Reference Partitioning in der Praxis

Dr. Frank Haney
Consultant
Jena

Schlüsselworte:

Oracle Database, Manageability, Partitionierung

Partitionierung von Tabellen

Die Partitionierung gehört zur physischen Datenbankimplementierung und ermöglicht eine für die Applikationen völlig transparente Aufteilung der Daten auf getrennte Bereiche, die bei Bedarf wiederum verschiedenen Tablespaces und damit Datendateien und Laufwerken zugeordnet werden können. Daraus ergeben sich vor allem bei großen Tabellen zwei wesentliche Vorteile, die so bekannt sein sollten, daß sie hier noch nicht einmal im Detail ausgebreitet werden müssen:

- **Performancegewinn durch Partition Pruning**: Bei einer Datenbankabfrage muß in Abhängigkeit vom Prädikat nicht mehr die ganze (große) Tabelle, sondern nur noch ein Teil, nämlich eine bestimmte Anzahl von Partitionen gelesen werden. Bei Joins ermöglicht die Partitionierung der beteiligten Tabellen anhand des Join-Attributs partitionsweise Joins.
- **Erleichterte Pflege**: Die Verwaltung großer Tabellen mit DML macht häufig Schwierigkeiten, weil dabei große Mengen Redo und Undo generiert werden, was wiederum die Performance beeinträchtigt. Partitionierte Tabellen können mit DDL-Befehlen verwaltet werden (Hinzufügen, Löschen, Zusammenführen und Splitten von Partitionen). Das ist wesentlich effektiver.

Oracle hat im Laufe der Zeit drei grundsätzlich unterschiedliche Partitionierungsmöglichkeiten eingeführt:

- **Range-Partitionierung** (Oracle 8): Die Daten werden entsprechend bestimmter aufeinanderfolgender Bereiche, z.B. bezüglich des Datums, also nach Tagen, Wochen, Monaten oder Jahren, aufgeteilt. Diese Form der Partitionierung macht aber nur Sinn, wenn die Aufteilung der Daten auf die Bereiche möglichst gleichmäßig erfolgt.
- **Hash-Partitionierung** (Oracle 8i): Wenn sich die Daten nicht gleichmäßig auf Bereiche aufteilen lassen, dann ist eventuell eine Aufteilung auf Hash-Partitionen sinnvoll. Das erfolgt, indem auf den Partitionierungsschlüssel eine Hash-Funktion angewendet wird. Das ist für die Abfrageperformance aber nur sinnvoll, wenn die Daten mit Gleichheitsprädikat abgefragt werden.
- **List-Partitionierung** (Oracle 9i): Der vorgesehene Partitionierungsschlüssel verteilt sich einigermaßen gleichmäßig auf eine überschaubare Menge von Werten.

Dazu gibt es diverse Erweiterungen. Dazu gehören die Index-Partitionierung und die Möglichkeit diverser Kombinationen von Sub-Partitionierung. Mit Oracle 11g sind drei wichtige Ergänzungen eingeführt worden.

- **Intervall-Partitionierung**: Dies ist eine Form der Range-Partitionierung, bei der sich der Administrator nicht mehr um das rechtzeitige Anlegen neuer Partitionen kümmern muß. Wenn z.B. im neuen Monat der erste Datensatz eingefügt wird, legt das System automatische eine neue Partition an.
- **Virtuelle Spalten** als Partitionierungsschlüssel
- **Reference-Partitionierung**: Das soll Gegenstand der folgenden Ausführungen sein.

