zeigt einen Ausschnitt des Ausführungsplans eines solchen Berichts.



Execution Plan

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT UNIQUE	
2	UNION-ALL	
3	MERGE JOIN OUTER	
4	SORT JOIN	
5	NESTED LOOPS	
6	VIEW	
7	SORT GROUP BY	
8	VIEW	DBA_DATA_FILES
9	UNION-ALL	
10	MERGE JOIN	
11	SORT JOIN	
12	NESTED LOOPS	
13	NESTED LOOPS	
14	FIXED TABLE FULL	X$KCCFE
15	TABLE ACCESS BY INDEX ROWID	FILE$
16	INDEX UNIQUE SCAN	I_FILE1
17	TABLE ACCESS CLUSTER	TS$
18	INDEX UNIQUE SCAN	I_TS#

Abbildung 8.3 Ausschnitt aus einem AWR-SQL-Statement-Report

Leider werden die Informationen wie Rows, Bytes etc. nicht pro Zeile ausgegeben, so dass man nicht einfach herausfinden kann, welcher Teil des Ausführungsplans am meisten Ressourcen verbraucht. Das am Kapitelanfang vorgestellte Skript für die Anzeige des Ausführungsplans einer gerade laufenden problematischen Session liefert hingegen diese Informationen.

Mit beiden Berichten erfährt man jedoch nichts über die Größen der beteiligten Segmente, welche Indizes eine Tabelle hat, wie die Objekt-Statistiken der beteiligten Tabellen und Indizes sind. All diese Informationen sind jedoch sehr hilfreich, um den hohen Ressourcen-Bedarf eines SQL-Statements zu analysieren.

Der Vorläufer des AWR ist Statspack in Oracle 9i und funktioniert auf ähnliche Art und Weise. Statspack wird ausführlich beschrieben im Oracle9i Database Performance Tuning Guide and Reference.

8.6 Automatic Database Diagnostic Monitor

Oracle 10g bietet mit dem Automatic Database Diagnostic Monitor (ADDM) eine automatische Diagnose von Performance-Problemen. Der ADDM analysiert das Automatic Workload Repository regelmäßig, geht den Ursachen von Performance-Problemen auf den Grund und liefert Empfehlungen für die Problemkorrektur, identifiziert aber auch Felder, die keine Probleme darstellen.

Das Ziel der Analyse des ADDM ist es, eine einzige Durchsatz-Metrik, nämlich die DB Time, zu reduzieren. Diese ist die kumulative Zeit, die die Datenbank-Instanz für die Verarbeitung von Benutzer-Anfragen verbrauchte, und enthält Warte- und CPU-Zeiten.

Der ADDM hat also nicht zum Ziel, individuelle Benutzer-Antwortzeiten zu verbessern. Für diesen Zweck sollten Tracing-Techniken eingesetzt werden. Stattdessen sollen bei gleichem Ressourcen-Verbrauch mehr Benutzeranfragen abgearbeitet werden können.

Auf den ADDM kann sowohl über den Enterprise Manager als auch programmatisch über die `dbms_advisor` APIs zugegriffen werden. Oracle stellt jedoch auch ein Skript zur Verfügung, mit dem man sich einen ADDM-Bericht erzeugen lassen kann (`addmrpt.sql`). Für den eingegebenen Zeitraum werden dann die Empfehlungen ausgegeben. Die folgende Ausgabe zeigt einen solchen Bericht:

```
FINDING 1: 100% impact (76819 seconds)
-------------------------------------
SQL statements consuming significant database time were found.

   RECOMMENDATION 1: SQL Tuning, 100% benefit (76819 seconds)
      ACTION: Investigate the SQL statement with SQL_ID "c0bmgbdjx242c"
         for possible performance improvements.
         RELEVANT OBJECT: SQL statement with SQL_ID c0bmgbdjx242c and
         PLAN_HASH 3797409222
         update emp set a = 'sddf'
      RATIONALE: SQL statement with SQL_ID "c0bmgbdjx242c" was executed 4
         times and had an average elapsed time of 19204 seconds.
```

```
FINDING 2: 99% impact (76802 seconds)
-------------------------------------
SQL statements were found waiting for row lock waits.

   RECOMMENDATION 1: Application Analysis, 99% benefit (76802 seconds)
      ACTION: Significant row contention was detected in the TABLE
         "SYS.EMP"
         with object id 13331. Trace the cause of row contention in the
         application logic using the given blocked SQL.
         RELEVANT OBJECT: database object with id 13331
      RATIONALE: The SQL statement with SQL_ID "c0bmgbdjx242c" was
         blocked on row locks.
         RELEVANT OBJECT: SQL statement with SQL_ID c0bmgbdjx242c
         update emp set a = 'sddf'

   SYMPTOMS THAT LED TO THE FINDING:
      SYMPTOM: Wait class "Application" was consuming significant database
               time. (99% impact [76802 seconds])

FINDING 3: 7.5% impact (5823 seconds)
-------------------------------------
Wait event "Backup: sbtbackup" in wait class "Administrative" was consuming
significant database time.

   RECOMMENDATION 1: Application Analysis, 7.5% benefit (5823 seconds)
      ACTION: Investigate the cause for high "Backup: sbtbackup" waits.
         Refer to Oracle's "Database Reference" for the description of
         this wait event.

   RECOMMENDATION 2: Application Analysis, 7.5% benefit (5823 seconds)
      ACTION: Investigate the cause for high "Backup: sbtbackup" waits in
         Service "SYS$USERS".

   RECOMMENDATION 3: Application Analysis, 6.3% benefit (4834 seconds)
      ACTION: Investigate the cause for high "Backup: sbtbackup" waits in
         Module "backup incr datafile".

   RECOMMENDATION 4: Application Analysis, 1.3% benefit (989 seconds)
      ACTION: Investigate the cause for high "Backup: sbtbackup" waits in
         Module "backup archivelog".

   SYMPTOMS THAT LED TO THE FINDING:
      SYMPTOM: Wait class "Administrative" was consuming significant
               database time. (7.7% impact [5956 seconds])


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~

          ADDITIONAL INFORMATION
          ----------------------

Wait class "Commit" was not consuming significant database time.
Wait class "Concurrency" was not consuming significant database time.
Wait class "Configuration" was not consuming significant database time.
CPU was not a bottleneck for the instance.
Wait class "Network" was not consuming significant database time.
Wait class "User I/O" was not consuming significant database time.
Session connect and disconnect calls were not consuming significant database
time.
Hard parsing of SQL statements was not consuming significant database time.
```

