Praxisbericht DWH mit 5 Milliarden Fakten

Marco Mischke
Robotron Datenbank Software GmbH
Dresden

Schlüsselworte:

DWH, ETL

Einleitung

In diesem Beitrag soll der effiziente Umgang mit großen Datenmengen in einem Datawarehouse-Umfeld im Mittelpunkt stehen. Die Basis bilden praktische Erfahrungen bei der Implementierung eines Systems mit einem Endausbau von ca. 5 Mrd. Fakten, dass mit Oracle 8.1.7 begonnen wurde und über 9.2.0 bis zur Version 10.2.0 betrieben und weiterentwickelt wurde. Schwerpunkte werden in diesem Beitrag auf die Umsetzung des Datenmodels, die Benutzung nicht alltäglicher Oracle Features, die Implementierung der ETL Prozesse sowie Aspekte des täglichen Betriebs gesetzt.

Umfeld, Anforderungen, Mengengerüst und Datenmodell

Als Anwendungsbeispiel wird im Folgenden ein Datawarehouse-System in der Halbleiterindustrie betrachtet. In der Halbleiterfertigung sind hunderte Arbeitsschritte vom Rohmaterial bis zum Fertigprodukt erforderlich. Dabei fallen neben den Stammdaten wie Prozessparametern sowie Maschineneinstellungen und Logistikdaten auch zahlreiche Messdaten pro Arbeitsschritt an.
Im betrachteten Beispiel werden 200 Maschinen mit etwa 2000 Bewegungen über einen Zeitraum von 45 Tagen berücksichtigt. In Summe sind dies 4,5 Mrd. Fakten, die im DWH vorgehalten werden.. Die Daten lassen sich auf Grund der großen Vielfalt nicht sinnvoll aggregieren, wie das bei einem Datawarehouse typischerweise mit Hilfe von OLAP Cubes o.ä. realisiert werden würde. Eine zweite Herausforderung ist das Data Lifecycle Management. Daten deren Vorhaltezeit abgelaufen ist, müssen wieder aus dem System entfernt werden.
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).

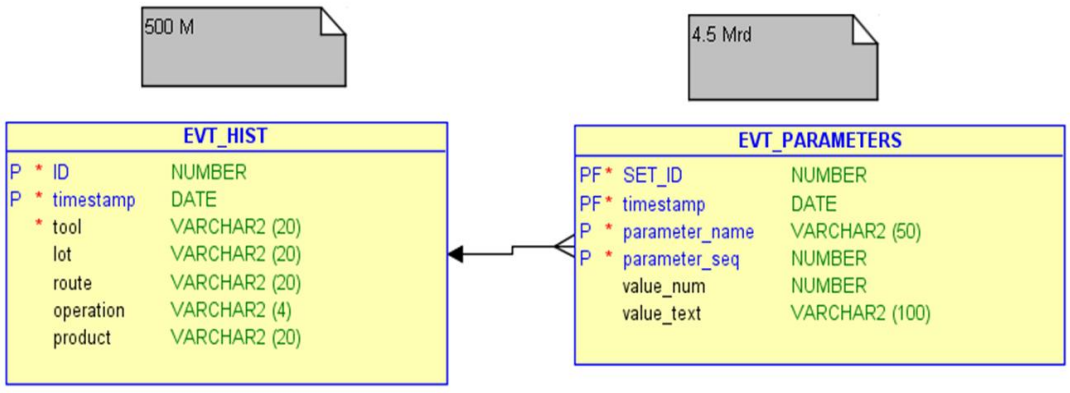

Abb. 1: logisches Datenmodell

Eine zentrale Anforderung der Endanwender ist eine kurze Latenzzeit zwischen dem Entstehen der Daten und deren Auswertbarkeit im System. Im Schnitt wurde eine maximale Verzögerung von 15 Minuten akzeptiert. Bei der Auswertung der Daten spielen wiederum die Laufzeiten der verschiedenen Reporte eine große Rolle. Da es aufgrund der Vielzahl verschiedener Daten nicht sinnvoll möglich ist, Reporte im Voraus zu berechnen, sind alle Anfragen interaktiv. Das führt zu einer maximal akzeptierten Antwortzeit von 30-60 Sekunden.
Alle genannten Anforderungen und deren Umsetzungen werden in den folgenden Abschnitten näher erläutert.