Ausgangssituation

In einem Kundenprojekt habe ich eine Datenbank vorgefunden, die der Meßwerterfassung bei der Produktion von Solarmodulen dient. Mit Hilfe selbstgeschriebener Skripte wurden dort bestimmte Tabellen Range-partitioniert, und diese Partitionierung verwaltet. Das ganze Verfahren war ziemlich inkonsistent und fehlerträchtig. Das zeigte sich insbesondere bei Erweiterungen des Datenmodells. Dieses muß man sich in etwa so vorstellen: Neben wenigen unabhängigen Tabellen gibt es eine Mastertabelle, die gewissermaßen sternförmig von ca. 90 anderen Tabellen referenziert wird. Dabei handelte es sich sowohl um 1:n- als auch um 1:1-Beziehungen. Die Mastertabelle enthält eine ContextID für den einzelnen Meßpunkt und dazu einen Zeitstempel. Von den abhängigen Tabellen werden dieser ContextID Meßwerte zugeordnet, die im Verlauf des Produktionsprozesses entstehen. Die Referenz erfolgt also über die ContextID und nicht über den Zeitstempel.
Es wurde eine Range-Partitionierung nach ContextID vorgenommen, wobei das nicht konsequent für alle abhängigen Tabellen reproduziert wurde, was die Verwaltung nicht gerade vereinfacht hat. Entweder mußte man vor dem Löschen von Partitionen in der Master-Tabelle in den nichtpartitionierten Detail-Tabellen abhängige Datensätze mit `DELETE` entfernen oder auf Fremdschlüssel-Constraints verzichten, was die Integrität der Daten hätte beeinträchtigen können.

Die Idee war nun, die Verwaltung des ganzen Konstrukts durch eine Reference-Partitionierung zu vereinfachen, wobei sich der Zeitstempel als Partitionierungsschlüssel anbot. Performanceüberlegungen standen nicht im Vordergrund.

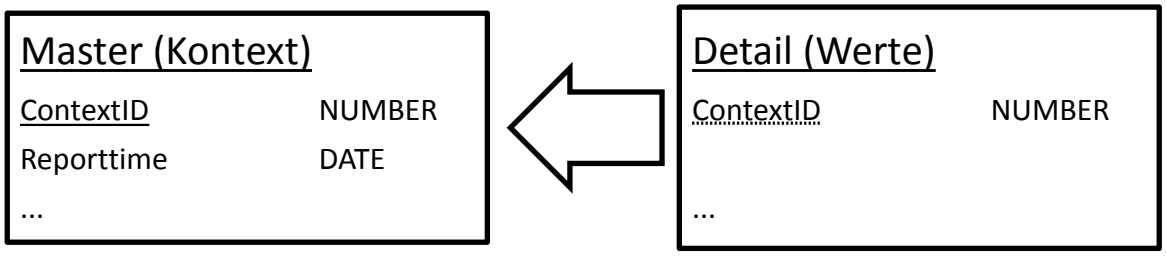

Abbildung 1: Das Datenmodell schematisch am Beispiel

Was ist Reference-Partitionierung?

Kurz gesagt, handelt es sich dabei um die Partionierung entlang von Fremdschlüsselbeziehungen. Die abhängigen Tabellen werden nach dem gleichen Kriterium wie die Mastertabelle partitioniert, ohne daß sie den Partitionierungsschlüssel explizit enthalten müssen. Im vorliegenden Beispiel bedeutet das eine Partitionierung nach der *Reporttime* und nicht nach der *ContextID*. Was sind die Vorteile dieser neuen Möglichkeit, Tabellen zu partitionieren:

- Tabellen mit Master-Detail-Beziehungen können gleich partitioniert werden, ohne daß der Partitionierungsschlüssel in der abhängigen Tabelle dupliziert werden muß.
- Wartungsmaßnahmen an der Partitionierung der Master-Tabelle werden automatisch in die abhängigen Tabellen kaskadiert, was die Wartung sehr vereinfacht und Erweiterungen des Datenmodells konsistenter und weniger fehleranfällig macht.
- Partionsweise Joins funktionieren auch, wenn das Join-Attribut nicht der Partitionierungsschlüssel ist.
- Reference-Partitionierung ist auch sinnvoll für die Partitionierung von Nested Tables.

Wie wird es nun gemacht?

Zunächst wird die Master-Tabelle angelegt:

```
  CREATE TABLE "MASTER"
  (
    "CONTEXTID" NUMBER(*,0) NOT NULL,
    "REPORTTIME" TIMESTAMP (6) NOT NULL,
    <weitere Spalten>
    CONSTRAINT "MASTER_PK" PRIMARY KEY ("CONTEXTID")
)
  PARTITION BY RANGE (REPORTTIME)
(
partition dez2012 values less than ( to_date( '1.1.2013', 'dd.mm.yyyy') ),
partition jan2013 values less than ( to_date( '1.2.2013', 'dd.mm.yyyy') ),
partition maxpart values less than (maxvalue)
);
```

