Effizienz und Laufzeiten von ETL-Prozessen werden immer wieder heftig diskutiert. Die Auswirkungen von ungeschickt entworfenen Ladestrecken zeigen sich schließlich im gesamten System und sind nicht zuletzt teuer. Mehr und mehr setzt sich die Erkenntnis durch, dass kritische ETL-Prozesse am besten in der Data-Warehouse-Datenbank selbst durchgeführt werden sollten. Deshalb hier einige Hinweise zur Planung und Umsetzung von ETL-Prozessen in der Oracle-Datenbank.

ETL-Prozesse in der Oracle-Datenbank

Alfred Schlaucher, ORACLE Deutschland B.V. & Co. KG

Ein Data Warehouse ist im Gegensatz zu den meisten OLTP-Anwendungen ein datengetriebenes System. Es geht um die Organisation von statischen Informationen sowie die Modellierung von Informations-Zusammenhängen und nicht um die Modellierung von Prozessen, bei denen Datenflüsse zu regeln sind. „Funktionen zu den Daten bringen" kann daher ein wegweisendes Paradigma sein.

Die spezifischen Eigenschaften des Data Warehouse nicht unterschätzen

Ein Data Warehouse ist kein OLTP-System. Bei der Planung von ETL-Prozessen in der Datenbank ist diese Unterscheidung besonders wichtig. Tabelle 1 zeigt zusammengefasst die wichtigsten Unterschiede, die für die Planung von ETL-Prozessen bedeutsam sind.

Diese Liste lässt sich weiter fortsetzen. Doch schon jetzt kann man sagen, dass die Mehrzahl aller Änderungsoperationen in einem Data Warehouse durch „INSERTS" durchgeführt wird, die zusammenhängende Bereiche in den Datafiles beschreiben. Die erste Betrachtung gilt also der „INSERT"-Operation.

Logging/Nologging

Zunächst ist es eine bewusste Entscheidung, in einem Data Warehouse mit „Nologging" zu arbeiten. Das bedeutet, die DWH-Datenbank wird im „NOARCHIVELOG"-Modus betrieben, was zur Folge hat, dass alle Log-auslesenden Programme nicht mehr funktionieren. Entsprechende alternative Konzepte sind zu überlegen:

- *Data Guard*
  Ein DWH muss nicht repliziert werden, es ist bereits ein logisches Replikat
- *Flashback-Feature*
  Die Funktion hört sich gut an, die Wirkung ist jedoch durch einen geschickten ETL-Prozess einfacher und Ressourcen-günstiger zu haben
- *RMAN*
  Da meist nur kontrollierte Änderungen stattfinden, kann man ein Sicherungskonzept auch ohne RMAN realisieren

Bleibt der „ARCHIVELOG"-Modus dennoch aktiv, weil man beispielsweise Änderungen an den vielen kleineren DWH-Tabellen doch mit RMAN sichern will, so ist der „NOLOGGING"-Hint (/*+ NOLOGGING */) eine wichtige Einstellung in den „ETL-INSERT"- Statements. Ansonsten stört der „ARCHIVLOG"-Modus wenig, weil abfragende Benutzer keine Änderungen in den Daten hinterlassen – nur der ETL-Prozess ist betroffen.

Insert, CTAS, Direct-Path-Load

Auf dem Weg der Daten in die Datenbank-Blöcke unternimmt Oracle einige Schritte, die bei einfachen „INSERT"-Operationen in einem OLTP-System sinnvoll sind, aber bei einem schnellen Laden im Data Warehouse stören, wie das Space-Management und das Zwischenpuffern in der SGA. Diese Aktivitäten lassen sich durch den Direct-Path-Load-Hint (/*+ APPEND */) umgehen. Darüber hinaus ist die schnellste Schreib-Operation das Schreiben in eine neue und damit leere Tabelle (CREATE

