SQL als ETL Tool

Christian König
CGI (Germany) GmbH & Co. KG
Düsseldorf

Schlüsselworte
SQL, ELT, ETL, Oracle

Einleitung

In jedem Data Warehouse gehört die Entscheidung, welches ETL – Tool für die Datenbewirtschaftung verwendet wird, mit zu den wichtigsten technischen Entscheidungen einer DWH-Strategie. Dieses Thema ist vor allem Aufgrund des Support-Endes für den Oracle Warehouse Builder (OWB) besonders aktuell. Viele OWB-Nutzer evaluieren die Migration von OWB nach ODI (Oracle Data Integrator), aber auch zu anderen ETL-Tools wie Informatica.
Aber geht es nicht auch ganz ohne ETL-Tool? Man kann auch mit Oracle-Datenbank-Bordmitteln ein Data Warehouse sehr gut beladen und betreuen. Eine Umfassende Gegenüberstellung aller Vor- und Nachteile von ETL-Tools gegenüber „handgeschriebener" ETL-Strecken würde den Rahmen dieser Veranstaltung sprengen. Deswegen werde ich, um meine These zu untermauern, beispielhaft 3 Oracle-DB-Technologien betrachten, und deren Vorteile gegenüber ETL-Tools herausstellen:

Meta-Programming

Um Daten von den Quellsystemen in die Staging Area zu laden, gibt es verschiedene Möglichkeiten. Daten können z.B. als Datei geliefert und mit Oracle External Tables eingelesen werden, oder Tabellen werden importiert und wieder exportiert, oder die Quelldatenbanken werden direkt per Database Link verbunden.
In diesem Beispiel nehmen wir an, dass die Quelldatenbank direkt an das DWH angebunden wurde. Alle neuen Datensätze der Quelltabelle *orders* soll in die Staging-Tabelle *s_orders* geladen werden. Neue Datensätze erkennen wir daran, dass das INSERT_DATE größer ist als das Lade-Datum des letzten Ladelaufes, dann könnte man die Staging-Tabelle z.B. wie folgt beladen:

```
INSERT INTO SA.s_orders (
SELECT * FROM orders
WHERE
INSERT_DATE > (Select max(load_date) from sa.utl_src_loads where table_name
= 'ORDERS')
);
```

Wenn man jetzt mehrere Tabellen mit mit dieser Vorgehensweise beladen will, dann muss man dieses Statement nicht kopieren. Man kann die zu ladenden Tabellen in einer Metadaten-Tabelle speichern (utl_src_tables), und dann dynamisch mit einem Statement laden.

```
FOR t_name in (Select table_name from sa.utl_src_tables)
LOOP
   EXECUTE IMMEDIATE '
      INSERT INTO SA.s_' || t_name || '(
         SELECT * FROM src.' || t_name || '
         WHERE INSERT_DATE > (Select max(load_date) from   sa.utl_src_loads
where table_name = ''' || t_name || ''') )';
END LOOP;
```

Natürlich ist das jetzt nur ein einfaches Beispiel. Eine Abfrage mit ‚SELECT * FROM' ist immer fehleranfällig, weil hier man sich hier auf die richtige Reihenfolge der Attribute verlässt. Besser wäre es, die Attribute explizit anzugeben. Aber das ist auch kein Problem, die Attribute kann man z.B. aus der Oracle-View ALL_TAB_COLUMNS auslesen und in das Statement einbauen. Zusätzlich könnte man z.B. noch technische Attribute, wie, z.B. einen Timestamp oder eine load_id in die Zieltabelle schreiben. Auch lassen sich nicht alle Tabellen so einfach beladen, aber auch komplexere Beladungsroutinen (Deltaerkennung, Partitionsweises laden, oder laden mit External Tables) kann man gut standardisieren, und generische Beladungsroutinen schreiben.
Dyamisches SQL bietet sich für viele Prozesse oder Teilprozesse an, die relativ generisch sind. Beispielsweise wird in vielen DWHs während der Beladung auch die Datenqualität anhand von vorher definierten Regeln geprüft. In einem Projekt generieren wir z.B. Views, um Testdaten zu anonymisieren.

