StarView: mit statischem SQL dynamische Abfragen auf StarSchema

Slavomir Nagy
metafinanz Informationssysteme GmbH
München

Schlüsselworte
Starschema; Business Intelligence; Data Warehousing; dimensionale Datenmodellierung; SQL; Ausführungsplan

Einleitung

Starschema hat sich im Data Warehousing Umfeld in den vergangenen Jahren als Standardlösung bei der dimensionalen Daten-Modellierung der fachlichen Data Marts etabliert. Im Vortrag werden anhand des Oracle Sample Schemas SH die verschieden Möglichkeiten untersucht, wie aus diesem Starschema mit Hilfe von einem View mit statischem SQL dynamische Abfragen erzeugt werden können. Die Lösungen werden dann aus verschieden Blickwinkeln, wie Abfrageperformanz und Unterstützung von Oracle Features, wie „Partition Pruning“, „Query Rewrite“ und „Row Level Security“, untersucht und ausgewertet.

Zielsetzung

Die meisten BI Tools speichern die Informationen über das Datenmodell in Form von Metadaten in einem Repository. Diese Angaben werden dann von SQL Generatoren verwendet, um auf Basis der Benutzervorgaben in einer meist grafischen Oberfläche dynamische SQL Abfragen zu generieren und diese dann in einer relationalen Datenbank auszuführen.

Eine durch BI Tool generierte Abfrage aus dem Beispielschema SH kann dann wie folgt aussehen:

```
SELECT calendar_year, calendar_month_name,
  prod_category,prod_subcategory,
  SUM(amount_sold) as amount_sold
FROM sales s
JOIN times t ON t.time_id=s.time_id
JOIN products p ON p.prod_id=s.prod_id
GROUP BY calendar_year, calendar_month_name,
  prod_category,prod_subcategory
ORDER BY calendar_year, calendar_month_name,
  prod_category,prod_subcategory
```

Durch die Bereitstellung eines einzelnen Views wird versucht diese Flexibilität des BI Tools nachzubauen. Ziel ist es durch verschiedene Lösungswege einen identischen Ausführungsplan abzubilden, damit die Performanz akzeptabel bleibt. Die Abfrage aus dem View hat dabei das folgende Muster:

```
SELECT calendar_year, calendar_month_name,
  prod_category,prod_subcategory,
  SUM(amount_sold) as amount_sold
FROM <view>
GROUP BY calendar_year, calendar_month_name,
```

```
  prod_category,prod_subcategory
ORDER BY calendar_year, calendar_month_name,
  prod_category,prod_subcategory
```

Lösungen

Die einfachste Lösung ist alle Verknüpfungen zwischen der Faktentabelle und den Dimensionen direkt im View vorzubereiten. Das führt allerdings im Ausführungsplan dazu, dass immer alle Dimensionstabellen in jeder Abfrage aus dem View benutzt werden. Bei den Dimensionstabellen, bei welchen mindestens eine Spalte abgefragt wird, erfolgt der Zugriff über den Index auf dem Primärschlüssel mit anschließendem ROWID-Zugriff auf die Dimensionstabelle. Bei den Dimensionstabellen, bei welchen keine Spalte abgefragt wurde, wird lediglich der Primärschlüssel geprüft. Das ist allerdings nicht ganz notwendig, weil die Fremdschlüssel in der Faktentabelle diese Beziehung garantieren und die Primärschlüssel auf den Dimensionstabellen stellen sicher, dass es immer nur eine Zeile in der Dimensionstabelle mit diesem Wert gibt.

Deswegen wird bei der zweiten Lösung dieser Nachteil durch korrelierte Unterabfragen eliminiert. Dabei wird pro Spalte der Dimensionstabelle eine Unterabfrage in der SELECT-Klausel des Views generiert. Der Ausführungsplan verrät uns allerdings, dass bei diesem View pro Spalte der Dimensionstabelle immer ein Indexzugriff über den Index auf dem Primärschlüssel stattfindet.

Um die Zugriffe pro Spalte zu vermeiden wird bei der dritten Lösung nur eine Unterabfrage pro Dimensionstabelle generiert. Dabei werden in der inneren Abfrage alle Spaltenwerte der Dimensionstabelle verkettet und durch ein geeignetes Trennzeichen getrennt. In der äußeren Abfrage wird dann die Zeichenkette wieder in die einzelnen Werte zerlegt. Der Ausführungsplan zeigt, dass es weder Zugriffe auf die nicht benötigten Dimensionstabellen noch die mehrfache Abfragen aus der gleichen Dimensionstabelle gibt. Doch bei der Ausführung merkt man bereits beim Beispielschema SH, dass die Laufzeiten deutlich länger sind als es bei den anderen Lösungen der Fall war. Ein genaueres betrachten des Ausführungsplans verrät auch den Grund: die Speicherplatzanforderungen sind enorm hoch.