OLTP	DWH
Viele einzelne, nicht kontrollierbare „INSERT"-, „UPDATE"- und „DELETE"-Operationen	Alle Änderungsvorgänge sind in der Regel durch den ETL-Prozess kontrolliert. „Man weiß, was man tut", und man kann gezielt Einfluss nehmen.
„UPDATE"- im Vergleich zu „INSERT"-Operationen relativ häufig	Bei sinnvollem Konzept kaum „UPDATE"-Operationen
Ressourcen-Auslastung relativ homogen verteilt	Ressourcen-Auslastung gliedert sich in Online- und ETL-Phasen, Datenbestände werden unterschiedlich intensiv bearbeitet
Permanent unterschiedliche Blöcke von Änderungen und Lese-Operationen betroffen	Eher zusammenhängende Blockbereiche von Lese- und Änderungs-Operationen betroffen
Die Anzahl großer und kleiner Tabellen lässt sich nicht eindeutig in Gruppen einteilen. Daher müssen gleiche Verarbeitungsregeln für alle Tabellen aufgestellt werden	Tabellen lassen sich gruppieren in eine Gruppe mit vielen kleinen Tabellen und eine Gruppe mit wenigen großen Tabellen. Davon kann man gesonderte Verarbeitungsregeln für die unterschiedlichen Gruppen ableiten.
Backup der gesamten Datenbank	Nur einzelne Bereiche von Backup betroffen

Tabelle 1

	Statement	Zeit in Min:Sek	Redo Size
1	Create Table T20 as select * from T10;	01:00	keine
2	Insert in leere Tabelle	01:46	1519380548
3	Insert in gefüllte Tabelle (Wiederholung)	01:58	1518890292
4	Insert in gefüllte Tabelle (Wiederholung)	01:59	1518890295
5	insert /*+ APPEND */ into t20 select * from t10;	01:00	777596
6	insert /*+ NOLOGGING */ into t20 select * from t10;	01:48	1519391944
7	insert /*+ APPEND NOLOGGING */ into t20 select * from t10;	01:03	777504
8	Einzel-INSERTs in Schleife einer PL/SQL-Prozedur	08:31	Nicht messbar
9	insert into t20 select * from t10 (ARCHIVLOGMODUS)	02:56	1518884743
10	(Setting „parallel_degree_policy=AUTO") insert /*+ APPEND */ into t20 select * from t10;	00:23	779124
11	(Setting „parallel_degree_policy=AUTO") Create Table T20 as select * from T10;	00:20	Nicht messbar
12	(Tabelle T20 auf SSD-Platte) insert /*+ APPEND */ into t20 select * from t10;	00:10	776998
13	(Anlegen Tabelle T20 auf SSD-Platte) Create Table T20 as select * from T10;	00:10	777476
14	(Mehrmaliges Schreiben, ohne zuvor die Buffer Caches zu leeren) insert /*+ APPEND */ into t20 select * from t10;	00:29	777596

Tabelle 2

TABLE name AS SELECT feld1, feld 2.... FROM quelle), auch „CTAS" genannt. In einer Umgebung mit „ARCHIVELOG" und bei bereits bestehenden Tabellen sollte ein „INSERT" immer mit den Hints „/*+ NOLOGGING APPEND */" gesteuert werden.

Folgende kleine Testreihe verdeutlicht anschaulich die Unterschiede (siehe Tabelle 2). Man beachte dabei auch die Verringerung der „Redo Size", mit der zusätzlich Performance-Effekte erzielbar sind. Der Test eignet sich lediglich, um grundsätzliche Trends zu erkennen, da es sich nur um eine kleine Testmaschine mit unzureichendem Storage-System handelt. Geschrieben werden 10 Millionen Sätze (ca. 1,7 GB), bei denen alle Felder zu 100 Prozent gefüllt sind. Gelesen wird aus einer Tabelle „T10" und geschrieben in eine Tabelle „T20".