Materialized Views

Die nächste Technologie, die ich betrachten möchte, heißt *Materialized View*. Eine „normale" View ist bekanntermaßen eine Abfrage, die wie eine Tabelle angesprochen werden kann. Bei einer Materialized View, abgekürtzt MatView oder MView, werden die Daten wirklich in einer Zwischentabelle gespeichert, um die Abfrage-Performanz zu erhöhen. Diese MViews können also z.B. jede Nacht neu beladen werden, tagsüber können dann in der Mview aktuelle Datenabgefragt werden. Soweit entspricht eine Mview quasi einer normalen ETL-Beladungsstrecke, die jede Nacht eine Zieltabelle belädt. Mviews habe jedoch diverse Vorteile gegenüber konventionellen ETL-Strecken:

- Die Art der Beladung ist von der fachlichen Logik getrennt. Man kann erst die fachliche Logik entwickeln, und z.B. als View implementieren, und sich danach über die technische Beladung Gedanken machen. Bei klassischen ETL-Tools muss man schon bei der Implementierung der Berechnungslogik berücksichtigen, ob die Zieltabelle inkrementell oder bei jeder Beladung komplett beladen wird. Bei Matviews gibt es diverse Beladungsmöglichkeiten (MView Refresh):
    - Complete Refresh: Die Mview wird jedes Mal komplett gelöscht und neu aufgebaut.
    - Fast Refresh: Die Mview loggt alle Änderungen in den Quelltabellen seit dem letzten Refresh mit, und führt ein inkrementelles Update aus. Dieses Refresh kann bei jeder Änderung in der Quelltabelle direkt ausgeführt werden (on COMMIT), oder während der klassischen ETL-Verarbeitung.
    - Partition Change Tracking (PCT) Refresh: Hier werden nur die geänderten Partitionen in der Quelltabelle neu berechnet.
    - Out-Of-Place Refresh: Hier wird erst eine temporäre Tabelle geladen, die dann nach der Beladung mit der MView-Tabelle oder einer Partition getauscht wird.
    - Synchronous Refresh: Jede Änderung in den Quelltabellen wird gleichzeitig auf die MView angewendet.
- Oracle stellt Mechanismen bereit, um Änderungen in den Quelltabellen automatisch zu erkennen.
- Mviews sind immer synchron zu den Quelltabellen. Bei normalen Beladungsstrecken können Daten-Inkonsistenzen auftreten, z.B. durch Programmierfehler oder Beladungsabbrüchen.
- Mviews bieten eine ganze Reihe von Metadaten (Tabelle USER_MVIEWS), vergleichbar mit Metadaten eines ETL-Tools:
    - Sind die Daten aktuell?
    - Wie wurde die Mview das letzte Mal beladen?
    - Wann wurde die MVIEW beladen?
    - Wann haben sich die Quelltabellen das letzte Mal geändert?

Mviews werden im DWH häufig dazu verwendet, Aggregate von Faktentabellen bereitzustellen. Allerdings kann man m.E. damit auch Datentransformationen, wie z.B. eine Kennzahlenhierarchie, aufbauen: als erstes wird eine Faktentabelle mit Basisfakten, geliefert vom Quellsystem beladen. Darauf aufbauend kann man dann mit MViews mit komplexeren Kennzahlen erstellen.

SQL Model Clause

Materialized Views bieten eine gute Möglichkeit, die Beladungsstrategie von der fachlichen Logik zu trennen. Das bringt uns zur nächsten Frage: Wie setze ich die fachliche Logik am besten in SQL um? Viele ETL-Tools bieten gerade hier einen großen Vorteil durch ihre grafische Oberfläche. Mit SQL hat man nur Code, deswegen sollte dieser so wartbar wie nur möglich sein.

Nehmen wir an, wir haben eine Faktentabelle, die die Verkäufe eines Produktes pro Jahr, Produkt, und Niederlassung enthält:

```
SELECT country, product, year, sales from fkt_sales;
```

Um diese Tabelle zu erstellen, haben wir die Daten aus verschiedenen Quellsystemen geladen und zusammengeführt, evtl. fehlerhafte Datensätze erkannt und ausgesteuert, und die Daten in eine Struktur aus Dimensionen und Fakten gebracht. Die Tabelle sieht also eigentlich ganz gut aus, aber leider fängt genau hier die Arbeit erst an. Ein Beispiel:
Im Jahr 2002 gab es viele fehlerhafte Datensätze, die nicht geladen werden konnten. Um diesen Fehler auszugleichen, sollen die Verkäufe von Tomaten in diesem Jahr mit dem Faktor 1.2 multipliziert werden :

```
Sales['Tomatoe', 2002] = Sales['Tomatoe',2002]*1.2
```

