Im Mittelpunkt dieses Beitrags steht der effiziente Umgang mit großen Datenmengen in einem Data Warehouse-Umfeld. Basis bilden praktische Erfahrungen bei der Implementierung eines Systems mit einem Endausbau von rund fünf Milliarden Fakten, das mit Oracle Enterprise Edition auf Version 8.1.7 begonnen und über 9.2.0 bis hin zu 10.2.0 betrieben und weiterentwickelt wurde. Schwerpunkte sind die Umsetzung des Datenmodells, die Benutzung nicht alltäglicher Oracle-Features sowie Aspekte des Data Lifecycle.

Strukturelle Optimierung im Data Warehouse

Marco Mischke, Robotron Datenbank-Software GmbH

Die Erfahrungen wurden in einem Data-Warehouse-System in der Halbleiterindustrie gesammelt. Dort sind Hunderte Arbeitsschritte vom Rohmaterial bis zum Fertigprodukt erforderlich. Dabei fallen neben Stammdaten wie Prozessparametern, Maschineneinstellungen und Logistikdaten auch zahlreiche Messdaten pro Arbeitsschritt an. In diesem Fall sind 200 Maschinen mit etwa 2.000 Bewegungen über einen Zeitraum von 45 Tagen berücksichtigt. In Summe sind 4,5 Milliarden Fakten im Data Warehouse vorgehalten. Die Daten lassen sich aufgrund der großen Vielfalt nicht sinnvoll aggregieren, wie das bei einem Data Warehouse typischerweise mithilfe von OLAP Cubes etc. realisierbar wäre. Eine zweite Herausforderung ist das Data Lifecycle Management. Daten, deren Vorhaltezeit abgelaufen ist, sind aus dem System zu entfernen. Das verwendete logische Datenmodell ist dabei sehr einfach gehalten und besteht nur aus zwei Tabellen, die über eine „1:n“-Beziehung miteinander verknüpft sind (siehe Abbildung 1).

Bei der Auswertung der Daten spielt die Laufzeit der verschiedenen Reporte eine maßgebliche Rolle. Da es aufgrund der Vielzahl verschiedener Daten und verschiedenartigster Anforderungen nicht sinnvoll möglich ist, Reporte im Voraus zu berechnen, sind alle Anfragen interaktiv. Das führt zu einer maximal akzeptierten Antwortzeit von 30 bis 60 Sekunden. Alle genannten Anforderungen und deren Umsetzungen werden in den folgenden Abschnitten näher erläutert.

Partitionierung

Als erste Optimierungsmöglichkeit bot sich die Partitioning-Option an. Dafür wurden im Vorfeld typische Zugriffsmuster der Endanwender auf die gesammelten Daten analysiert. Aufgrund der zeitlichen Abfolge der Produktionsschritte und der Unterteilung in verschiedene Fertigungsbereiche haben praktisch alle Auswertungen eine Begrenzung des betrachteten Zeitbereichs und darüber hinaus eine Einschränkung bezüglich der zu berücksichtigenden Maschinenbezeichnung, die den entsprechenden Fertigungsbereich widerspiegelt.

Daraus ergab sich ein Konstrukt mit Composite-Partitioning nach Zeitstempel und Maschinenbezeichnung per Range-Hash-Partitioning. Dieses Konstrukt wurde gewählt, da dies die einzige Möglichkeit war, die Oracle in der Version 8.1.7 zur Verfügung stellte. Zukünftig kann die Partitionierung auf das in höheren Oracle-Releases