Die Direct-Path-Load-Variante (APPEND, Fälle 5 und 7) ist die schnellste. Auch hinter „Create Table as select" (CTAS, Fall 1) steckt ein Direct-Path-Load. Zu erkennen ist auch dessen geringe Redo Size. Das Schreiben in eine bereits gefüllte Tabelle (Fälle 3 und 4) ist immer etwas langsamer als das Schreiben in eine komplett leere Tabelle (Fall 2).

„NOLOGGING (Fall 6) ist wesentlich schneller als das Schreiben in einer „ARCHIVELOG-MODUS"-Datenbank (Fall 9). Der Fall 8 sollte noch erwähnt werden: Er simuliert das Schreiben einzelner Sätze, also das Gegenteil einer mengenbasierten Verarbeitung mit reinem SQL.

Auch heute noch wird diese Variante in einzelnen Data-Warehouse-Systemen praktiziert. Hier wird innerhalb einer PL/SQL-Schleife ein zu schreibender Satz zunächst über Variablen zusammengesetzt und dann über einen einzelnen „INSERT" geschrieben. Dahinter steckt der Glaube, dass man zum Beispiel komplexe Prüfungen nur in dieser Weise lösen kann (Zur Lösung dieser Anforderung weiter unten mehr). Dieser Fall entspricht auch der Vorgehensweise einiger ETL-Tools, wenn diese nicht in einem Bulk-Modus arbeiten.

Parallelisierung

Das Parallelisierungs-Feature der Datenbank kann den Schreibvorgang erheblich beschleunigen. Dies setzt allerdings voraus, dass das Hardware-System genügend Platten und CPUs hat. Im Verlauf des ETL-Prozesses kann man die Parallelisierung zusätzlich gezielt und manuell steuern sowie einen höheren Parallelisierungsgrad einstellen, wenn die ETL-Jobs allein auf der Maschine laufen, also nicht gleichzeitig im Online-Betrieb gelesen wird.

Ab dem Datenbank-Release 11.2.0.2 sollte man neben der manuellen Steuerung unbedingt den Parameter „parallel_degree_policy" mit dem Setting „AUTO" testen, da dieser Parameter auch den jeweiligen ETL-Jobs die optimale Parallelisierung in Abhängigkeit von Hardware-Ressourcen und Auslastung zuweist. Mit diesem Parameter wurden in dem Test oben auch beim Direct-Path-Load die Ladezeiten noch einmal optimiert (Fälle 10 und 11).

SSD und andere schnelle Datenträger

Weiter unten im Artikel wird die Verwendung temporärer Tabellen vorgestellt. In der Praxis stellen diese ein sehr wichtiges Hilfsmittel bei der Strukturierung des Gesamtprozesses dar. Einige Informationen können im Verlauf des Ladeprozesses mehrfach gelesen und geschrieben werden, bis sie an ihren endgültigen Bestimmungsort gelangen. Wenn solche Tabellen nicht permanent im Hauptspeicher gehalten werden können, kann man dennoch versuchen, sie auf schnelle Datenträger wie SSDs oder Flashspeicher zu legen, auch wenn die Masse der Data-Warehouse-Daten noch auf klassischen Spindel-Datenträgern liegt. Im Testfall oben hat die Verwendung von nur einer SSD-Platte zu einer Reduzierung der Ladezeit auf 10 Sekunden geführt (Fälle 12 und 13).

DWH-Prüfungen und Umgang mit Constraints in der Datenbank

Alle notwendigen Plausibilitätsprüfungen sollten im Integration-Layer (Stage) des Data Warehouse erfolgen. Diese Prüfungen gehören zu den aufwändigsten Schritten in jedem ETL-Prozess. Datenbank-Constraints bieten sich für einige Prüfungen an. Doch diese verlangsamen ETL-Läufe erheblich. Anders als in OLTP-Systemen laufen Änderungsvorgänge im Data Warehouse kontrolliert ab. Im Rahmen des ETL-Prozesses kennt man jede Änderung. Daraus folgt, dass Constraints, die gegen unkontrolliertes Ändern schützen, nicht notwendig sind.