Evolution des Datenmodells und spezielle Oracle Features

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 Einschränkung bezüglich 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 8.1.7 zur Verfügung stellte. Zukünftig kann die Partitionierung auf das in höheren Oracle Releases verfügbare Range-List-Konstrukt umgestellt werden, was eine noch bessere Aufteilung der Daten über die Partitionen ermöglichen würde.
Das Range-Hash-Konstrukt ermöglicht eine Abfrageoptimierungen durch Partition-Pruning. Beim Partition- Pruning werden nur noch die relevanten Partitionen einbezogen, d.h., diejenigen die sowohl die benötigten Zeitbereiche als auch die gewünschten Maschinen enthalten. Bei typischen Abfragen werden in diesem Fall nur 1-3 Subpartitionen benutzt, das entspricht einem Anteil von ca. 2-4% der gesamten Datenmenge. Die beim Partition- Pruning 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 9i 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.

```
create table … pctfree 0 compress
   as select … from EVT_HIST
                order by tool, timestamp

alter table … exchange partition … including indexes without validation
```

Mit diesen Maßnahmen wurde eine Einsparung des benötigten Festplattenplatzes von etwa 60% erreicht. Eine aktuelle Woche umfasst ca. 3GB an Eventdaten, für weiter zurück liegende Wochen werden nach der Reorganisation hingegen nur ca. 1GB benötigt.

Ein wichtiger Aspekt war die Indizierung der Tabellen. Es wurden ausschließlich lokale Indizes verwendet da hierdurch keinerlei zusätzlicher Pflegeaufwand durch DDL-Operationen entsteht. Maintenance-Operationen wie Drop und Exchange hinterlassen keine unbenutzbaren Indizes, so dass 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 an Indizes durch Kombination der entsprechenden Bitmap-Indizes effizient zu gestalten. Die Speichernutzung der Bitmap-Indizes wurde durch die Option

```
Alter table … minimize_records_per_block
```

optimiert. Dadurch ergab sich eine Einsparung von etwa 20% pro Index. 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.

Großes Optimierungspotential bestand bei den Detaildaten in der Tabelle EVT_PARAMETERS. Der Primärschlüssel besteht hier aus vier Spalten, der gleichzeitig der einzig sinnvolle Zugriffspfad ist. Die ganze Tabelle umfasst nur noch zwei weitere Spalten mit Nutzdaten. Dies führte zuerst zu einem index-organisierten Aufbau der Tabelle. 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 etwa70% für typische Reports. Weiterhin gibt es zu jedem Event teilweise mehrere Hundert Parameter und entsprechend viele Duplikate von Timestamps und IDs. Die logische Konsequenz war eine Index-Prefix-Compression dieser beiden Spalten. Dies führt zu einer weiteren Einsparung von etwa 30% beim Speicherplatz. Die Parameternamen in der Tabelle sind ausschließlich Texte von etwa 20 Zeichen Länge. Insgesamt existierten nur ca. 20.000 verschiedene Parameternamen. Daher wurden diese Namen in eine Lookup-Tabelle ausgelagert und durch ein numerisches ID-Feld ersetzt. Der Parametername wird allerdings für alle Reporte benötigt. diese Namen werden per Join aus der Lookup Tabelle ermittelt. Dieser Join wurde transparent in einer View gekapselt so dass 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 3 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. Damit kann mit einer ID direkt die Row-ID in der Lookup Tabelle ermittelt werden. Bei jedem Lookup wird auf diese Weise nur noch 1 I/O benötigt. Weiterhin verringerte sich der Storage Bedarf der Lookup Tabelle um 40%.

Alle Optimierungen zusammen verringerten den Storage Bedarf von initial geschätzten deutlich über 1 TB auf etwa 300GB. Dadurch ergab sich eine gute I/O Performance durch Caching Mechanismen von Oracle und des SANs und eine zufrieden stellende Performance für 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.

Implementierung der ETL-Prozesse

Eine zentrale Forderung des Kunden in unserem Anwendungsbeispiel war, dass die Daten sehr zeitnah auswertbar sein sollten. Hierzu mussten die zugehörigen ETL-Prozesse entsprechend gut skalierbar gestaltet werden. Dies lässt sich nur erreichen, wenn möglichst viel parallelisiert werden kann. Die erste Stufe der Parallelisierung bestand in der Aufteilung der Ladeprozesse in verschiedene Kategorien

wie Stammdaten, unterschiedliche Typen von Bewegungsdaten usw. Daraus ergab sich bereits eine Unterteilung in sechs verschiedene Ladeprozesse, die alle unabhängig voneinander laufen können. Weitere Optimierungen fanden dann im Bereich der einzelnen Prozesse statt. Hier wurden die Prozesse in mehrere parallelisierte Stufen unterteilt, um eine datenstrombasierte Verarbeitung zu realisieren. Die zu ladenden Daten sind einem XML-ähnlichen Format und wird zunächst von einem Parser gelesen. Die zweite Stufe führt dann optional Validierungen durch. Aufgabe der dritten Stufe ist das Aktualisieren verknüpfter Stammdaten in der Datenbank und das Aufbereiten der Daten für die letzte Stufe, den SQL-Loader. Alle Stufen reichen ihre Daten über STDIN/STDOUT an die nachfolgende Stufe weiter. Dadurch werden die Zwischenergebnisse nicht nochmals als Datei oder Tabelle materialisiert sondern können von der Folgestufe bereits on-the-fly verarbeitet werden. Dargestellt ist dieser Prozess in Abbildung 2.