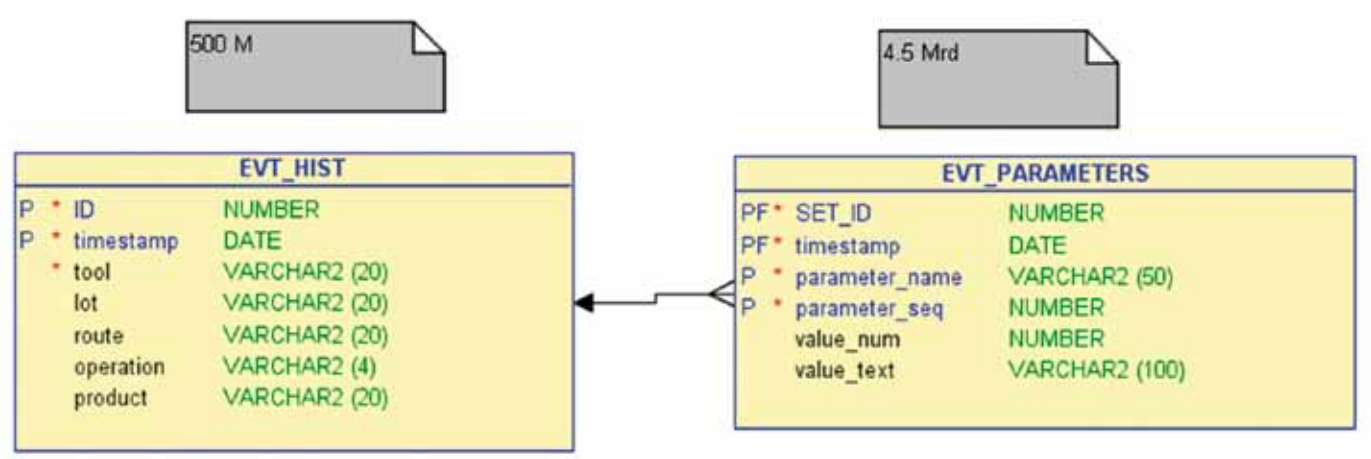

Abbildung 1: Logisches Datenmodell

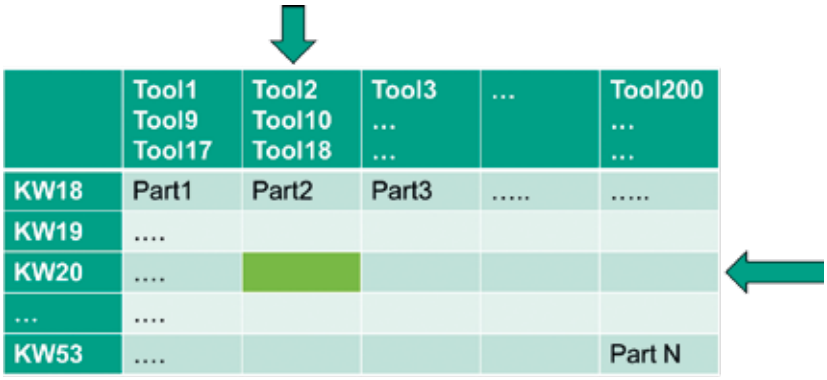

Abbildung 2: Partition-Pruning

verfügbare Range-List-Konstrukt umgestellt werden, was eine noch bessere Aufteilung der Daten über die Partitionen ermöglichen würde.

Das Range-Hash-Konstrukt ermöglicht eine Abfrageoptimierung durch Partition-Pruning. Beim Partition-Pruning werden nur noch die relevanten Partitionen einbezogen, also diejenigen, die sowohl die benötigten Zeitbereiche als auch die gewünschten Maschinen enthalten (siehe Abbildung 2). Bei typischen Abfragen werden in diesem Fall nur zwischen einer und drei Subpartitionen benutzt, das entspricht einem Anteil von rund zwei bis vier Prozent der gesamten Datenmenge (siehe Listing 1).

```
select … from EVT_HIST
where timestamp between ‚01-
Jun-2011‘
                    and ‚04-
Jun-2011‘
  and tool = ‚Tool2‘;
```

Listing 1

Die beim Partitionieren entstehende, physische Organisation der Daten vereinfacht das Data Lifecycle Management. Veraltete Daten können einfach per „drop partition“ aus dem System entfernt werden. Dadurch wird das Undo- und Redo-Aufkommen im Vergleich zu Delete-Operationen auf nahezu Null reduziert.

Weiterhin können Partitionen, in die keine Daten mehr einlaufen, reorganisiert werden, um Festplattenplatz zu sparen. Dabei werden die Daten komprimiert, was seit Oracle 9*i* für Bulk-Operationen möglich ist. Die Sortierung der Daten sorgt für eine bessere Kompressionsrate, da dadurch Duplikate eher in einem gemeinsamen Block landen und zusammengefasst werden können. Um die Reorganisation im laufenden Betrieb vornehmen zu können, wird zuerst eine strukturgleiche Zwischentabelle erzeugt (siehe Listing 2).

```
create table XXX pctfree 0 com-
press
   as select * from EVT_HIST
(partition YYY)
      order by tool, timestamp;
```