Die Konsistenz der neu zu ladenden Daten sollte dagegen mengenbasiert mit SQL in der Datenbank gesichert werden. Für die Planung solcher Prüfungen muss man die Art der Prüfung feststellen. Tabelle 3 zeigt die Kategorisierung aller durchzuführenden Prüf-Aktivitäten.

Für jede dieser Kategorien lassen sich standardisierte Wege mit SQL entwickeln. Anmerkung: Im Seminar „ETL in der Datenbank" des Autors werden Lösungen für die jeweiligen Kategorien vorgestellt (siehe Oracle-DWH-Community-Webseite www.oracledwh.de). Alles, was es in einem Data Warehouse zu prüfen gibt, lässt sich ohne Constraints und ohne prozedurale Programmierung mit schnellen, mengenorientierten „INSERT"-Operationen lösen.

Hier gibt es nur den einen Ausnahmefall: die Komplexität. Wird ein SQL-Statement zu komplex, kann man es in prozedurale PL/SQL-Logik umwandeln. Hierzu bieten sich dann parallelisierbare Table-Functions an, die hier nicht weiter betrachtet werden (siehe auch hierzu das Seminar „ETL in der Datenbank").

Kategorie	Beschreibung
Attribut-/Column-bezogene Regeln	Feldformate, Not Null, Maskenformate, Wertebereiche, Ober-, Untergrenzen, Listen
Satzbezogene Regeln	Abhängigkeiten zwischen Werten von Columns desselben Satzes, funktionale Abhängigkeiten
Satzübergreifende Regeln	Eindeutigkeit einer Tabelle (Primary Key), Intervalle, Satzgruppen, Gruppenwechsel, rekursive Strukturen, Aggregatprüfungen
Tabellenübergreifende Regeln	Child-/Parent (Orphan) / Parent-Child (Childless), Kardinalitäten, Aggregatbildungen
Zeit-/Zusammenhang-bezogene Regeln	Zeitintervalle, geographische, politische, soziale, organisatorische Fakten
Verteilungs-/Mengenbezogene Regeln	Durchschnittsbildung, Varianzen, Verteilungen

Tabelle 3

Das Arbeiten mit Zwischen-Tabellen

Um der potenziellen Komplexität von prüfenden SQL-Statements zu begegnen, sollte man temporäre Tabellen nutzen. Hier kann man Zwischen-Ergebnisse von Prüfungen ablegen. Die Tests oben haben gezeigt, dass das Anlegen und Beschreiben von neuen beziehungsweise leeren Tabellen zu den schnellen Vorgängen gehört. Um die Zeit der Einzelsatzverarbeitung in Fall 8 zu erreichen, kann man bequem die gleiche Datenmenge schon in bis zu acht temporären Zwischen-Tabellen speichern. In der Regel reichen schon eine bis drei Zwischen-Tabellen.

Aktivitäten bündeln und die Caching-Funktion der Datenbank nutzen

Ein großer Teil des Aufwands bei den oben gezeigten Lade-Operationen stellt der „SELECT"-Teil der Befehle dar. Über den Befehl „SELECT COUNT(*) FROM T10" (Full Table Scan) misst man 27,47 Sekunden, die in allen Testfällen gleichermaßen stecken. Über eine Gesamtplanung aller ETL-Schritte sind genau solche lang laufenden „SELECT"-Phasen zu identifizieren und so zusammenzufassen, dass sie nach Möglichkeit nur einmal ausgeführt werden müssen oder ein Nachladen bei mehrmaligem Lesen durch das automatische Cachen der Datenbank nicht nötig ist.