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------|
|   0 | SELECT STATEMENT             |              |       |       |   613 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| TIMES        |     1 |   198 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | TIMES_PK     |     1 |       |     1   (0)| 00:00:01 |
|   3 |  TABLE ACCESS BY INDEX ROWID| PRODUCTS     |     1 |   173 |     1   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN          | PRODUCTS_PK  |     1 |       |     0   (0)|          |
|   5 |  SORT ORDER BY               |              |   918K|  7028M|   613  (16)| 00:00:08 |
|   6 |   HASH GROUP BY              |              |   918K|  7028M|   613  (16)| 00:00:08 |
|   7 |    PARTITION RANGE ALL       |              |   918K|  7028M|   531   (3)| 00:00:07 |
|   8 |     VIEW                     | V_SALES_VER3 |   918K|  7028M|   531   (3)| 00:00:07 |
|   9 |      TABLE ACCESS FULL       | SALES        |   918K|    14M|   531   (3)| 00:00:07 |
-------------------------------------------------------------------------------------------
```

Außerdem ist diese Lösung nicht ohne Einschränkungen, weil die maximale Länge einer Zeichenkette in Oracle bei den Dimensionstabellen mit vielen Spalten schnell erreicht werden kann. Und zusätzlich kann auch die falsche Wahl des Trennzeichens zu Problemen führen, weil das Trennzeichen später auch in den Spaltenwerten vorkommen kann.

Trotzdem kann man die Lösung durch die Benutzung des Datenbanktyps OBJECT noch verbessern. Pro Dimensionstabelle wird eine CREATE TYPE Anweisung generiert, welche ein OBJECT anlegt und dieses alle Spalten als Attribute enthält.

```
create type t_TIMES_obj as object (
TIME_ID DATE
,DAY_NAME VARCHAR2(9)
,DAY_NUMBER_IN_WEEK NUMBER(1,0)
…
```

In der inneren Abfrage werden dann durch den „Konstruktor“ die Spaltenwerte gesammelt und in der äußeren Abfrage dann diese Attribute wieder als einzelne Spalten des Views abfragbar gemacht. Damit sind die Probleme mit der Länge der Zeichenkette und mit dem Trennzeichen gelöst

Als letzte Lösung wird dann ein OUTER JOIN statt INNER JOIN zwischen Faktentabelle und den Dimensionstabellen untersucht. Obwohl es auf den ersten Blick eine nicht ganz regelkonforme Lösung ist, hat dieser Ansatz auch viele Vorteile. Ein INNER JOIN kann durch ein OUTER JOIN ersetzt werden, weil die Fremdschlüssel in der Faktentabelle und die Primärschlüssel in den Dimensionstabelle garantieren, dass dabei keine Datensätze verloren gehen und auch keine Datensätze mehrmals abfragt werden. Das gilt auch für den Fall, dass diese nicht aktiviert sind, weil sich dann die Ladeprozesse um ihre Gültigkeit kümmern. Mit dieser Lösung erreichen wir einen Ausführungsplan., der nur die Dimensionstabellen benutzt, die, welche auch abgefragt werden. Die Performanz der Abfragen ist vergleichbar mit der generierten Abfrage aus dem BI Tool.

Es bleibt noch zu untersuchen, wie gut die einzelnen Lösungen die folgenden Oracle Features unterstützen.

Partitin Pruning
Um die Fähigkeiten der einzelnen Lösungen das „Partition Pruning“ zu unterstützen zu untersuchen, wird die BI Abfrage um eine WHERE-Bedingung erweitert.

```
WHERE calendar_year = 2001
AND calendar_month_name = 'April'
```

Anschließend wird der geschätzte und der tatsächliche Ausführungsplan analysiert. Die Lösungen mit Unterabfragen können vom „Partition Pruning“ nicht profitieren, weil die Information über den Partitionsschlüssel bei der Verarbeitung der Unterabfrage verloren geht. Das gleiche gilt auch für die Benutzung der Bitmapindizes auf den Fremdschlüsselspalten der Faktentabelle. Auch diese werden bei den Lösungen mit Unterabfragen nicht optimal benutzt.

Query Rewrite
Bei diesem Test wird ein einfaches „Materialized View“ erstellt, welche für Dimensionstabellen TIMES und CUSTOMERS alle Abfragen unterstützt:

```
CREATE MATERIALIZED VIEW mv_sales
ENABLE QUERY REWRITE
AS
SELECT time_id, cust_id,
SUM(amount_sold), COUNT(amount_sold),
COUNT(*)
FROM times t
GROUP BY time_id, cust_id
```

Die BI Abfrage wird so angepasst, dass sie statt Dimensionstabelle PRODUCTS die Dimensionstabelle CUSTOMERS verwendet. Durch „Query Rewrite“ wird sie dann so umgeschrieben, dass kein Zugriff auf die Faktentabelle notwendig ist.

