mvdbs: Updatable Materialized View

Tobias Lerch, Yanick Eberle, Pascal Schwarz

28. April 2013

1 Einleitung

Ihre Lösung muss aus folgenden Teilen bestehen:
• Eine Beschreibung Ihrer Szenarios –¿ meinsch da esch guet so?

In dieser Übung geht es darum, eine Replikation einzurichten und anschliessend die Auswirkungen und das Verhalten der Replikation genauer unter die Lupe zu nehmen.

Wir erstellen auf dem Telesto Server eine Master Group und definieren die Relation Filialen als Replikationsobjekt, welche anschliessend der Master Group hinzugefügt wird. Um dieses anschliessend replizieren zu können, müssen noch Trigger und Packages erstellt werden und alles muss in den Replikationsprozess aufgenommen werden.

Als Gegenstück zur Master Group auf Telesto erstellen wir auf auf dem Ganymed Server eine Materialized View Group, in welche ebenfalls die Relation Filiale als Replikationsobjekt aufgenommen wird. Anschliessend definieren eine Refresh Gruppe und fügen die Materialized View hinzu, damit die Änderungen auch repliziert werden.

Soblad alles eingerichtet ist, werden die Tests ausgeführt und analysiert.

2 Replikation einrichten

Die Master Site und die Materialized View Site wurde bereits eingerichtet, somit müssen nur noch die jeweiligen Gruppen erstellt werden.

2.1 Erstellen der Master Group

Wir verbinden uns als Benutzer repadmin auf den Telesto Server und führen folgende SQL Statements aus.

```
1 BEGIN
2         DBMS_REPCAT.CREATE_MASTER_REPGROUP (
3         gname => 'mvdbs10_repg');
4 END;
```

Das Resultat des SQL Developers ist ein einfaches „anonymer Block abgeschlossen“. Somit ist die Master Gruppe erstellt.

```
1 BEGIN
2 DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
3 gname => 'mvdbs10_repg',
4 type => 'TABLE',
5 oname => 'filialen',
6 sname => 'mvdbs10',
7 use_existing_object => TRUE,
8 Copy_rows => FALSE);
9 END;
```

Die Relation Filialen ist nun ein Replikationsobjekt und wird der Master Gruppe hinzugefügt.

```
1 BEGIN
2 DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
3 sname => 'mvdbs10',
4 oname => 'filialen',
5 type => 'TABLE',
6 min_communication => TRUE);
7 END;
```

Dieses Statement erstellt die Trigger und Packages, welche für die Replikation gebraucht werden.

```
1 BEGIN
2 DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
3 gname => 'mvdbs10_repg');
4 END;
```

Die Änderungen werden in den Replikationsprozess aufgenommen.

2.2 Erstellen der Materialized View Group

Wir verbinden uns als Benutzer mvdbs10 auf den Telesto Server und führen folgende SQL Statements aus.

```
1 CREATE MATERIALIZED VIEW LOG ON mvdbs10.filialen;
```

Auf Telesto wurde nun die Materialized View erstellt und mit „materialized view LOG erstellt.“bestätigt.

```
1 CREATE DATABASE LINK telesto.janus.fhnw.ch
2 CONNECT TO proxy_refresher IDENTIFIED BY &password;
```

Der Database Link wird als Benutzer mvdbs10 auf dem Server ganymed erstellt.

```
1 BEGIN
2 DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
3 gname => 'mvdbsx2_repg',
4 master => 'telesto.janus.fhnw.ch',
5 propagation_mode => 'ASYNCHRONOUS');
6 END;
```

Dieses Statement erstellt eine neue Materialized View Group.

```
1  BEGIN
2  DBMS_REFRESH.MAKE (
3  name => 'mviewadmin.mvdbs10_refg',
4  list => '',
5  next_date => SYSDATE,
6  interval => 'SYSDATE + 1/24',
7  implicit_destroy => FALSE,
8  rollback_seg => '',
9  push_deferred_rpc => TRUE,
10 refresh_after_errors => FALSE);
11 END;
```

Es wird eine Refresh Gruppe erstellt, welche einen stündlichen refresh definiert.

```
1  CREATE MATERIALIZED VIEW mvdbs10.filialen
2  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
3  AS SELECT * FROM mvdbs10.filialen@telesto.janus.fhnw.ch;
```

Als Benutzer mvdbs10 auf dem Server ganymed wird die Materialized View erstellt.

```
1  BEGIN
2  DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
3  gname => 'mvdbs10_repg',
4  sname => 'mvdbs10',
5  oname => 'filialen',
6  type => 'SNAPSHOT',
7  min_communication => TRUE);
8  END;
```

