SOUG : Beitrag **Trivadis AG, Dani Schnider**
August 1999

Betrifft	**Automatisches Verwalten von Partitionen**
Art der Info	**Technische Info**
Quelle	**Aus der Praxis der Trivadis**

Automatisches Verwalten von Partitionen

Im Data Warehouse-Umfeld werden oft Fakttabellen monatsweise partitioniert. Die Fakten werden mehrere Monate historisiert und nach einer gewissen Zeit durch neue Daten überschrieben. Wie das Erstellen und Löschen der entsprechenden Partitionen automatisiert werden kann, beschreibt der folgende Artikel.

Ausgangslage

Ein Data Warehouse besteht typischerweise aus vielen kleinen Tabellen für Metadaten, Dimensionen, etc . Daneben gibt es einige wenige Fakttabellen, welche den grössten Teil des Datenbestandes ausmachen. In die Fakttabellen werden in der Regel täglich neue Daten geladen. Aufgrund des grossen Datenbestandes – ein paar hundert Gigabyte sind keine Seltenheit – werden die Fakttabellen meistens als partitionierte Tabellen implementiert. Nach einer gewissen Zeit – sagen wir nach zwei Jahren – werden die ältesten Fakten gelöscht, um Platz für die neu geladenen Daten zu schaffen. Um also eine Fakttabelle in dieser Grössenordnung zu implementieren, werden eine entsprechende Anzahl Tablespaces zur Verfügung gestellt, in welchen zyklisch Monatspartitionen angelegt werden. Das Problem besteht nun einerseits darin, jeweils rechtzeitig neue Partitionen zu erstellen, damit die neuen Daten geladen werden können, andererseits die im Data Warehouse vorhandenen Fakten nicht zu früh zu löschen, damit sie den Anwendern möglichst lange zur Verfügung stehen.

Partitionierte Tabellen

Oracle bietet seit Version 8 die Möglichkeit, Tabellen in mehrere physische Einheiten, sogenannte Partitionen, aufzuteilen. Jede einzelne Partition kann eigene physische Attribute (Tablespace, Storage-Parameter) haben und einzeln angesprochen werden. Dies ist insbesondere für die Administration und die Reorganisation von grossen Tabellen ein nicht mehr wegzudenkender Vorteil. Aus Sicht des Benutzers handelt es sich dabei weiterhin um eine einzige Tabelle. Aufgrund eines „Partition Keys“ werden die eingefügten Daten automatisch in der jeweiligen Partition gespeichert.
Im folgenden (vereinfachten) Beispiel wird eine Fakttabelle **sales** erstellt, welche nach dem Attribut **sales_date** partitioniert wird. Jeweils alle Verkäufe eines Monats werden in einer Partition abgelegt. Die Daten müssen mindestens zwei Jahre aufbewahrt werden. Wir

erstellen also eine partitionierte Tabelle mit 25 einzelnen Partitionen. Jede Partition wird in einem separaten Tablespace gespeichert.

```
CREATE TABLE sales
   (sales_date      DATE
   ,product_id      NUMBER
   ,region_id       NUMBER
   ,number_of_sales NUMBER
   ,total_amount    NUMBER)
PARTITION BY RANGE (sales_date)
   (PARTITION p_jan_1998
      VALUES LESS THAN (TO_DATE('01.02.1998','DD.MM.YYYY'))
      TABLESPACE sales_01_dta
   ,PARTITION p_feb_1998
      VALUES LESS THAN (TO_DATE('01.03.1998','DD.MM.YYYY'))
      TABLESPACE sales_02_dta
   ,PARTITION p_mar_1998
      VALUES LESS THAN (TO_DATE('01.04.1998','DD.MM.YYYY'))
      TABLESPACE sales_03_dta
   ...
   ,PARTITION p_jan_2000
      VALUES LESS THAN (TO_DATE('01.02.2000','DD.MM.YYYY'))
      TABLESPACE sales_25_dta);
```

Um auch die Indices in einzelnen Partitionen zu speichern, werden „local indexes“ erstellt. Bei Fakttabellen wird in der Regel für jede Dimension ein Index erstellt. Das nachfolgende Beispiel erstellt einen Bitmap-Index über das Attribut **product_id**, ebenfalls verteilt auf 25 einzelne Partitionen.

```
CREATE BITMAP INDEX sales_prod_idx ON sales(product_id)
LOCAL
   (PARTITION p_jan_1998
      TABLESPACE sales_01_idx
   ,PARTITION p_feb_1998
      TABLESPACE sales_02_idx
   ,PARTITION p_mar_1998
      TABLESPACE sales_03_idx
   ...
   ,PARTITION p_jan_2000
      TABLESPACE sales_25_idx);
```

