Reference Partitioning in der Praxis

Dr. Frank Haney, Haney IT

Oracle führt mit jedem Release neue Features ein. Der Anwender muss diese auf Sinn und Praktikabilität testen. Der Artikel beschreibt das für die mit der Version 11*g* eingeführte Funktion „Reference Partitioning".

Die Partitionierung gehört zur physischen Datenbank-Implementierung 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 erklärt werden müssen:

- *Performance-Gewinn 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 an 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 (seit Version 8)*
  Die Daten werden entsprechend bestimmter aufeinanderfolgender Bereiche, zum Beispiel bezüglich des Datums, also nach Tagen, Wochen, Monaten oder Jahren, aufgeteilt. Diese Form der Partitionierung ergibt aber nur Sinn, wenn die Aufteilung der Daten auf die Bereiche möglichst gleichmäßig erfolgt.
- *Hash-Partitionierung (seit Version 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 jedoch nur sinnvoll, wenn die Daten mit Gleichheitsprädikat abgefragt werden.
- *List-Partitionierung (seit Version 9i)*
  Der vorgesehene Partitionierungsschlüssel verteilt sich einigermaßen gleichmäßig auf eine überschaubare Menge von Werten.

Dazu gibt es diverse Erweiterungen wie die Index-Partitionierung und die Möglichkeit diverser Kombinationen von Sub-Partitionierung. Mit Version 11*g* 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 zum Beispiel im neuen Monat der erste Datensatz eingefügt wird, legt das System automatisch eine neue Partition an.
- *Virtuelle Spalten*
  Diese können als Partitionierungsschlüssel verwendet werden.
- *Reference-Partitionierung*
  Gegenstand der folgenden Ausführungen.

Ausgangssituation im Projekt

Eine Datenbank dient der Messwerterfassung bei der Produktion von Solarmodulen. Mithilfe 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

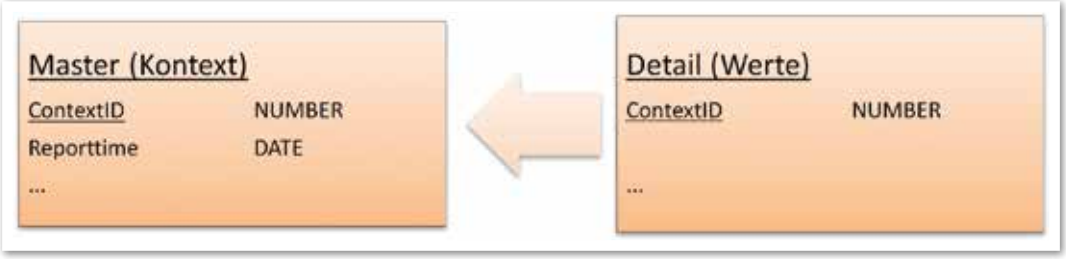

Abbildung 1: Das Datenmodell schematisch am Beispiel

so vorstellen: Neben wenigen unabhängigen Tabellen gibt es eine Master-Tabelle, die gewissermaßen sternförmig von ca. 90 anderen Tabellen referenziert wird. Dabei handelt es sich sowohl um 1:n- als auch um 1:1-Beziehungen. Die Master-Tabelle 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, allerdings wurde das nicht konsequent für alle abhängigen Tabellen reproduziert, 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 eventuell die Integrität der Daten beeinträchtigen konnte.

Außerdem mussten Partitionen auf Vorrat angelegt werden, da nur ungefähr bekannt war, wie viele ContextID-Einträge pro Zeiteinheit generiert werden. Es gab auch kein Partition Pruning für zeitbasierte Abfragen. Die Idee war nun, die Verwaltung des ganzen Konstrukts durch eine Reference-Partitionierung zu vereinfachen, wobei sich der Zeitstempel als Partitionierungsschlüssel anbot.

Reference-Partitionierung

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

- 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.
- Partitionsweise Joins funktionieren auch, wenn das Join-Attribut nicht der Partitionierungsschlüssel ist.
- Reference-Partitionierung ist auch sinnvoll für die Partitionierung von Nested Tables.

Zunächst wird die Master-Tabelle wie üblich angelegt *(siehe Listing 1)*, dann die abhängigen Tabellen *(Beispiel: Listing 2)*. Dabei ist Folgendes zu beachten:

- Die Fremdschlüsselspalten dürfen keine „NULL“ enthalten und müssen daher explizit als „NOT NULL“ deklariert sein.
- Damit man die Reference-Partitionierung in der Detail-Tabelle deklarieren kann, muss man die Fremdschlüssel-Constraints benennen.
- Diese Fremdschlüssel dürfen nicht ausgeschaltet („disable“) sein.
- Darüber hinaus dürfen diese Constraints nicht auf transaktionsbezogene Prüfung („DEFERRABLE“) gestellt sein.
- Eine abhängige Tabelle kann Referenzen in mehrere Master-Tabellen besitzen. Für die Reference-Partitionierung kann allerdings nur immer eine verwendet werden.
- Eine kaskadierende Reference-Partitionierung ist möglich.

Im Projekt waren folgende Fragen zu beantworten:

- *Gibt es in den abhängigen Tabellen Datensätze, die „NULL“ 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.

```
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 dez2013 values less than ( to_date( '1.1.2014', 'dd.
mm.yyyy') ),
partition jan2014 values less than ( to_date( '1.2.2014', 'dd.
mm.yyyy') ),
partition maxpart values less than (maxvalue));
```

Listing 1

```
CREATE TABLE detail (
contextid NUMBER(*,0) NOT NULL,
materialid NUMBER(*,0) NOT NULL,
<weiter Spalten>,
CONSTRAINT detail_context_fk FOREIGN KEY (contextid) REFERENC-
ES master (contextid))
PARTITION BY REFERENCE (detail_context_fk);
```

Listing 2

- *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. Abgesehen davon, dass es gute Praxis ist, Constraints zu benennen, ergibt sich sonst ein unlösbarer Widerspruch: Für die Reference-Partitionierung benötigt man den Namen des Constraint. Den erfährt man jedoch vom Dictionary erst nach Anlegen der Tabelle, falls man ihn nicht selber explizit vergeben hat.
- *Auf welchem Wege konnten das Datenmodell und die Daten in die neue Struktur migriert werden?*
  Dazu im folgenden Abschnitt mehr.