8.7 Applikations-Tracing

Während AWR-Reports allgemeine Statistiken über die Oracle-Instanz liefern und Auskunft darüber erteilen, welche SQL-Statements besonders viele Ressourcen benötigen, liefert der ADDM entsprechende Empfehlungen wie „Application Analysis“ oder bei welchen SQL-Statements ein Tuning sinnvoll wäre. Mittels Application Tracing lassen sich noch viel mehr Informationen über eine bestimmte Applikation oder Session gewinnen.

Bei Oracle 9i kann das Tracing einer Session z.B. mit

```
exec sys.dbms_system.set_ev(70, 26155, 10046, 12, '');
```

ein- und mit

```
exec sys.dbms_system.set_ev(70, 26155, 10046, 0, '');
```

ausgeschaltet werden. Die beiden ersten Parameter sind hier SID und Serial# der Session, 10046 die Event-Nummer für das Tracing, und 12 und 0 geben den Trace-Level an.[1] Ab Oracle 10g sollte jedoch nur noch das `dbms_monitor` Package zum Tracing verwendet werden, so z.B. für das Tracing einer Session dbms_`monitor.session_trace_enable` und `dbms_monitor.session_trace_disable`.

Erst seit Oracle 10g wird End-to-End-Application-Tracing ermöglicht, was insbesondere in Multitier-Umgebungen von Bedeutung ist. In Multitier-Umgebungen wird eine Anfrage von einem Client auf verschiedene Datenbank-Sessions verteilt. Mittels Client Identifier wird ein spezifischer Client durch alle Tiers hindurch eindeutig identifiziert. Ebenso können spezifische Module und Aktionen innerhalb bestimmter Services, die eine Gruppe von Applikationen mit gemeinsamen Attributen und Prioritäten darstellen, verfolgt werden.

Client-Identifier-Tracing wird mit dem in Oracle 10g eingeführten `dbms_monitor`-Package z.B. wie folgt ermöglicht:

```
EXECUTE DBMS_MONITOR.CLIENT_ID_STAT_ENABLE(client_id => 'OE.OE');
```

Tracing kann dann wie folgt eingeschaltet werden:

```
EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE(client_id => 'OE.OE',
waits => TRUE, binds => FALSE);
```

Als Folge davon werden verschiedene Trace Files geschrieben. Diese können mit der von Oracle zur Verfügung gestellten `trcsess` Utility nach verschiedenen Kriterien (z.B. Client Identifier, Service) in ein einziges konsolidiert werden.

Mittels `TKPROF` lässt sich ein Trace-File formatieren und in eine lesbare Form bringen. Außerdem kann der Ausführungsplan eines SQL-Statements generiert werden.

`TKPROF` zeigt für jedes ausgeführte SQL-Statement, welche Ressourcen verbraucht wurden, wie oft es innerhalb des Beobachtungszeitraums ausgeführt wurde und die Zahl der verarbeiteten Zeilen. Auf diese Weise lassen sich diejenigen Statements lokalisieren, die am meisten Ressourcen verbrauchen.

---

[1] Es gibt auch ein Package namens dbms_support, das mit dem Skript dbmssupp.sql (im Verzeichnis $ORACLE_HOME/rdbms/admin/) installiert und fürs Tracing benutzt werden kann. Allerdings ist dieses Skript laut Kommentar „to be used only as directed by Oracle Support“.

Da in den Ausführungsplänen für jeden Schritt auch die tatsächlich benötigte Zeit und die Anzahl der Lese- und Schreiboperationen mit enthalten sind, kann man sehr schnell erkennen, wo Verbesserungspotenzial besteht.

Weitere Anhaltspunkte liefern dann die von uns vorgestellten Skripte zur Größe der beteiligten Segmente, zur Feststellung der vorhandenen Indizes und der Objektstatistiken der beteiligten Objekte. Mit diesen umfangreichen Informationen lassen sich nun z.B. das SQL-Statement anpassen, eventuell die Statistiken der Objekte verbessern oder Hints in das SQL-Statement einfügen, um einen bestimmten Ausführungsplan zu erhalten. Basierend auf der Größe der Segmente, sind z.B. auch Schlussfolgerungen über die Brauchbarkeit bestimmter Indizes möglich.