Die nun erstellte Fakttabelle **sales** ist jetzt soweit vorbereitet, dass sie die Verkaufsdaten für 25 Kalendermonate speichern kann. Doch nun ist es schon bald soweit, dass die ersten Daten gelöscht werden müssen, um für den Februar 2000 Platz zu schaffen. Dazu soll zuerst die älteste Partition vom Januar 1998 gelöscht und anschliessend im freigewordenen Tablespace **sales_01_dta** eine neue Partition erstellt werden. Ganz einfach:

```
ALTER TABLE sales DROP PARTITION p_jan_1998;

ALTER TABLE sales ADD PARTITION p_feb_2000
   VALUES LESS THAN (TO_DATE('01.03.2000','DD.MM.YYYY'))
   TABLESPACE sales_01_dta;
```

Weil dabei die entsprechenden Index-Partitionen ebenfalls im Tablespace **sales_01_dta** alloziert werden, müssen sie anschliessend verschoben werden:

```
ALTER INDEX sales_prod_idx
REBUILD PARTITION p_feb_2000
```

```
   TABLESPACE sales_01_idx;
```

Dieses Prozedere muss nun jeden Monat wiederholt werden. Um den armen DBA von dieser Routinetätigkeit zu entlasten, kann das Ganze auch automatisiert werden.

Lösungsansatz für automatischen Ablauf

Anstatt jeden Monat daran denken zu müssen, die nächste Partition vorzubereiten, soll eine Prozedur **manage_sales_partition** zur Verfügung gestellt werden, die vor jedem Ladevorgang aufgerufen werden kann und die automatisch dafür sorgt, dass die entsprechenden Partitionen bereitgestellt werden. Die Prozedur wird in den täglichen Ladeablauf eingebaut. Als Parameter wird das jeweilige Ladedatum übergeben.
Die Prozedur **manage_sales_partition** führt folgende Einzelschritte durch:

- Zuerst wird das höchste Datum ermittelt, das in den bestehenden Partitionen gespeichert werden kann.
- Ist das Ladedatum grösser als dieses Enddatum, wird eine neue Monatspartition erstellt:
  - Das Enddatum der neuen Partition soll um einen Monat höher als das bisherige Enddatum sein.

Nun wird ermittelt, in welchen Tablespace die neue Partition zu liegen kommt.

  - Als nächstes wird die bestehende Partition in diesem Tablespace gelöscht.
  - Schliesslich wird die neue Partition im frei gewordenen Tablespace erstellt.

- Dieser Vorgang wird allenfalls wiederholt, bis das Ladedatum im Bereich der letzten Partition liegt.

Und so sieht das Ganze in PL/SQL aus:

```
PROCEDURE manage_sales_partition(p_load_date IN DATE)
IS
   v_last_date DATE;
   v_ts_pos    VARCHAR2(2);
BEGIN
   v_last_date := last_partition_date('SALES');
   WHILE p_load_date > v_last_date
   LOOP
      v_last_date := ADD_MONTHS(v_last_date, 1);
      v_ts_pos := tablespace_position(v_last_date);
      clean_tablespace
         (p_table_name => 'SALES'
         ,p_tablespace => ,SALES_'||v_ts_pos||'_DTA');
      add_partition
         (p_table_name      => 'SALES'
         ,p_partition_name  => 'P_'||TO_CHAR(v_last_date,
'MON_YYYY')
         ,p_high_value_date => v_last_date + 1
         ,p_dta_tablespace  => 'SALES_'||v_ts_pos||'_DTA'
         ,p_idx_tablespace  => 'SALES_'||v_ts_pos||'_IDX');
   END LOOP;
END manage_sales_partition;
```

Das war's schon fast. Vier Punkte sind allerdings noch offen und werden nachfolgend behandelt: Die Funktionen **last_partition_date** und **tablespace_position** sowie die Prozeduren **clean_tablespace** und **add_partition** müssen noch implementiert werden.