Dann die abhängigen Tabellen (Beispiel):

```
  CREATE TABLE "DETAIL"
  (
    "CONTEXTID"     NUMBER(*,0) NOT NULL,
    "WAFERMATERIALID" NUMBER(*,0) NOT NULL,
<weiter Spalten>
    CONSTRAINT "DETAIL_CONTEXT_FK" FOREIGN KEY ("CONTEXTID") REFERENCES
"MASTER" ("CONTEXTID")
  )
  PARTITION BY REFERENCE ("DETAIL_CONTEXT_FK");
```

Was muß man beachten:

- Die Fremdschlüsselspalten dürfen keine NULLs enthalten und müssen explizit als NOT NULL deklariert werden.
- Damit man die Reference-Partitionierung in der Detail-Tabelle deklarieren kann, muß man die Fremdschlüssel-Constraints benennen.
- Diese Fremdschlüssel dürfen nicht ausgeschaltet (DISABLE) werden.
- Desgleichen dürfen diese Constraint nicht auf transaktionsbezogene Prüfung (DEFERRABLE) gestellt werden.
- Eine abhängige Tabelle kann Referenzen in mehrere Master-Tabellen besitzen. Für die Reference-Partitionierung kann aber nur immer eine verwendet werden.
- Eine kaskadierende Reference-Partitionierung ist aber möglich.

Im gegebenen Projekt waren folgende Fragen zu beantworten:

- Gibt es in den abhängigen Tabellen Datensätze, die NULLs in der Fremdschlüsselspalte enthalten?
  Das war der Fall. Es zeigte sich aber, daß das wenige Datensätze aus einer Teststellung waren, die gelöscht werden konnten.
- Sind alle Fremdschlüsselspalten NOT NULL deklariert?
  Das war nicht der Fall und mußte nachgeholt werden.
- Waren Fremdschlüssel-Constraints auf DEFERRABLE gestellt?
  Nein, es mußte also kein Eingriff in die Applikationslogik vorgenommen werden.
- Waren die Fremdschlüssel-Constraints benannt?
  Das war nicht der Fall. Natürlich vergibt das System beim Anlegen der Tabellen einen Namen. Es stellte sich aber als ausgesprochen unpraktisch heraus, ein vorhandenes Datenmodell nach Reference-Partitionierung zu migrieren, wenn man keine sinnvollen Constraint-Namen hat.
- Ansonsten war das Datenmodell durch seine flache Hierarchie und die nahezu ausschließliche Referenz auf eine Master-Tabelle gut für die Reference-Partitionierung geeignet.
- Auf welchem Wege konnten das Datenmodell und die Daten in die neue Struktur migriert werden?
  Dazu im folgenden Abschnitt mehr.

Migration des Datenmodells und der Daten

Grundsätzlich gibt es zwei Methoden dafür:

1. Online mit Hilfe des Package `DBMS_REDEFINITION`:
   Das war leider im vorliegenden Fall nicht möglich, wegen des vorliegenden Datenbank-Releases 11.2.0.3.0 war. Da gibt es aber infolge des Fixes für Bug 9777229 den Bug 13572659, in dessen Konsequenz während der Redefinition die Fremdschlüssel ausgeschaltet werden, was einer der Voraussetzungen für die Reference-Partitionierung widerspricht. Der Bug ist in 11.2.0.3.4 und natürlich in 12.1.0.1 gefixt. Im Vortrag wird das Verfahren exemplarisch vorgestellt. Der Ablauf ist in etwa folgender:
   - Anlegen einer Interimstabelle für die Mastertabelle (gleiche Struktur, aber neuer Name) ohne Constraints, aber mit der neuen Partitionierung nach dem Zeitstempel
   - Abschalten aller Fremdschlüssel-Constraints, die auf die Mastertabelle zeigen
   - Überprüfen, ob die Online-redefinition möglich ist mit `DBMS_REDEFINITION.CAN_REDEF_TABLE`
   - Start der Redefinition mit `DBMS_REDEFINITION.START_REDEF_TABLE`
   - Transfer der Constraints auf die Interimstabelle mit `DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS`
   - Eventuelle Synchronisation der beiden Tabellen mit `DBMS_REDEFINITION.SYNC_INTERIM_TABLE`
   - Statistiken für die neue Tabelle sammeln
   - Beenden der Redefinition mit `DBMS_REDEFINITION.FINISH_REDEF_TABLE`
   - Löschen der alten Tabelle, die jetzt den Namen der Interimstabelle hat
   - Dieser Vorgang wird für die abhängigen Tabellen wiederholt, wobei hier die Interimstabellen mit Reference-Partitioning angelegt werden.
     Noch ein Hinweis: Wenn die abhängigen Tabellen keinen Primärschlüssel haben, was beim gegebenen Datenmodell teilweise der Fall war, dann muß die Redefinition mit der ROWID-Methode erfolgen.