Listing 2

Diese Zwischentabelle muss dann dieselben Indizes und Constraints erhalten wie die Originaltabelle. Insbesondere Check-Constraints stellten eine Hürde dar, da diese exakt denselben Text aufweisen müssen. Ist das vorbereitet, so wird diese Zwischentabelle gegen die originale Partition ausgetauscht. Danach ist die Zwischentabelle als Partition eingebunden, während die originale Partition nun als einzelne Tabelle existiert und somit gelöscht werden kann (siehe Listing 3).

```
alter table EVT_HIST
  exchange partition YYY
  with table XXX
  including indexes without
validation;

drop table XXX;
```

Listing 3

Für das Löschen der Partitionen mussten allerdings die Fremdschlüssel deaktiviert werden, da Oracle sonst ein Löschen von Partitionen der Parent-Tabelle nicht zulässt. Durch die Reorganisation wurde eine Einsparung des benötigten Festplattenplatzes von etwa 60 Prozent erreicht. Eine aktuelle Woche umfasste rund 3 GB an Eventdaten, für weiter zurückliegende Wochen ist nach der Reorganisation hingegen nur rund 1 GB erforderlich.

Indizierung

Ein wichtiger Aspekt war die Indizierung der Tabellen. Es wurden ausschließlich lokale Indizes verwendet, da auf diese Weise kein zusätzlicher Pflegeaufwand durch DDL-Operationen entsteht. Maintenance-Operationen wie „Drop“ und „Exchange“ hinterlassen keine unbenutzbaren Indizes, sodass in der Anwendung keine Auswirkungen solcher Operationen spürbar sind.

Da die Zugriffe auf die Tabellen über viele verschiedene Attribute oder Kombinationen von Attributen erfolgen, wurden auf allen relevanten Spalten Bitmap-Indizes angelegt. Dabei wurde auch die Zeitstempel-Spalte mit einem Bitmap-Index belegt, obwohl dort die Anzahl der eindeutigen Werte entsprechend hoch war. Es ergibt sich damit die Möglichkeit, sehr viele Abfragemuster mit einer geringen Anzahl von Indizes durch Kombination der entsprechenden Bitmap-Indizes effizient zu gestalten. Da die Tabellen lediglich durch Bulk-Inserts verändert wurden, spielt das ungünstige Locking bei Bitmap-Indizes keine Rolle.

Die Speichernutzung der Bitmap-Indizes wurde durch die Option „Alter table … minimize records_per_block;“ optimiert. Die Datenbank analysiert dabei die Tabelle und ermittelt die maximale Anzahl von Datensätzen pro Block. Danach wird ein Datenblock nie mehr Datensätze enthalten als diese ermittelte Menge. Dadurch können die Bitmap-Indizes effizienter aufgebaut werden, was eine Einsparung von etwa 20 Prozent pro Index ergab.

Allerdings ergaben sich hier Probleme beim Austauschen der Partitionen während der Reorganisation. Die Größe der Bitmaps ergibt sich aus der Anzahl der Datensätze pro Datenblock. Dieses Verhältnis wird von Oracle als „Hakan-Faktor" bezeichnet. Beim Erzeugen der Zwischentabellen für die Reorganisation kann in manchen Fällen ein anderer Hakan-Faktor entstehen. Dies führt dazu, dass die erzeugte Tabelle sich dann nicht mit einer Partition der produktiven Tabelle austauschen lässt.

Index-organisierte Tabelle

Großes Optimierungspotential bestand bei den Detaildaten in der Tabelle „EVT_PARAMETERS". Diese wurde genau wie die Tabelle „EVT_HIST" nach Zeitstempel und Toolname Range-Hash-partitioniert. Der Primärschlüssel, der gleichzeitig der einzig sinnvolle Zugriffspfad ist, besteht hier aus vier Spalten. Die ganze Tabelle umfasst nur noch zwei weitere Spalten mit Nutzdaten. Dies führte zuerst zu einem Index-organisierten Aufbau der Tabelle (siehe Abbildung 3). Das sparte etwa die Hälfte des benötigten Speicherplatzes ein. Da die Daten durch die Indexstruktur vorsortiert sind, liegen gemeinsam abgefragte Daten auch physikalisch nebeneinander. So ergab sich eine I/O-Ersparnis von etwa 70 Prozent für typische Reports (Abbildung 3).