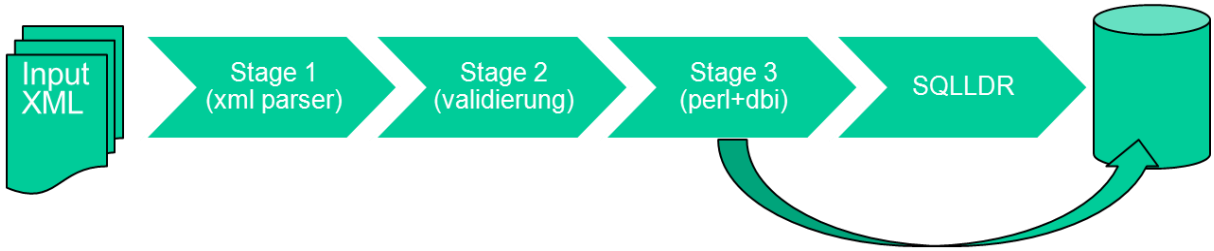

Abb. 2: parallelisierter ETL-Prozess

Für die erste Stufe zum Laden dieser Daten wurde ein SAX Parser verwendet, der im Gegensatz zu einem DOM Parser nicht den gesamten Baum im Hauptspeicher aufbaut sondern vielmehr per Callback Funktionen für entsprechende Elemente ein schrittweises Abarbeiten der Eingabedateien ermöglicht. Der zur Verfügung stehende Hauptspeicher oder die Dateigröße sind daher kein limitierender Faktor und bereits gelesene Daten können schon an die nächste Stufe weitergegeben werden, bevor die Verarbeitung der Eingabedatei abgeschlossen ist.
Die zweite Stufe ist wie auch die erste Stufe in Perl realisiert und validiert die Eingabedaten nach verschiedenen Regeln. So werden dort unter anderem Datentypen überprüft und fehlerhafte Datensätze aussortiert. Es werden nur den Regeln entsprechende Datensätze an die dritte Stufe weitergegeben.
Aufgabe der dritten Stufe ist die Aufbereitung der Daten für den abschließenden SQL-Loader. Da die Daten im Rohformat nur Textwerte enthalten, das Datenmodell aber verschiedene Lookup Tabellen verwendet, müssen diese Werte entsprechend gesondert behandelt werden. Dazu wird pro Datensatz die entsprechende ID in der Lookup-Tabelle ermittelt, sofern bereits ein Datensatz existiert. Ist der Wert neu hinzugekommen, wird dieser der Lookup-Tabelle hinzugefügt und dessen ID ermittelt. Für den SQL-Loader wird der Wert aus der Eingabe entsprechend durch die ermittelte ID ersetzt. Die Kommunikation ist mit dem DBI-Modul von Perl realisiert. Durch die Optimierung des Datenmodells und der zugehörigen Zugriffspfade ist dieses Vorgehen ausreichend performant.
Das finale Laden der Daten übernimmt der SQL-Loader, der eine statische Kontrolldatei entsprechend dem jeweils zu ladenden Datenstrom verwendet. Die Eingabedatei ist auch hier STDIN. Der Aufruf des SQL-Loaders erfolgt direkt durch die dritte Stufe aus dem Perl-Skript heraus mit einem Konstrukt dieser Art:

```
open('| sqlldr … infile=-');
```

Durch dieses Konstrukt kann man aus der Programmiersprache heraus mit traditionellen „print“-Anweisungen direkt Daten in die Datenbank schreiben, ohne sich intensiv mit SQL o.ä. beschäftigen zu müssen. Es können direkt alle Optimierungen des SQL-Loaders genutzt werden.

Ein weiterer wichtiger Mechanismus ist das Sperren der ankommenden Eingabedateien. Wenn eine Ladekette begonnen hat, eine Datei zu bearbeiten, dann wird diese Datei für alle Prozesse der gleichen Kette gesperrt. Das ermöglicht ein paralleles Arbeiten verschiedener Prozesse der gleichen Ladekette.

Fazit

Durch die Kombination verschiedener Oracle-Technologien sowie der robusten Implementierung der Ladeprozesse erreichte das System eine sehr hohe Akzeptanz beim Kunden und war darüber hinaus nahezu wartungsfrei. So konnte der Fokus auf die Implementierung verschiedener Reporte gelegt werden, die letztendlich ebenfalls dem Kunden zugutekamen.

Kontaktadresse:

Marco Mischke
Robotron Datenbank Software GmbH
Stuttgarter Straße 29
D-01189 Dresden

Telefon:	+49 (0) 25 85 9-28 84
Fax:	+49 (0) 25 85 9-36 96
E-Mail	marco.mischke@robotron.de
Internet:	www.robotron.de