Materialized Views – Wissenswertes und Neuerungen in 12c

Jonas Gassenmeyer
Syntegris information solutions GmbH
Neu Isenburg

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

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 MVs für nahezu jedes Szenario einsatzfähig zu machen. Hierzu zählen unter anderem:

- Performance Optimierung beim Refresh Vorgang (atomic refresh)
- Inkrementelle Refreshes beim Replizieren großer Datenmengen (fast refresh)
- Change Management für MVs - geringe Downtime (prebuild table)
- Neuerungen in 12c (12.1) out of place refresh, (12.2) Real-Time Materialized Views, (12.2) Statement-Level Refresh, (12.2) Refresh Statistics History

Der Vortrag berichtet von Praxisbeispielen (aus dem Pharma-Umfeld) und realistischen 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 (connect by, set Operatoren, distinct, not exists). Replikationslösungen können außerdem so aufgebaut werden, dass UPDATES auf Seite der Satelliten möglich sind (updateable Mvs). Auch das wird nicht gezeigt.

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 Objekte, 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 performancekritisches 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 tollen Eigenschaften, die ohne zusätzlichen Programmieraufwand einzusetzen sind, die passende Antwort. So komplex die jeweiligen Features im Hintergrund auch implementiert sein mögen - es ist häufig spielend einfach sie mittels minimaler Codeänderungen einzusetzen. 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. Häufig erfolgt eine Definition über den Header. Im unteren Beispiel wird z.B. angegeben, dass ein inkrementeller Refresh des Datenbestands erfolgen kann.

```
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 des 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 einen Complete Refresh erhalten. Das macht auch Sinn: Wenigstens beim Anlegen muss schließlich einmal der gesamte Datenbestand ermittelt werden. Oracle erkennt das automatisch und wirft einen Fehler, wenn direkt ein Fast Refresh erfolgt. Die Prozedur könnte z.B. regelmäßig über einen Scheduled Job aufgerufen werden.

Performance Überlegungen

Unter Umständen lohnen sich weitere Performance Optimierung während des Refresh Vorgangs. Dabei handelt es sich um weitere Parameter (z.B. `atomic_refresh`) in der Prozedur `dbms_mview.refresh` oder Änderungen im Header der MV Definition. Vor allem die Datenbank in Version 12.2 hat erfreuliche Neuigkeiten gebracht. Genannt werden sollten auf jeden Fall die folgenden Möglichkeiten:

- (schon vorherige Versionen): atomic_refresh und out_of_place
- (ab 12.2): Real-Time Materialized Views, Statement-Level Refresh und die neuen Data Dictionary Views, die Refresh-Statistiken über die einzelnen Refreshes enthalten und eine genauere Analyse ermöglichen.

Atomic Refresh

Per Default steht dieser Parameter auf „true“, sodass die Frage naheliegt, was ein `atomic_refresh=false` bewirkt. Der Name lässt aber schon erahnen, was im Hintergrund passiert:

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 Informationen. 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 direct path Load, wenn kein TRUNCATE gemacht wurde, entstehen Lücken unterhalb der High Watermark, was sich negativ auf die Performance auswirken kann. 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.

Out of Place Refresh
Der out of place Refresh kommt als vierter Parameter zur bereits bekannten Refresh Prozedur hinzu:
`DBMS_MVIEW.REFRESH('MV_NAME', 'F', FALSE, out_of_place => TRUE);`
Beim dieser Form des Refreshs 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 quittieren:

Real-Time Materialized Views
Queries, die adhoc in einer Session (OLAP) gegen die Datenbank geschickt werden, kann der Optimizer auf den voraggregierten Datenbestand in der MV umbiegen, weil das in der MV hinterlegte SQL die Ergebniszeilen bereits ermittelt hat. Sobald jedoch an der abhängigen Tabelle Änderungen (DML-Operationen) stattgefunden haben, wird die dazugehörige MV als „stale“ markiert und erfordert einen erneuten Refresh. Je nach Konfiguration muss ein solcher Refresh aber zum Zeitpunkt der adhoc Abfrage aber noch nicht stattgefunden haben. Deshalb können ab 12.2 auch als „stale“ betrachtete MVs für die Query Rewrite Option herangezogen werden. Im Hintergrund nutzt der Optimizer dann die Materialized View Logs (=> Fast Refresh), um das Delta zu ermitteln. Das Resultset wird also in „Real Time“ ermittelt. Die Abfrage liefert somit auch performante Ergebnisse ohne, dass der Entwickler den idealen Zeitpunkt für die Refreshes genau kennen muss.

Statement-Level Refresh
In der Praxis sind dem Autor keine Fälle bekannt, in denen die Option `refresh ON COMMIT` genutzt wird. Die Neuerung zielt darauf ab, dass eine DML Operation nun nicht einmal commited sein muss und trotzdem schon in die MV übertragen wird. Die Dokumentation erwähnt einen vorrangigen Anwendungsfall bei Star Schema Deployments. Genutzt werden kann das Feature ganz einfach indem im bereits bekannten Header folgendes angegeben wird:

```
CREATE MATERIALIZED VIEW schema.mv_name
REFRESH FAST ON STATEMENT USING TRUSTED CONSTRAINT
AS (select --…
```

Bemerkenswert ist die Tatsache, dass die auch ein zeitintensives Rollback in einer Transaktion zeitgleich auch auf die MV angewendet werden muss! Wie beim on commit Verhalten kann das signifikante Auswirkungen auf das jeweilige DML Statement haben, weil die MV Refreshes Teil der Transaktion sind. Oracle legt hierzu automatisch einen Index auf ROWID Spalte an.

Neue Data Dictionary Views
In verschiedenen Detail Stufen lassen sich nun die durchgeführten Refreshes analysieren. Hierzu stehen im Data Dictionary die folgenden neuen Views bereit. Sie enthalten je nach Einstellung (`DBMS_MVIEW_STATS.SET_MVREF_STATS_PARAMS` Prozedur) verschiedene historische Daten vor. Problematische Vorgänge lassen sich somit klar ermitteln und ausbessern.
Namentlich sind das:

- `USER_MVREF_STATS_SYS_DEFAULTS`
- `USER_MVREF_STATS_PARAMS`
- `USER_MVREF_STATS`
- `DBA_MVREF_RUN_STATS`
- `DBA_MVREF_CHANGE_STATS`
- `DBA_MVREF_STMT_STATS`

Besonders die letzte View eignet sich zum Tunen, da man hier sowohl die SQLID als auch den Ausführungsplan eines jeden Vorgangs aufzeichnet!

Fazit
Der Vortrag zeigt anhand klarer Praxisbeispiele und realistischer Zahlen, wann und weshalb sich der Einsatz von MVs 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. Entsprechende Einschränkungen beim Verwenden der Features können Sie der Dokumentation entnehmen. Der Vortrag erwähnt einige von ihnen.

Kontaktadresse:
Jonas Gassenmeyer
Syntegris information solutions 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