Weiterhin gibt es zu jedem Event teilweise mehrere Hunderte Parameter und entsprechend viele Duplikate von Timestamps und IDs. Die logische Konsequenz war eine Index-Prefix-Compression dieser beiden Spalten. Dabei werden mehrfach vorkommende Werte nur noch einmal gespeichert und dann referenziert. Dies führte zu einer weiteren Einsparung von etwa 30 Prozent Speicherplatz.

Single Table Hash Cluster

Die Parameternamen in der Parameter-Tabelle waren ausschließlich Texte von etwa 20 Zeichen Länge. Insgesamt existierten nur ca. 20.000 verschiedene Parameternamen. Daher wurden sie in eine Lookup-Tabelle ausgelagert und durch ein numerisches ID-Feld ersetzt. Der Parametername wurde jedoch für alle Reporte benötigt. Die Namen wurden per Join aus der Lookup-Tabelle ermittelt, wobei dieser Join transparent in einer View gekapselt wurde, sodass die Anwendung wie zuvor auf die Werte der „EVT_PARAMETERS" zugreifen konnte. Es stellte sich heraus, dass ein Index-Zugriff auf die Lookup-Tabelle mit etwa drei I/Os pro Lookup und in der Regel mehreren Tausend Lookups pro Report nicht optimal war. Die Lookup-Tabelle wurde daher als Single Table Hash Cluster aufgebaut, der entsprechend viele Schlüssel aufnehmen konnte (siehe Listing 4).

Dadurch kann mit der Hash-Funktion aus der numerischen ID direkt die Row-ID in der Lookup-Tabelle ermittelt werden. Bei jedem Lookup wird auf diese Weise nur noch ein I/O benötigt. Weiterhin verringerte sich der Storage-Bedarf der Lookup-Tabelle durch das Wegfallen von Indizes um 40 Prozent.

Fazit

Alle Optimierungen zusammen verringerten den Storage-Bedarf von anfänglich geschätzten weit über 1 TB auf etwa 300 GB. Dadurch ergaben sich eine gute I/O-Performance durch Caching-Mechanismen von Oracle und des SAN sowie eine zufriedenstellende Performance für die Ausführung von durchschnittlichen Reports. Alle Maßnahmen zusammen sorgten für eine hohe Akzeptanz beim Kunden, da insbesondere das interaktive Arbeiten mit hoher Performance maßgebend für die Zufriedenheit beim Endanwender ist.

Ausblick

Mit der Version 11.2.0 sind einige interessante Neuerungen im Bereich „Partitioning" verfügbar. So gibt es nun die Möglichkeit, nach „Range/List" zu partitionieren. Das würde eine wesentlich bessere Verteilung der Daten auf die Subpartitionen ermöglichen, als das mit dem bisherigen Range/Hash-Konstrukt der Fall ist. Weiterhin können Tabellen nun nach Reference partitioniert werden. Das bedeutet, dass eine Child-Tabelle exakt wie ihre Parent-Tabelle partitioniert wird. Dazu muss die partitionierte Spalte nicht in der Child-Tabelle vorhanden sein. Dies ermöglicht dann beispielsweise das Löschen von Partitionen aus der Parent-Tabelle, ohne die Fremdschlüssel deaktivieren zu müssen. Die zugehörige Partition der Child-Tabelle wird einfach mit gelöscht. Leider kann die Child-Tabelle dann nicht mehr Index-organisiert aufgebaut werden. Es bleibt also weiter Raum für Verbesserungen und Optimierungen.

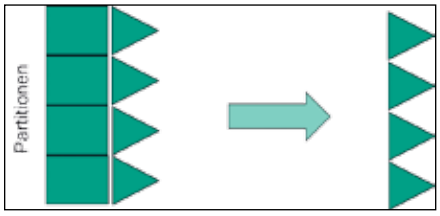

Abbildung 3: Partitionierte, Index-organisierte Tabelle

```
CREATE CLUSTER name_cluster (id
number)
  SIZE 30 SINLGE TABLE HASHKEYS
40000;

CREATE TABLE parameter_names (
  id number,
  parameter_name varchar2(50)
)
cluster name_cluster(id);
```

Listing 4

Marco Mischke
marco.mischke@robotron.de

Vorschau auf die nächste Ausgabe

Die Ausgabe 04/2012 hat das Schwerpunktthema:

„BI & Data Warehouse, Geodaten"

Sie erscheint am 17. August 2012