Materialized Views – Praktischer Einsatz vor und in 12c

Jonas Gassenmeyer und Sven Weller
syntegrisinformationsolutions GmbH
Neu Isenburg

Schlüsselworte
Materialized Views, Performance, Replikation, fast refresh, complete refresh, Downtime, out of place refresh, truncate, append

Einleitung
Materialized Views (MVs) bieten sowohl eine einfache Lösung für eine voraggregierte Form großer Datenmengen als auch für die Vervielfältigung gleicher Daten auf verschiedene Standorte (Replikation). Der Vortrag arbeitet heraus, welche Vorteile Materialized Views gegenüber normalen Tabellen haben. Er geht auf die tollen kleinen Features ein, die Oracle bereitstellt, um eine MVs für nahezu jedes Szenario einsatzfähig zu machen. Hierzu zählen unter anderem:

- Performance Optimierung beim Refresh Vorgang (atomicrefresh)
- Inkrementelle Refreshes beim Replizieren großer Datenmengen (fast refresh)
- Performance Optimierung für Queries, die die MV bereits kennt (queryrewrite)
- Change Management für MVs - geringe Downtime (prebuildtable)
- Eine interessante Neuerung in 12c (out of place refresh)

Der Vortrag berichtet Praxisbeispiele (aus dem Pharma Umfeld) und realistische Zahlen. Das hilft einzuschätzen, wann und weshalb sich der Einsatz von MVs lohnt und welche Fallstricke Sie unbedingt beachten sollten.
Im Vortrag wird nicht behandelt, welchen Einschränkungen das SQL unterliegt (connectby, setoperators, distinct, not exists).Replikationslösungen können außerdem so aufgebaut werden, dass UPDATES auf Seite der Satelliten möglich sind (updateableMvs).

Was sind Materialized Views?
MVs sind zum einen ein Performance Feature, das ohne zusätzliches Tuning für schnellere SQLs sorgt. Zum anderen lassen sich Tabellen ohne großen Aufwand (Trigger, Tracken der Änderungen, Updates...) replizieren.Grundsätzlich kann eine MV wie eine Tabelle in Oracle behandelt werden. DML Operationen können durchgeführt werden, wenn die vorhandenen Rechte eingeräumt wurden und gewöhnliche SQL Abfragen funktionieren reibungslos. Auch ein Index lässt sich auf beliebige Spalten der MV anlegen.
Betrachtet man die nach einem `CREATE MATERIALIZED VIEW` entstandenen Objekt, wird deutlich, dass Oracle im Hintergrund sogar eine Tabelle angelegt hat:

```
select SEGMENT_TYPE
fromuser_segments
where SEGMENT_NAME = 'MV_NAME';


SEGMENT_TYPE

------------------

TABLE
```

Anders als bei Tabellen wird beim Anlegen einer MV allerdings ein SQL hinterlegt, welches die Daten beschreibt, die in der Datenbank abgelegt werden. Nimmt man den Namen "Materialized View" genauer unter die Lupe, dann steckt diese Eigenschaft schon darin:

-Materialized = "gespeichert/persistiert"
-View = "SQL Definition"

MVs bieten somit eine Möglichkeit, aufwendige SQLs (die unter Umständen auf sehr großen Datenmengen aufsetzen) in einer voraggregierten Form in der Datenbank abzulegen. Ein performance-kritisches SQL könnte dann beispielsweise nachts laufen. Analysen, die tagsüber mit geringeren Abfragezeiten gegen die Datenbank abgesetzt werden müssen, können mit einfacheren SQLs darauf aufsetzen. Was aber ist der Vorteil gegenüber einer normalen Tabelle, die mittels

```
CREATE TABLE daily_statsas(
               SELECT avg(col)--...
               FROM BIG_TABLE
               --WHERE...
);
```

das selbe bieten könnte? - Oracle liefert mit der query rewrite Option die passende Antwort. Durch eine Angabe beim Erstellen der MV kennt die Oracle Datenbank das in der MV hinterlegte SQL und erkennt automatisch, wenn Queries, die adhoc in einer Session (OLAP) gegen die Datenbank geschickt werden, auf den voraggregierten Datenbestand umgebogen werden können, weil dieser bereits die notwendigen Ergebnisse enthält. So komplex dieses Performance Feature im Hintergrund auch implementiert sein mag - es ist spielend einfach mittels des folgenden Befehls einzusetzen:

```
CREATE MATERIALIZED VIEW schema.mv_name
ENSABLE QUERY REWRITE
AS (SELECT avg(col)--...
FROM BIG_TABLE
--WHERE...
);
```