Ansonsten war das Datenmodell durch seine flache Hierarchie und die nahezu ausschließliche Referenz auf eine Master-Tabelle gut für die Reference-Partitionierung geeignet.

Migration des Datenmodells und der Daten

Grundsätzlich gibt es zwei Migrations-Methoden. Erstens online mithilfe des Package „DBMS_REDEFINITION". Dies war leider im vorliegenden Fall nicht möglich, weil es sich um das Datenbank-Release 11.2.0.3.0 handelt. Bei diesem gibt es jedoch infolge des Fixes für Bug 9777229 den Bug 13572659, in dessen Konsequenz während der Redefinition die Fremdschlüssel ausgeschaltet werden; das widerspricht einer der Voraussetzungen für die Reference-Partitionierung. Der Bug ist in 11.2.0.3.4 und natürlich in 12.1.0.1 gefixt. Der Ablauf ist in etwa folgender:

- Anlegen einer Interimstabelle für die Master-Tabelle (gleiche Struktur, aber neuer Name) ohne Constraints, aber mit der neuen Partitionierung nach dem Zeitstempel
- Abschalten aller Fremdschlüssel-Constraints, die auf die Master-Tabelle zeigen
- Überprüfen, ob die Online-Redefinition möglich ist mit „Dbms_Redefinition.Can_Redef_Table"
- Start der Redefinition mittels „DBMS_REDEFINITION.start_redef_table"
- Transfer der Constraints auf die Interimstabelle mit „dbms_redefinition.copy_table_dependents"
- Eventuelle Synchronisation der beiden Tabellen über „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.

Die zweite Möglichkeit der Migration ist offline durch Neuanlegen des Datenmodells mit Reference-Partitionierung und anschließendem Import der Daten aus einem aktuellen Dump. Dem kam im Projekt 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.

Wartung der Partitionierung

Durch die Reference-Partitionierung erleichtert sich die Verwaltung der Tabellen enorm. Alle Wartungsmaßnahmen werden allein an der Master-Tabelle durchgeführt, weil beispielsweise „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 Projekt wird das durch einen Job realisiert, der regelmäßig läuft, die Max-Partition splittet und für den kommenden Monat eine neue Partition anlegt. Durch die Reference-Partitionierung muss das jeweils nur für die Master-Tabelle gemacht werden. Zum Löschen alter Partitionen gibt es derzeit noch keine konkreten Vorstellungen. Das ist allerdings genauso einfach.

So viel zum Stand in der Version 11*g* R2. Dort war es noch nicht möglich, Reference-Partitionierung mit Intervall-Partitionierung zu kombinieren, was jetzt in 12*c* geht. Dazu braucht man in dem Code-Beispiel nur die RANGE-Partitionierung durch die Spezialform Intervall-Partitionierung ersetzen *(siehe Listing 3)*.

An der Definition der abhängigen Tabellen ändert sich nichts. 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; das wird dann in die abhängi-

```
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 dez2013 values less than ( to_date( '1.1.2014', 'dd.mm.yyyy')),
partition jan2014 values less than ( to_date( '1.2.2014', 'dd.mm.yyyy')));
```

Listing 3

gen Tabellen propagiert. Es gibt folgende Überlegungen zur Performance:

- Wartungsmaßnahmen wie „SPLIT PARTITION“ sollten möglichst präventiv ohne vorhandene Daten durchgeführt werden. Speziell im vorliegenden Fall, bei mehr als 90 abhängigen Tabellen in der Reference-Partitionierung, führt das sonst zu erheblichen Blockierungen (Ereignis „enq: TM – contention“). In Oracle 12*c* lässt sich das natürlich durch die Kombination mit Intervall-Partitionierung lösen. Da diese Möglichkeit in 11*g* noch nicht verfügbar ist, wurde das mit einem Job realisiert.
- Alle Operationen, die zur Bewegung von Zeilen führen, sind mit allergrößter Vorsicht und nach entsprechenden Tests zu verwenden. Das betrifft insbesondere Updates von Partitionierungsschlüsseln in der Master-Tabelle, die zu massenhaften Updates der abhängigen Tabellen führen – ungeachtet der Tatsache, dass man dazu „ROW MOVEMENT“ einschalten muss, wenn sich dadurch die Partition ändert. Solche Updates sind auch mit der Gefahr von Deadlocks verbunden.
- Das Entfernen alter Daten durch „DELETE“ sollte möglichst vermieden werden. Dafür sollten konsequent die DDL-Befehle der Partitionierung („DROP PARTITION“) genutzt werden.
- Die Fremdschlüsselspalten, über die die Reference-Partitionierung realisiert wird, sollten indiziert werden. Darüber hinaus ist die Indizierung der abhängigen Tabellen mit Bedacht vorzunehmen.

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 12*c* vereinfacht sich das Ganze noch einmal, weil es möglich wird, Reference-Partitionierung mit Intervall-Partitionierung zu kombinieren. Ergänzt sei noch, dass Reference-Partitionierung auch mit Hash- oder List-Partitionierung kombinierbar ist.

Frank Haney
info@haney.it