Ermitteln des höchsten Datums der letzten Partition

Um das höchste Datum zu ermitteln, das in der partitionierten Tabelle abgespeichert werden kann, wird die Data-Dictionary-View **user_tab_partitions** verwendet. Sie enthält ein Feld **high_value**, das im Falle eines Partition Keys vom Typ DATE einen Text der folgenden Art enthält:

```
TO_DATE(' 2000-02-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS',
'NLS_CALENDAR=GREGORIAN')
```

Für unsere Anwendung ist nur der Substring „2000-02-01“ relevant, der aus dem gesamten Text extrahiert werden kann. Das höchste zu speichernde Datum ist dann der Vortag dieses Datums, in diesem Beispiel also der 31. Januar 2000.
Mit diesen Angaben kann nun die Funktion **last_partition_date** implementiert werden:

```
FUNCTION last_partition_date(p_table_name IN VARCHAR2)
   RETURN DATE
IS
   v_high_value user_tab_partitions.high_value%TYPE;
BEGIN
   -- get high value string of last partition
   SELECT high_value
     INTO v_high_value
     FROM user_tab_partitions
    WHERE table_name = UPPER(p_table_name)
      AND partition_position =
         (SELECT MAX(partition_position)
             FROM user_tab_partitions
            WHERE table_name = UPPER(p_table_name));
   -- extract date from high value string
   RETURN TO_DATE(SUBSTR(v_high_value,11,10),'YYYY-MM-DD') - 1;
EXCEPTION
   WHEN OTHERS
   THEN
      RETURN TO_DATE('31.12.9999','DD.MM.YYYY');
END last_partition_date;
```

Ermitteln des richtigen Tablespaces

Die Partitionen werden wie zu Beginn beschrieben zyklisch in 25 separaten Tablespaces abgespeichert. Die Tablespaces werden mit einer zweistelligen Laufnummer versehen. Um für ein bestimmtes Datum die Laufnummer des zugehörigen Tablespaces zu ermitteln, wird eine Funktion **tablespace_position** implementiert. Die Funktion berechnet zuerst die Anzahl Monate zwischen dem übergebenen Datum und dem Startdatum. Daraus lässt sich dann die richtige Position nach folgender Formel berechnen:

Position = (Anzahl Monate) MOD 25 + 1

Die errechnete Position wird als zweistelliger Text zurückgegeben, der dann als Teil des entsprechenden Tablespace-Namens verwendet werden kann.

```
FUNCTION tablespace_position(p_date IN DATE)
   RETURN VARCHAR2
IS
   c_start_date  CONSTANT DATE   :=
TO_DATE('01.01.1998','DD.MM.YYYY');
   c_tablespaces CONSTANT NUMBER := 25;
   v_position    NUMBER;
```

```
BEGIN
   v_position := MOD(FLOOR(MONTHS_BETWEEN(p_date, c_start_date))
                    ,c_tablespaces) + 1;
   RETURN TO_CHAR(v_position, 'fm00');
END tablespace_position;
```

Zur Vereinfachung wurden das Startdatum und die Anzahl der vorhandenen Tablespaces hier als Konstanten definiert. In der Praxis werden diese Angaben natürlich auch pro Tabelle definiert und als Parameter übergeben.

Bestehende Partition löschen

Weil es sich bei den SQL-Befehlen zum Erstellen und Löschen von Partitionen um DDL-Befehle (DDL = „Data Definition Language“) handelt, können sie nicht direkt in PL/SQL verwendet werden. Deshalb benützen wir zu diesem Zweck dynamisches SQL, das ausserdem den Vorteil besitzt, dass die SQL-Befehle zur Laufzeit zusammengesetzt werden können und somit Tabellennamen, etc. als Variablen übergeben werden können. Obwohl Oracle 8i nun „Native dynamic SQL“ unterstützt, wird hier noch die „alte“ Methode unter Verwendung des Standard-Packages **dbms_sql** verwendet.
Über die Data Dictionary-View **user_tab_partitions** werden alle existierenden Partitionen der angegebenen Tabelle ermittelt, die sich in diesem Tablespace befinden. Dies sollte in der Regel genau eine Partition sein. Daraus wird dynamisch der DDL-Befehl zum Löschen der jeweiligen Partition generiert und ausgeführt. Die zugehörigen Index-Partitionen werden automatisch gelöscht und müssen nicht speziell behandelt werden.

```
PROCEDURE clean_tablespace(p_table_name IN VARCHAR2
                          ,p_tablespace IN VARCHAR2)
IS
   CURSOR tablespace_cursor IS
      SELECT partition_name
        FROM user_tab_partitions
       WHERE table_name = UPPER(p_table_name)
         AND tablespace_name = UPPER(p_tablespace);

   v_cursor INTEGER;
   v_ddl    VARCHAR2(32000);
   v_dummy  INTEGER;
BEGIN
   FOR t IN tablespace_cursor
   LOOP
      v_ddl := 'ALTER TABLE '    ||p_table_name||
               ' DROP PARTITION '||t.partition_name;
      v_cursor := dbms_sql.open_cursor;
      dbms_sql.parse(v_cursor,v_ddl,dbms_sql.native);
      v_dummy := dbms_sql.execute(v_cursor);
      dbms_sql.close_cursor(v_cursor);
   END LOOP;
END clean_tablespace;
```