Als Benutzer mviewadmin auf ganymed wird die Relation Filialen als Replikationsobjekt zu der Materialized View Group hinzugefügt.

```
1  BEGIN
2  DBMS_REFRESH.ADD(
3  name => 'mviewadmin.mvdbs10_refg',
4  list => 'mvdbs10.filialen',
5  lax => TRUE);
6  END;
```

Die Materialized View wird zur Refresh Grupppe hinzugefügt.

```
1  BEGIN
2  DBMS_REFRESH.REFRESH (
3  name => 'mviewadmin.mvdbs10_refg' );
4  end;
```

Mit diesem Statement kann der Refresh direkt ausgeführt werden.

3 Testszenarien

3.1 Ohne Konflikt

3.1.1 updates

Um zu überprüfen, ob updates korrekt repliziert werden, erstellen wir ein Query für die Master Site und ein zweites Query für die Materialized View Site, wobei diese unterschiedliche Daten verändern. Dabei sollte kein Konflikt auftreten. Nach der Replikation sollte die Tabelle Filiale auf beiden Sites identisch sein.

Folgendes Query wird auf der Master Site ausgeführt (ab Zeile 5 Output):

```
1 UPDATE filialen
2 SET ort='Brugg'
3 WHERE ort='Basel' and fnr='F4';
4
5 1 Zeilen aktualisiert.
```

Auf der Materialized View Site erstellen wir ebenfalls ein Query (ab Zeile 5 Output):

```
1 UPDATE filialen
2 SET ort='Brugg'
3 WHERE ort='Basel' and fnr='F1';
4
5 1 Zeilen aktualisiert.
```

Nun untersuchen wir die beiden Logs.

Nach der Ausführung des Querys auf der Master Site sehen wir mit folgendem Befehl die Einträge im Log:

```
1 SELECT * FROM MLOG$_FILIALEN ;
```

Im Log auf der Master Site ist ersichtlich, dass sich der Eintrag mit dem Primary Key FNR F4 geändert hat. Ebenfalls wurde eine CHANGE_VECTOR und eine XID generiert, welche anschliessend für die Replikation verwendet wird. Der Zeitstempel bei SNAPTIME ist noch mit dem Default Wert abgefüllt. Es wird nur verwendet, wenn mehrere Snapshots auf einem Master definier sind. Da wir nur eine Materialized View haben, wird dieses Feld nicht verwendet. Der Eintrag 'U' unter DMLTYPE und OLD_NEW zeigt, dass es sich bei der Änderung um ein Update handelt.

```
1 FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$ CHANGE_VECTOR$$ XID$$
2 ——— ————————— ————————— ————————— ——————————————— ————————
3 F4  01.01.00   U          U         04               1.7E+15
```

Auch auf der Materialized View Site kann nach dem Ausführen des Querys das Log eingesehen werden:

```
1 SELECT * FROM USLOG$_FILIALEN ;
```

Hier ist im Log ersichtlich, dass sich der Eintrag mit dem Primary Key FNR F1 geändert hat. Auf der Materialized View Site wird kein CHANGE_VECTOR und XID generiert.

```
1 FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$
2 ——— ————————— ————————— ————————
3 F1  01.01.00   U          U
```

Wir geben die Änderungen für die Replikation frei, indem wir auf beiden Sites ein commit ausführen (ab Zeile 3 Output):

```
1 COMMIT
2
3 festgeschrieben.
```

Nachdem die Replikation durchgeführt wurde, betrachten wir die Logs erneut.

Der Eintrag im Log ist verschwunden. Das heisst, die Einträge bleiben nur gespeichert, bis die Replikation durchgeführt wurde. Jede Zeile im Log bedeutet, dass eine Zeile geändert wurde und repliziert werden muss. Bei der Replikation kann mithilfe des Logs die Änderung auf der Materialized View vorgenommen werden.

```
1 Keine Zeilen gewaehlt
```

Das gleiche gilt für das Log auf der Materialized View Site.

```
1 Keine Zeilen gewaehlt
```

3.1.2 inserts

Um zu überprüfen, ob inserts korrekt repliziert werden, erstellen wir ein Query für die Master Site und ein zweites Query für die Materialized View Site, wobei diese unterschiedliche Daten einfügt. Dabei sollte kein Konflikt auftreten. Nach der Replikation sollte die Tabelle Filiale auf beiden Sites identisch sein.