```
SELECT calendar_year, cust_city,
  SUM(amount_sold) AS amount_sold
FROM sales s
JOIN times t ON t.time_id=s.time_id
JOIN customers c ON c.cust_id=s.cust_id
GROUP BY calendar_year, cust_city
ORDER BY calendar_year, cust_city
```

Auch die Abfrage aus dem View wird auf die Dimensionstabelle CUSTOMERS umgeschaltet:

```
SELECT calendar_year, cust_city,
  SUM(amount_sold) AS amount_sold
FROM <view>
GROUP BY calendar_year, cust_city
ORDER BY calendar_year, cust_city
```

Wieder ist die Lösung mit OUTER JOINs im Vorteil, sie unterstützt „Query Rewrite“. Die Lösungen mit Unterabfragen nicht, genauso wie die Lösung mit INNER JOINs. Warum funktioniert „Query Rewrite“ mit INNER JOINs nicht? Weil auch die anderen Dimensionstabellen im Ausführungsplan angesprochen werden und weil diese Fremdschlüsselspalten im „Materialized View“ nicht in der GROUP BY-Klausel vorkommen.

Row Level Security (RLS)

Um diese Feature zu testen, wird die Dimensionstabelle CHANNELS durch RLS geschützt. Die Prädikatfunktion erlaubt dann dem Schemabesitzer SH vollen Zugriff auf die Tabelle, der Benutzer SCOTT darf aber nur die Zeilen mit CHANNEL_CLASS gleich ‚Direct‘ sehen. Um dieses Feature zu prüfen, werden die BI Abfrage und die Abfrage aus dem View so angepasst, dass sie mit der Dimensionstabelle CHANNELS arbeiten.

Auf den ersten Blick hat keine der Lösungen Probleme mit diesem Feature. Alle liefern die gleichen Ergebnisse, der Schutz funktioniert. Aber wenn man die Dimensionstabelle CHANNELS aus den Abfragen entfernt, merkt man, dass die Lösung mit INNER JOINs andere Ergebnisse liefert als alle anderen Lösungen. Warum? Weil diese Lösung weiterhin intern im Ausführungsplan die Dimensionstabelle CHANNELS verwendet. Das kann erwünscht werden oder nicht. Wenn die Endanwender die Gesamtzahlen des Unternehmens nicht sehen dürfen, dann hat INNER JOIN die richtigen Ergebnisse geliefert. Wenn aber der Schutz auf der Dimensionstabelle CHANNELS nur dazu dient, es nur privilegierten Benutzern die Analyse nach der Dimension CHANNELS zu ermöglichen, sie dürfen aber die Gesamtzahlen der Firma sehen, dann waren die anderen Lösungen richtig. Es hängt also von den fachlichen Anforderungen ab, welche Lösung hier die richtige ist. Die BI Tools unterscheiden dabei bei der Generierung der SQL Abfragen zwischen einen erforderlichen und einer nicht erforderlichen Tabelle. Der INNER JOIN ist also die Lösung für die angeforderte Tabelle, alle anderen Lösungen unterstützen die nicht erforderliche Tabelle.

Fazit
Es ist tatsächlich möglich ein View, also statisches SQL, auf dem Starschema zu bauen, welches es ermöglicht, die Abfragen relativ dynamisch auszuführen.
Für den Aufbau eines StarView muss man allerdings ein kleines Regelwerk beachten.

- Durch RLS geschützte Dimensionstabellen, welche es verhindern sollen, die Auswertung über alle Daten zu machen, müssen über INNER JOIN verknüpft werden

- Durch RLS geschützte Dimensionstabellen, welche es nicht verhindern sollen, die Auswertung über alle Daten zu machen, dafür aber die Analyse nach dieser Dimension nicht erlauben sollen, müssen über OUTER JOIN verknüpft werden

- Fremdschlüssel in die Dimensionstabelle, welche **nicht** in allen Materialized Views verwendet werden, müssen über OUTER JOIN angebunden werden

- Partitionsschlüssel, welche gleichzeitig als Fremdschlüssel in die Dimensionstabelle funktionieren, können mit INNER JOIN oder OUTER JOIN verknüpft werden

- Selektive Dimensionen, bei welchen die Bitmapindizes oft verwendet werden, idealerweise mit OUTER JOIN anbinden

Kontaktadresse:
Slavomir Nagy
Metafinanz Informationssysteme GmbH
Leopoldst. 146
D-80804 München

Telefon: +49 (0) 89-360531 5135
Fax: +49 (0) 89-360531 5015
E-Mail slavomir.nagy@metafinanz.de
Internet: www.metafinanz.de