Ob ihre Query dann einem rewrite unterzogen wurde, zeigt Ihnen der Explain Plan an:

```
-------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes | Cost
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |     4 |   123 |     3
|   1 |  MAT_VIEW REWRITE ACCESS FULL| MV_NAME |     4 |   545 |     3
-------------------------------------------------------------------------------
```

Damit ist der Einsatz im DWH Umfeld besonders attraktiv. Ein weiterer Vorteil ist die Vervielfältigung gleicher Daten auf verschiedene Standorte (Replikation). Der wesentliche Vorteil gegenüber Tabellen Objekten ist hierbei, dass der Refresh Vorgang bereits durch Oracle Funktionalitäten erreicht werden kann. Einer Tabelle würde diese zusätzliche Intelligenz fehlen. Dem Entwickler stehen sogar verschiedene Methoden zur Verfügung: Er kann über den Zeitpunkt des Refreshs (bei Commit oder manuell gesteuert) und über die Art des Updates (inkrementell oder komplett) entscheiden. Hierzu müssen die notwendigen Einstellungen ebenfalls im Header definiert werden:

```
CREATE MATERIALIZED VIEW schema.mv_name
REFRESH FAST ON DEMAND
--REFRESH COMPLETE ON COMMIT
--REFRESH FAST ON COMMIT
--REFRESH COMPLETE ON DEMAND
AS (select --…
```

Entscheidet man sich für den manuellen Refresh (on demand) so aktualisiert sich der Datenbestand der MV nur dann, wenn der Befehl `exec dbms_mview.refresh('MV_MASTER', 'C');` ausgeführt wird.
Der zweite Parameter bestimmt, ob die MV lediglich inkrementelle Änderungen erhält („F“ wie fast), oder ob der gesamte Datenbestand erneuert wird („C“ wie complete).
Die Art der Refreshs, die in der MV Definition mitgegeben wurde, würde vom Parameter dann überschrieben werden. Eine mittels `refresh fast on demand` angelegte MV kann somit ebenfalls mittels complete refresh erneuert werden. Das macht auch Sinn: Wenigstens beim Anlegen muss schließlich einmal der gesamte Datenbestand ermittelt werden. Oracle erkennt das automatisch.
Die Prozedur könnte z.B. regelmäßig über einen scheduled job aufgerufen werden.

Performance Überlegungen
Unter Umständen lohnt sich eine weitere Performance Optimierung während des Refresh Vorgangs. Es handelt sich um einen dritten Parameter in der Prozedur (atomic_refresh). Er kann nur beeinflusst werden, wenn man den refresh manuell anstößt (nicht bei on commit).
Per Default steht er auf „true“, sodass die Frage naheliegt, was ein `atomic_refresh=false` bewirkt.

Wird der gesamte Datenbestand erneuert, so führt Oracle im Hintergrund ein `TRUNCATE` statt einem `DELETE` aus.
Alle neu einzutragenden Rows werden mittels dem vom Loader bekannten /*+ append*/ Hint angehängt, ohne die eventuell durch ein `DELETE` entstandenen Lücken im Datenblock zu suchen. Das sorgt im Hintergrund für wesentlich weniger `UNDO` und `REDO` (lediglich Metadaten, keine Sicherung der eigentlichen Daten) Informationen.
Außerdem werden erst nach dem Eintrag aller neuen/ geänderten Rows eventuelle Indizes gepflegt. Auch der Zeitpunkt der Commits kann variieren.