Folgendes Query wird auf der Master Site ausgeführt (ab Zeile 3 Output):

```
1 INSERT INTO filialen (fnr, ort, plz) values ('F5', 'Genf', 9000);
2
3 1 Zeilen eingefuegt.
```

Auf der Materialized View Site erstellen wir ebenfalls ein Query (ab Zeile 3 Output):

```
1 INSERT INTO filialen (fnr, ort, plz) values ('F6', 'Genf', 9000);
2
3 1 Zeilen eingefuegt.
```

Nun untersuchen wir die beiden Logs.

Nach der Ausführung des Querys auf der Master Site sehen wir mit folgendem Befehl die Einträge im Log:

```
1 SELECT * FROM MLOG$_FILIALEN ;
```

Wir sehen dass der Eintrag mit dem Primary Key F5 geändert wurde. Das I bei DMLTYPE steht für insert. Der Wert N bei OLD_NEW bedeutet, es ist ein neuer Wert dazugekommen. Wiederum wurde ein CHANGE_VECTOR und eine XID generiert.

```
1 FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$ CHANGE_VECTOR$$ XID$$
2 ——— ———————— ————————— ————————— ——————————————— ————————
3 F5  01.01.00   I          N          FE              2.5E+15
```

Auch auf der Materialized View Site kann nach dem Ausführen des Querys das Log eingesehen werden:

```
1 SELECT * FROM USLOG$_FILIALEN ;
```

Hier ist im Log ersichtlich, dass sich der Eintrag mit dem Primary Key FNR F6 geändert hat. Es handelt sich ebenfalls um ein Insert.

```
1 FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$
2 ——— ———————— ————————— ————————
3 F6  01.01.00   I          N
```

Wir geben die Änderungen für die Replikation frei, indem wir auf beiden Sites ein commit ausführen (ab Zeile 3 Output):

```
1 COMMIT
2
3 festgeschrieben.
```

Wie bereits beim Update beschrieben ist nach der Replikation nichts mehr im Log.

```
1 Keine Zeilen gewaehlt
```

3.1.3 deletes

Um zu überprüfen, ob deletes korrekt repliziert werden, erstellen wir ein Query für die Master Site und ein zweites Query für die Materialized View Site, wobei diese unterschiedliche Daten löschen. Dabei sollte kein Konflikt auftreten. Nach der Replikation sollte die Tabelle Filiale auf beiden Sites identisch sein.

Folgendes Query wird auf der Master Site ausgeführt (ab Zeile 5 Output):

```
1 DELETE FROM filialen
2 WHERE ort='Genf' and fnr='F5';
```

Auf der Materialized View Site erstellen wir ebenfalls ein Query (ab Zeile 5 Output):

```
1 DELETE FROM filialen
2 WHERE ort='Genf' and fnr='F6';
```

Nun untersuchen wir die beiden Logs.

Nach der Ausführung des Querys auf der Master Site sehen wir mit folgendem Befehl die Einträge im Log:

```
1 SELECT * FROM MLOG$_FILIALEN ;
```

Wir sehen, dass der Eintrag mit dem Primary Key F5 geändert wurde. Das D bei DMLTYPE steht für delete. Der Wert O bei OLD_NEW bedeutet, es wurde ein alter Wert gelöscht. Wiederum wurde ein CHANGE_VECTOR und eine XID generiert.

```
1 FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$ CHANGE_VECTOR$$ XID$$
2 ——— ———————— ————————— ————————— ——————————————— ———————
3 F5  01.01.00   D          O         00               2.0E+15
```

Auch auf der Materialized View Site kann nach dem Ausführen des Querys das Log eingesehen werden:

```
1 SELECT * FROM USLOG$_FILIALEN ;
```

Hier ist im Log ersichtlich, dass sich der Eintrag mit dem Primary Key FNR F6 geändert hat. Es handelt sich ebenfalls um ein Delete.

```
1  FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$
2  ——— ———————— ———————— ————————
3  F6  01.01.00   D         O
```

Wir geben die Änderungen für die Replikation frei, indem wir auf beiden Sites ein commit ausführen (ab Zeile 3 Output):

```
1  COMMIT
2
3  festgeschrieben.
```

Wie bereits beim Update beschrieben ist nach der Replikation nichts mehr im Log.

```
1  Keine Zeilen gewaehlt
```

3.2 Mit Konflikt

3.2.1 updates

3.2.2 inserts

3.2.3 deletes

3.3 Regel zur Konfliktauflösung

3.4 Mit Konflikt und Konfliktauflösung

3.4.1 updates

3.4.2 inserts

3.4.3 deletes