In dem oben gezeigten Test wurde vor jedem Aufruf der Buffer-Cache geleert (ALTER SYSTEM FLUSH BUFFER_CACHE). Macht man das nicht, so halbieren sich zumindest für „DIRECT PATH"-Läufe die Zeiten. Das bedeutet, dass man aufwändige Lese-Operationen aus den gleichen Tabellen in zeitlich unmittelbarer Abfolge bündeln sollte, weil damit eine große Chance besteht, dass unmittelbar zuvor gelesene Tabellen sich noch im Hauptspeicher der Datenbank befinden (Fall 14).

Das zusätzliche Cachen bestimmter Tabellen (ALTER TABLE name CACHE) kann nützlich sein. Es behindert jedoch die Automatik, mit der Oracle häufig genutzte Tabellen sowieso im Hauptspeicher vorhält, weil es die für diese Dynamik bereitstehende Speichergröße verkleinert.

Umgang mit aufwändigen Join-Operationen

Müssen mehrmals im Verlauf des gesamten ETL-Prozesses Daten aus einem aufwändigen Join gelesen werden, dann kann eine einmal erstellte temporäre Join-Tabelle mit den wichtigsten Join- und Abfragekriterien helfen. Eine solche Tabelle besteht oft nur aus wenigen Spalten und umfasst nur einen Bruchteil des Datenvolumens des kompletten Joins. Das System kann eine solche kleine Tabelle besser im Hauptspeicher aufbewahren, also automatisch cachen. Kleine Join-Tabellen helfen bei immer wiederkehrenden Lookup-, Childless- und Orphan-Prüfungen, also immer dann, wenn Beziehungen im Spiel sind.

Die Reihenfolge der Prüfungen

Die oben aufgelisteten Prüf-Kategorien sollten in einer bestimmten Reihenfolge abgearbeitet werden, denn über eine geschickte Abfolge kann man die Gesamtlaufzeit ebenfalls minimieren:

1. Zunächst sind Formatprüfungen auf ihre Notwendigkeit hin zu überprüfen: Ist die Datenquelle eine Datenbank, in der Daten bereits formatgerecht abgelegt sind, dann muss man solche Format-Prüfungen in der Data-Warehouse-Datenbank nicht mehr wiederholen.
2. Daten aus Textdateien müssen dagegen geprüft werden. Dabei sollte man die Prüflogik der External Tables beziehungsweise des SQL-Loader nutzen. Man prüft also bereits, bevor die Daten in die Datenbank gelangen. Prüfungen des SQL-Loader beziehungsweise der External Tables sind schneller als Formatprüfungen von Sätzen, die bereits in der Datenbank gespeichert sind, zumal die Datenbank keine „is_Date"- oder „is_Numeric"-Funktion kennt. Solche Funktionen muss man selbst schreiben und sie als „Einzelsatz-Select" aufrufen, sodass die Prüfung in der Datenbank aufwändig werden kann.
3. Innerhalb der Datenbank sind als Erstes immer Beziehungsabhängigkeiten, also Orphan, Childless bezie-

hungsweise satzübergreifende Zusammenhänge wie Eindeutigkeiten zu prüfen. Dies ist meist mengenbasiert durchführbar. Sätze, die diesen als schnell eingestuften Prüfungsschritt nicht passieren, belasten auch nicht mehr die nachfolgenden langsameren Einzelfeld-Prüfungen.

4. Als Letztes sind Einzelfeld-Prüfungen, Format-Prüfungen etc. durchzuführen. Solche Prüfungen sind oft nur als Einzelsatzverarbeitung mit Funktionsaufrufen oder „CASE"-Strukturen möglich. Von solchen Einzelfeld-Prüfungen sollte man möglichst viele in einem einzigen Verarbeitungsschritt bündeln. Denn man ist bereits in der Einzelsatzverarbeitung und in dieser sollte man Sätze nur einmal anfassen.