Sie sollten definitiv bedenken, dass das `TRUNCATE` nicht in allen Fällen eine geeignete Lösung ist. Bis die neuen Daten geladen sind, liefern Abfragen gegen die MV keine Daten. Durch einen directpathload, wenn kein `TRUNCATE` gemacht wurde, entstehen Lücken unterhalb der high watermark, was sich negativ auf die Performance auswirken kann. Auf diese Besonderheiten wird im Vortrag weiter eingegangen. Es ist Aufgabe des Entwicklers, die richtige Kombination zu finden. Nur weil der ein refresh `FAST` genannt wird, muss er z.B. nicht immer der schnellste sein.

Change Management
Eine weitere Einstellungsmöglichkeit kann die Arbeit in der Praxis extrem erleichtern:

```
CREATE MATERIALIZED VIEW schema.mv_name
ON PREBUILD TABLE
AS (select
```

Haben Sie schon einmal versucht, vorhandene Spaltennamen abzuändern oder neue Spalten zur MV hinzuzufügen? Ein einfaches `ALTER MATERIALIZED VIEW` oder `CREATE OR REPLACE` werden Sie

nicht finden. Das erschwert Änderungen an Materialized Views – gerade wenn eine geringe Downtime notwendig ist. Bedenken Sie, dass Ihr hinterlegtes SQL unter Umständen mehrere Tage laufen könnte bis es alle Daten ermittelt hat. Während dieser Zeit wäre das Objekt für den Livebetrieb nicht erreichbar! Im Vortrag wird die prebuildtable Option (und auch `build deffered`) erläutert. Es handelt sich um die Möglichkeit, eine bereits vorhandene Tabelle in eine MV zu überführen. Somit können die Daten zunächst in eine Tabelle geladen werden. Nebenbei setzen Abfragen weiterhin auf der alten MV auf. Ist die Tabelle fertig, wird sie zu einer MV umgewandelt und um die nützlichen Features für den Refresh etc. ergänzt.
Fallstricke könnten sich ergeben, sobald Sie die eigentliche MV wieder löschen. Im Gegensatz zu direkt erzeugten MVs bleibt das eigentliche Tabellen Objekt erhalten. In der Praxis sorgte folgendes Szenario, welches in Abbildung 1 gezeigt ist in der Oracle Version 10.2.0.5.0 - 64bit, für fehlerhafte SQLs:
Die Abbildung zeigt, dass `MVTOP` mit seinem SQL auf `MVERR` geht (Datenbanklink zu DB 2).
In DB2 wurde ein Synonym `MVERRSYN` angelegt, welches auf die darunterliegende `MVERR` zeigte.
Diese `MVERR` wurde gedropt und das Synonym wurde auf `MVNEW` umgebogen.
Das hinterlegte SQL in `MVTOP` hat das `SYNONYM` scheinbar aufgelöst und in den darunterliegenden Objektnamen `MVERR` umgewandelt.
Da nur die MV und nicht der prebuild table gelöscht werden, findet DB3 weiterhin ein Objekt `MVERR` und sieht keinen Anlass den Cache zu erneuern. Die Daten werden also fälschlicherweise noch von `MVERR` selektiert, statt von `MVNEW`.

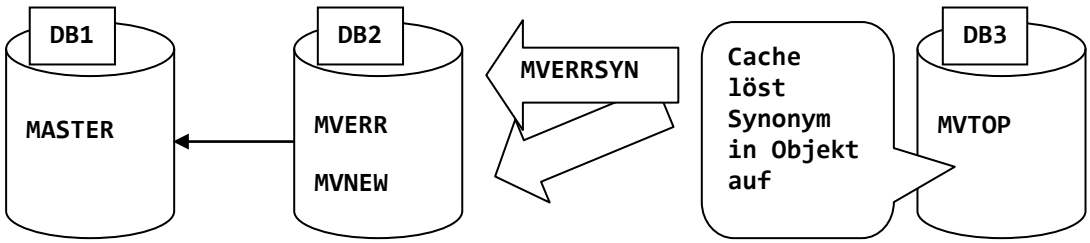

Abb. 1: Caching Probleme bei der prebuildtable Option in 10.2.0.5.0

Abhilfe würden eine explizite Erneuerung des Caches mittels `alter system flush shared_pool` schaffen. Allerding sind dazu DBA Rechte erforderlich. Häufig hat der Entwickler hierauf also keinen Einfluss. Mit einem `DROP TABLE MVERR` löschen Sie den prebuild table und sind auf der sicheren Seite.