2. Offline durch Neuanlegen des Datenmodells mit Reference-Partitionierung und anschließendem Import der Daten aus einem aktuellen Dump. Dem kam entgegen, daß ohnehin eine größere Produktions-Downtime geplant war. Beim Import waren die Abhängigkeiten zu berücksichtigen. Er wurde im Modus `DATA_ONLY` und in mehreren Schritten durchgeführt. Mehr läßt sich eigentlich dazu nicht sagen. Im Vortrag wird es dazu ein kleines Beispiel geben.

Wartung der Partitionierung

Durch die Reference-Partitionierung erleichtert sich die Verwaltung der Tabellen enorm. Alle Wartungsmaßnahmen werden allein an der Master-Tabelle durchgeführt, weil z.B. `SPLIT PARTITION` und `DROP PARTITION` direkt an die abhängigen Tabellen propagiert werden. An der Master-Tabelle müssen diese Befehle manuell oder durch ein Skript ausgeführt werden. Im vorgestellten Projekt wird das durch einen Job realisiert, der regelmäßig läuft und am ersten jedes Monats die Max-Partition splittet und für den Monat eine neue anlegt. Durch die Reference-Partitionierung muß das jeweils nur für die Master-Tabelle gemacht werden. Zum Löschen alter Partitionen gibt es noch keine konkreten Vorstellungen.

Das war der Stand in Oracle 11g Release 2. Dort war es noch nicht möglich, Reference-Partitionierung mit **Intervall-Partitionierung** zu kombinieren. Das geht jetzt in Oracle 12c. Dazu braucht man in dem Code-Beispiel von Seite 3 nur die RANGE-Partitionierung durch die Spezialform Intervall-Partitionierung ersetzen. Etwa so:

```
  CREATE TABLE "MASTER"
  (
    "CONTEXTID" NUMBER(*,0) NOT NULL,
    "REPORTTIME" TIMESTAMP (6) NOT NULL,
    <weitere Spalten>
    CONSTRAINT "MASTER_PK" PRIMARY KEY ("CONTEXTID")
)

  PARTITION BY RANGE (REPORTTIME) INTERVAL (NUMTOYMINTERVAL(1,'month'))
(
partition dez2012 values less than ( to_date( '1.1.2013', 'dd.mm.yyyy') ),
partition jan2013 values less than ( to_date( '1.2.2013', 'dd.mm.yyyy') )
);
```

An der Definition der abhängigen Tabellen ändert sich nichts. Ein Beispiel wird im Vortrag gezeigt. Zunächst werden entsprechend dem Partitionierungsschlüssel Datensätze in die vordefinierten RANGE-Partitionen eingefügt. Wenn dieser Bereich überschritten wird, kommt die Intervall-Partitionierung ins Spiel. Immer wenn ein Datensatz eingefügt wird, der in das nächste Intervall fällt, wird automatisch vom System eine neue Partition angelegt, was dann in die abhängigen Tabellen propagiert wird.

Fazit

Mit der Reference-Partitionierung hat Oracle eine Möglichkeit eröffnet, in bestimmten Situationen die Einrichtung und Wartung der Partitionierung deutlich zu vereinfachen. Allerdings sollte das Datenmodell nicht beliebig komplex sein. Günstig ist, wenn sich die Vielzahl der abhängigen Tabellen auf eine Master-Tabelle referenziert. Dazu sind einige Voraussetzungen wie die Deklaration der Fremdschlüsselspalten als NOT NULL zu erfüllen. Mit der Version 12c vereinfacht sich das Ganze noch einmal, weil es möglich wird, Reference-Partitionierung mit Intervall-Partitionierung zu kombinieren.

Kontaktadresse: Dr. Frank Haney
Anna-Siemsen-Str. 5
D-07745 Jena

Telefon: +49(0)3641-210224
E-Mail: info@haney.it
Internet: http://www.haney.it