Prüfen mit Error-Log-Tabellen

Seit den jüngsten Releases der Oracle-Datenbank kann man sogenannte „Error-Log-Tabellen" zusätzlich zu den Ziel-Tabellen definieren. Darin sammelt man fehlerhafte Sätze, die einen aktivierten Constraint verletzt haben. Error-Log-Tabellen können bei kleinen Tabellen hilfreich sein. Sie bremsen jedoch bei Massendaten, weil bei jeder Constraint-Verletzung ein Insert in die Fehler-Tabelle erfolgen muss. Die Constraint-Prüfung an sich verhindert darüber hinaus den Direct-Path-Load (außer „Not Null" und „Unique Key Constraints").

Umgang mit sehr großen Tabellen

In den meisten Fällen finden wir in einem Data Warehouse wenige Tabellen, die sehr groß sind (Bewegungsdaten- und Fakten-Tabellen), während die Masse der Tabellen klein ist (Stammdaten- und Referenz-Tabellen). Große Tabellen sind auch oft diejenigen mit dem größten Ladeanteil während des ETL-Prozesses. Für diese lohnt sich daher auch eine besondere Behandlung. Will man die oben beschriebenen Performance-Effekte nutzen, so sollte man neue Daten für große Tabellen zunächst in neu erstellte temporäre Tabellen schreiben. „Create Table As Select" (CTAS) beziehungsweise das Schreiben in leere Tabellen mit dem Direct-Path-Load sind, wie gesagt, die schnellsten Schreibvorgänge. Diese temporären Tabellen können dann an die eigentliche Ziel-Tabelle über das „Partition Exchange and Load Feature" (PEL) als weitere Partition angeschlossen werden. Dies zeigt, dass die Partitionierung großer Tabellen im Data Warehouse nicht nur für Performance-Effekte während des Lesens wichtig ist, sondern auch für die Beschleunigung des ETL-Prozesses.

Physische Struktur und Eigenschaften der DWH-Tabellen

Data-Warehouse-Tabellen erfahren in der Regel keine „UPDATE"-Operationen. Daher sollten sie grundsätzlich mit „PCTFREE=0" angelegt sein. Es werden also nach Möglichkeit alle Blöcke zu 100 Prozent vollgeschrieben. Das minimiert die Schreibzugriffe und bei nachfolgenden Lesevorgängen die physikalischen Lesezugriffe auf die Speicherplatte um bis zu 10 Prozent. Die Tabellen im Beispiel weiter oben haben mit „PCTFREE=0" anstatt 190.000 nur noch 170.000 Blöcke und anstatt 1,7 GB nur noch 1,46 GB Volumen.

Index-free

Man sollte das Schreiben in eine Tabelle mit aktiviertem Index vermeiden, weil der Index immer aktuell gehalten wird. Man löscht also den Index vor einem Massen-Load beziehungsweise setzt ihn auf „UNUSABLE", um ihn nach dem Laden entweder neu anzulegen oder mit „REBUILD" zu aktualisieren (ALTER INDEX name REBUILD). Generell sollte man die Notwendigkeit von Indizes im Data Warehouse überprüfen.

Temporäre Tabellen im Integration-Layer (Stage) und die meisten Tabellen im Enterprise-Layer (Data-Warehouse-Kern-Schicht) haben keine Index-Definitionen. Im Enterprise-Layer verfügen nur Stammdaten- und Referenzdaten-Tabellen über einen „Btree"-Index, der nach Abschluss des Ladevorgangs zu aktualisieren ist. Große Bewegungsdaten-Tabellen brauchen normalerweise keinen Index. Im User-View-Layer (Data Marts) gibt es auf den Primary-Key-Feldern der Dimensionen „Btree"-Indizes, auf den übrigen Feldern „Bitmap"-Indizes und auf den „FK"-Feldern der Fakten-Tabellen „Bitmap"-Indizes, die allerdings im Rahmen des Partition Exchange Load (PEL) sehr schnell aktualisiert werden können.