Die Daten für das Jahr 2003 liegen für das Produkt Tomate noch nicht vor. Deswegen soll die Summe der Jahre 2000 und 2001 genommen werden:

```
Sales['Tomatoe', 2003] = Sales['Tomatoe,2000']+Sales['Tomatoe',2001]
```

Wäre es nicht schön, wenn wir solche Regeln direkt in SQL verwenden könnten, ohne komplexe CASE WHEN-Ausdrücke? Das ist möglich mit dem SQL Model Clause. In diesem Beispiel werden die Kennzahlen über die Dimensionsattribute angesprochen, ähnlich wie Zellen in einer Tabellenkalkulation. Dazu müssen wir definieren, welche Attribute Dimensionen sind, und welche Attribute Kennzahlen sind:

```
PARTITION BY (country) DIMENSION BY (product, year)
MEASURES (sales sales)
```

Der Ausdruck “Partition by …“ teilt die Tabelle in logische Partitionen, die unabhängig voneinander bearbeitet werden, ähnlich wie der partition by-Clause in analytischen Funktionen oder der group by Clause bei Aggregatfunktionen.
Zusammengefügt sieht das dann so aus:

```
SELECT country, product, year sales
FROM fkt_sales
WHERE country IN ('Italy', 'Japan')
MODEL
PARTITION BY (country) DIMENSION BY (product, year)
```

```
MEASURES (sales sales)
RULES
(
 Sales['Tomatoe', 2002] = Sales['Tomatoe'],2002]*1.2,
 Sales['Tomatoe', 2003] = Sales['Tomatoe',2000]+Sales['Tomatoe',2001]
 )
```

Einige weitere Regeln sind z.B.:
Für alle Produkte in Jahr 2000, nehme den Wert des Vorjahres, und multipliziere ihn mit 2:

```
Sales[any, 2000] = Sales[CV(prod), CV(year-1)]*2
```

Für Verkäufe von Potatoes im Jahr 2001, nehme das Maximum der Verkäufe aus den Jahren 1997 bis 2000:

```
Sales['Potatoes',2001] = MAX(sales)['Potatoes',year BETWEEN 1997 AND 2000]
```

Der Model Clause ist sehr mächtig, und bietet viele Möglichkeiten, die weit über diesen Vortrag hinausgehen (z.B. Analytische Funktionen, Regeln, die auf anderen Regeln basieren, Loops, …)
Interessante fachliche Fragestellungen, die man mit dem Model Clause gut lösen kann, sind z.B. auch Regeln für Verkaufsprovisionierungen (die umfangreich und komplex sein können), oder z.B. die unterschiedliche Behandlung von verschiedenen Vertriebsorganisationen. Besonders, wenn Regeln von mehreren Dimensionen abhängen die Verkäufe eines Produktes sollen, ist es einfacher, die Kennzahlen per Model Clause umzubiegen, als Dimensionen umzumappen.

Fazit

Anhand der obigen Beispiele kann man sehen, dass sich auch die Oracle-Datenbank-Bordmittel gut für ETL-Prozesse eignen. Die vorgestellten Techniken lassen sich aber auch in Verbindung mit ETL-Tools einsetzen. Selbst in DWH-Projekten, in denen Tools wie z.B. Informatica eingesetzt werden, gibt es diverse Views, PL/SQL- Routinen (z.B. für Index- und Partition-Management, Metadaten) und MatViews (um Daten zu aggregieren).

Eine Besonderheit sind ELT-Tools wie z.B. ODI. Diese Tools verarbeiten die Daten nicht in einer eigenen Engine, sondern generieren SQL-Code, der dann auf der Datenbank ausgeführt wird.

Wichtig ist es, genau die Anforderungen zu kennen, die man an ETL-Tool hat. Viele ETL-Tools bieten z.B. komplexere Module, um Quellsysteme wie z.B. SAP anzubinden.

Kontaktadresse:
Christian König
CGI (Germany) GmbH & Co. KG
Heerdter Lohweg 35
D-40549 Düsseldorf

Telefon: +49 170569 7832
E-Mail christian.koenig@cgi.com
Internet: www.de.cgi.com