Neuerungen in 12c
Im Bereich des Change Managements ist auch die interessanteste Neuerung in 12c angesiedelt. Der outofplace Refresh kommt als vierter Parameter zur bereits bekannten Refresh Prozedur hinzu:

```
DBMS_MVIEW.REFRESH('MV_NAME', 'F', FALSE, out_of_place => TRUE);
```

Beim Refresh wird Oracle im Hintergrund automatisch eine Kopie der MV in einer neuen Tabelle mit dem Präfix „RV$“ (gefolgt von der hexadezimalen Objekt Id der ursprünglichen MV) erzeugen. In dieser findet die neue Datensammlung statt. Danach transformiert Oracle analog zum prebuild table Vorgang die Tabelle in eine MV.

Zwar werden alle Arten (fast, complete und force) unterstützt, jedoch ist der `atomic_refresh =false` zwangsläufig einzusetzen, wenn `out_of_place=true` gewünscht ist. Oracle würde eine falsche Kombination der Parameter mit einer entsprechenden Fehlermeldung quitieren: Damit sind Refresh Vorgänge, welche `ON COMMIT` getriggert, werden natürlich auch wieder aus dem Rennen. MVs mit LOB Spalten oder materialized view logs werden derzeit noch nicht unterstützt. Andere Einschränkungen können Sie der Dokumentation entnehmen.

Weil der out of place refresh ein `INSERT` in eine vollkommen neue Tabelle vornimmt, wird Oracle noch keine Statistiken kennen. Diese sollten also noch einmal explizit berechnet werden, z.B.

```
dbms_stats.gather_table_stats (
                ownname => 'MOSAIC_CUSTOM'
                , tabname => 'RPT_BATCHES_MONTHLY_OLD'
                , estimate_percent =>dbms_stats.auto_sample_size
                , block_sample =>true);
```

atomic refreshes=false hat übrigens ähnliche Probleme. Für die Indizes auf der MV sollte das Ganze anders aussehen, da diese ja erst nach dem `INSERT` erzeugt wurden (siehe oben).

Geheimtip
Häufig werden in der Praxis Datenbanklinks eingesetzt, welche dann von den Satelliten zum Master zeigen. Je nach Datenbestand kann das, was übers Netzwerk ausgetauscht wird, ziemlich viel werden. Wenn das SQL Joins enthält, hat die Satelliten Datenbank unter Umständen keinen Zugriff auf wichtige Statistiken in der Masterinstanz. Das könnte im schlimmsten Fall zu einem fulltable Scan führen. Dann lohnt sich ein Test mit dem `/*+driving_site(site)*/` Hint.
Hier kann in Klammern gesteuert werden, welche der beiden Datenbanken den Join ausführen soll. Liegt die kleinere Tabelle zum Beispiel im Satelliten, wird das Resultset durch den Join mit dem Datensatz im Master stark reduziert.Wenn also die Berechnung vom Master ausgeführt wird und nur das eigentlich Ergebnis über das Netzwerk mitgeteilt wird, verringert sich die Netzwerklast. Vorsicht ist bei userdefinedfunctionsgeboten. Die könnten nicht im Master definiert sein, was zu einer Art Daten Ping Pong werden könnte.

Fazit
Der Vortrag zeigt anhand klarer Praxisbeispiele und realistischer Zahlen, wann und weshalb sich der Einsatz von Materialized Views lohnt. Man könnte sie als eine Art Tabelle mit zusätzlicher Intelligenz beschreiben. Das bezieht sich vor allem darauf, dass sie wissen, welche Daten sie enthalten, wie sich andere Abfragen diese zu Nutze machen können und wie häufig dieser Datenbestand aktualisiert wird.

Kontaktadresse:
Jonas Gassenmeyer
syntegrisinformationsolutions GmbH
Hermannstraße 54-56
D-63263 Neu Isenburg

Telefon: +49 (0) 6102 - 29 86 68
Fax: +49 (0) 6102 - 55 88 06
E-Mail jonas.gassenmeyer@syntegris.de
Internet: www.syntegris.de/jg