Kennzahlen und Aggregat-Tabellen

Neben Fakten-Tabellen gibt es häufig auch noch fest strukturierte Kennzahlen- oder Aggregat-Tabellen im User-View-Layer beziehungsweise in den Data Marts. Diese Tabellen enthalten über bereits bekannte Algorithmen erstellte Berichtsdaten. Solche Aggregat-Tabellen sollte man nicht mit ETL-Prozeduren oder mit ETL-Tool-Mappings herstellen. Dafür gibt es die Materialized Views in der Datenbank. Diese können sich bei Bedarf selbst aktualisieren und nutzen gegebenenfalls auch das Partition-Change-Tracking (PCT), mit dem nur Deltadaten aktualisiert werden müssen. Das ist meist schneller und spart ETL-Verwaltungsaufwand.

Das Schichtenmodell hilft bei der Planung von ETL-Prozessen

Die vorgenannten Schritte zeigen bereits auf, dass eine Gesamtübersicht über alle Lade- und Transformations-Prozesse im Data Warehouse sehr wichtig ist. Lade- und Prüfschritte an irgendwelchen beliebigen Stellen durchzuführen, kann die Gesamtladezeit um ein Vielfaches verlängern.

Das Schichtenmodell strukturiert den Informationsbeschaffungsprozess in einem Data Warehouse in folgende Phasen:

- Prüfen/harmonisieren (Stage- oder Integration-Layer)
- Übergreifend integrieren (Warehouse-Schicht oder Enterprise-Layer)
- Sachgebietsbezogen zusammenfassen (Data Mart oder User-View-Layer)

Zusammen mit den Datenmodellen der Enterprise- und User-View-Layer sollte deutlich sein, an welcher Stelle eine entsprechende Information benötigt wird, und von wo sie zu beziehen ist. Das nutzt man und konzentriert alle prüfenden Vorgänge im Integration-Layer und noch davor. Man sucht also für die Platzierung von Transformationen und Prüfungen die frühestmögliche Stelle im Gesamtprozess, um mög-

lichst viele nachfolgende Stellen davon profitieren zu lassen. Wer erst beim Wechsel in den User-View-Layer prüft, der provoziert potenziell doppelte und inkonsistente Prüfungen. Auf dem Weg in den User-View-Layer sollte es nur noch zusammenführende (Joins und) Lookup-Operationen geben.

Günstige Rahmenbedingungen für den ETL-Prozess schaffen

Die genannte Strukturierung gelingt nur, wenn sich alle Schichten des Data Warehouse in einer Datenbank befinden. Das Auslagern von Data Marts auf separate Maschinen verhindert die gemeinsame Nutzung von sehr großen Tabellen, die man am geschicktesten im Enterprise-Layer belässt und aus den Data Marts heraus referenziert. Auch das spart aufwändige Ladeprozesse.

Die Verwendung von separaten ETL-Servern (oft bei einem Einsatz von ETL-Tools üblich) zersplittert den Ladeprozess. Das Zusammenfassen temporärer Tabellen wird erschwert und die Speicherausnutzung der Datenbank verhindert; mengenbasierte Prüfungen in der Datenbank sind unmöglich.

ETL-Tools

Der Einsatz von ETL-Tools, die außerhalb der Datenbank arbeiten, führt fast immer zu langsameren Ladeprozessen. ETL-Tools haben gegenüber dem Laden in der Datenbank nur einen Vorteil: Sie bieten eine übersichtlichere Dokumentation und damit unter Umständen eine schnellere Implementierung und Wartung. Als Kompromiss wird man aufwändige Lade-Aktivitäten in die Datenbank legen und diese von außen über das jeweilige ETL-Tool steuern. Dieser Weg wird bei fast allen größeren Data-Warehouse-Umgebungen mit performancekritischen Ladeprozessen beschritten.

Alfred Schlaucher
alfred.schlaucher@oracle.com