Neue Partition erstellen

Nach dem gleichen Prinzip wird auch die neue Partition erstellt. Zuerst wird mit dynamischem SQL die neue Partition angefügt. Anschliessend werden nacheinander alle vorhandenen Index-Partitionen in den entsprechenden Tablespace verschoben.

```
PROCEDURE add_partition(p_table_name      IN VARCHAR2
                       ,p_partition_name  IN VARCHAR2
                       ,p_high_value_date IN DATE
                       ,p_dta_tablespace  IN VARCHAR2
                       ,p_idx_tablespace  IN VARCHAR2)
IS
   CURSOR index_cursor IS
      SELECT index_name
        FROM user_indexes
       WHERE table_name = UPPER(p_table_name);

   v_cursor INTEGER;
   v_ddl    VARCHAR2(32000);
   v_dummy  INTEGER;
BEGIN
   -- add new table partition
   v_ddl := 'ALTER TABLE '   ||p_table_name||
            ' ADD PARTITION '||p_partition_name||
            ' VALUES LESS THAN(TO_DATE('''||
               TO_CHAR(p_high_value_date,'DD.MM.YYYY')||
               ''',''DD.MM.YYYY''))'||
            ' TABLESPACE '   ||p_dta_tablespace;
   v_cursor := dbms_sql.open_cursor;
   dbms_sql.parse(v_cursor,v_ddl,dbms_sql.native);
   v_dummy := dbms_sql.execute(v_cursor);
   dbms_sql.close_cursor(v_cursor);

   -- rebuild index partitions
   FOR i IN index_cursor
   LOOP
      v_ddl := 'ALTER INDEX '       ||i.index_name||
               ' REBUILD PARTITION '||p_partition_name||
               ' TABLESPACE '       ||p_idx_tablespace;
      v_cursor := dbms_sql.open_cursor;
      dbms_sql.parse(v_cursor,v_ddl,dbms_sql.native);
      v_dummy := dbms_sql.execute(v_cursor);
      dbms_sql.close_cursor(v_cursor);
   END LOOP;
END add_partition;
```

Nun ist unsere automatische Partitionenverwaltung komplett. Die einzelnen Prozeduren und Funktionen lassen sich gut zu einem PL/SQL-Package zusammenfassen und so verallgemeinern, dass das Package für verschiedene partitionierte Tabellen verwendet werden kann.

Schlussbemerkungen

Der hier aufgezeigte Lösungsansatz zeigt, wie regelmässig anfallende Routinetätigkeiten durch eine generische Implementation automatisiert werden können. Die gezeigten Codeausschnitte zeigen jedoch nur das Prinzip. Eine in der Praxis einsetzbare Lösung ist natürlich einiges umfangreicher. So wurde in diesem Beitrag auf „Kleinigkeiten“ wie Storage-Parameter, Fehlerbehandlung und Parameterüberprüfungen weitgehend verzichtet. Automatische Abläufe wie der hier beschriebene sind zwar sehr nützlich, aber auch gefährlich. Man stelle sich vor, was passiert, wenn jemand die Prozedur

manage_sales_partition mit einem viel zu grossen Datum aufruft! Deshalb ist es sehr wichtig, dass vor dem Ausführen der DDL-Befehle Datumprüfungen und Plausibilitätschecks ausgeführt werden. Dass ein solches Package nicht an PUBLIC gegranted wird, versteht sich von selbst.

Trivadis AG
Dani Schnider
Sägereistrasse 24
8152 Glattbrugg
Internet: http://www.trivadis.com

Mail: dani.schnider@trivadis.com

Tel: +41 1 808 70 18
Fax: +41 1 808 70 21