mvdbs: Updatable Materialized View

Tobias Lerch, Yanick Eberle, Pascal Schwarz

28. April 2013

1 Einleitung

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 wir eine Refresh Gruppe und fügen die Materialized View hinzu, damit die Änderungen auch repliziert werden.

Sobald 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 im SQL Developer ist ein einfaches „anonymer Block abgeschlossen“. Somit ist die Master Gruppe erstellt.

```
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
gname => 'mvdbs10_repg',
type => 'TABLE',
oname => 'filialen',
sname => 'mvdbs10',
use_existing_object => TRUE,
Copy_rows => FALSE);
END;
```

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

```
BEGIN
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
sname => 'mvdbs10',
oname => 'filialen',
type => 'TABLE',
min_communication => TRUE);
END;
```

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

```
BEGIN
DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
gname => 'mvdbs10_repg');
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.

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

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

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

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

```
BEGIN
DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
gname => 'mvdbsx2_repg',
master => 'telesto.janus.fhnw.ch',
propagation_mode => 'ASYNCHRONOUS');
END;
```

Dieses Statement erstellt eine neue Materialized View Group.

```
BEGIN
DBMS_REFRESH.MAKE (
name => 'mviewadmin.mvdbs10_refg',
list => '',
next_date => SYSDATE,
interval => 'SYSDATE + 1/24',
implicit_destroy => FALSE,
rollback_seg => '',
push_deferred_rpc => TRUE,
refresh_after_errors => FALSE);
END;
```

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

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

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

```
BEGIN
DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
gname => 'mvdbs10_repg',
sname => 'mvdbs10',
oname => 'filialen',
type => 'SNAPSHOT',
min_communication => TRUE);
END;
```

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

```
BEGIN
DBMS_REFRESH.ADD(
name => 'mviewadmin.mvdbs10_refg',
list => 'mvdbs10.filialen',
lax => TRUE);
END;
```

Die Materialized View wird zur Refresh Grupppe hinzugefügt.

```
BEGIN
DBMS_REFRESH.REFRESH (
name => 'mviewadmin.mvdbs10_refg' );
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 Filialen 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. Er wird nur verwendet, wenn mehrere Snapshots auf einem Master definiert 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 Filialen 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 Filialen 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, dass ein alter Wert gelöscht wurde. 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:

```
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.

```
FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$
--- ---------- --------- ---------
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):

```
COMMIT

festgeschrieben.
```

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

```
Keine Zeilen gewaehlt
```

3.2 Mit Konflikt

3.2.1 Updates

Um herauszufinden, wie sich die Replikation verhält wenn Konflikte auftreten, erstellen wir für beide Sites dasselbe SQL Query, welches die selben Zeilen und die selben Werte verändert. Bei der Replikation sollte ein Konflikt auftreten.

SQL Query Master Site und Materialized View (ab Zeile 5 Output):

```
UPDATE filialen
SET ort='Brugg'
WHERE fnr='F1';

1 Zeilen aktualisiert.
```

Nun steht in beiden Logs dasselbe.

Die Zeile mit dem Primary Key F1 wurde geändert mit einem Update.

```
FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$ CHANGE_VECTOR$$ XID$$
--- ---------- --------- --------- ---------------- ----------
F1  01.01.00   U         U         04               3.1E+15
```

```
FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$
--- ---------- --------- ---------
F1  01.01.00   U         U
```

Wiederum geben wir die Änderungen für die Replikation frei:

```
COMMIT

festgeschrieben.
```

Wenn wir nun die Replikation manuell starten, erhalten wir folgende Meldung:

```
Fehlerbericht:
ORA-23402: Wegen Konflikten bei verzoegerten Transaktionen wurde die Aktualisierung
    abgebrochen
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2558
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2771
ORA-06512: in "SYS.DBMS_IREFRESH", Zeile 685
ORA-06512: in "SYS.DBMS_REFRESH", Zeile 195
ORA-06512: in Zeile 2
23402. 00000 - "refresh was aborted because of conflicts caused by deferred txns"
*Cause:    There are outstanding conflicts logged in the DefError table
           at the materialized view's master.
*Action:   Resolve the conflicts in the master DefError table and
           refresh again after the table is empty.  Alternatively,
           refresh with refresh_after_errors set to TRUE, which
           will proceed with the refresh even if there are conflicts
           in the master's DefError table.  Proceeding despite conflicts
           can result with an updatable materialized view's changes appearing
           to be temporarily lost (until a refresh succeeds after the
           conflicts are resolved).
```

Wir schauen uns nun die Tabelle DefErrors genauer an:

```
SELECT * FROM DefError
```

Wir können erkennen, dass ein Konflikt zwischen Ganymed und Telesto vorhanden ist, welcher am 28.04.2013 erzeugt wurde. Ebenfalls sieht man die Fehlernummer und die Fehlermeldung (leider ist die Darstellung nicht überschtlich, da die Zeile zu breit ist).

```
DEFERRED_TRAN_ID ORIGIN_TRAN_DB ORIGIN_TRAN_ID CALLNO DESTINATION START_TIME
    ERROR_NUMBER ERROR_MSG   RECEIVER
---------------- ------------- -------------- ------- ------------- ----------
    ------------ ------------ ----------
8.4.89471   GANYMED.SIRIUS.FHNW.CH   7.31.97249   0 TELESTO.JANUS.FHNW.CH   28.04.13
    100 ORA-01403: Keine Daten gefunden   REPADMIN
```

Da wir später eine Regel zur Konfliktauflösung definieren, brechen wir hier ab. Wir haben gesehen, dass ein Konflikt auftritt.

3.2.2 Inserts

Um herauszufinden, wie sich die Replikation verhält wenn Konflikte auftreten, erstellen wir für beide Sites dasselbe SQL Query, welches die selben Zeilen und die selben Werte einfügt. Bei der Replikation sollte ein Konflikt auftreten.

SQL Query Master Site und Materialized View (ab Zeile 3 Output):

```
INSERT INTO filialen (fnr, ort, plz) values ('F6', 'Genf', 9000);

1 Zeilen eingefuegt.
```

Nun steht in beiden Logs dasselbe.

Die Zeile mit dem Primary Key F6 wurde geändert mit einem Insert und neuen Werten.

```
FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$ CHANGE_VECTOR$$ XID$$
--- ---------- --------- --------- ---------------- ----------
F6  01.01.00   I         N         FE               3.1E+15
```

```
FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$
--- ---------- ---------- ----------
F6  01.01.00   I          N
```

Wiederum geben wir die Änderungen für die Replikation frei:

```
COMMIT

festgeschrieben.
```

Wenn wir nun die Replikation manuell starten, erhalten wir folgende Meldung:

```
Fehlerbericht:
ORA-23402: Wegen Konflikten bei verzoegerten Transaktionen wurde die Aktualisierung
    abgebrochen
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2558
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2771
ORA-06512: in "SYS.DBMS_IREFRESH", Zeile 685
ORA-06512: in "SYS.DBMS_REFRESH", Zeile 195
ORA-06512: in Zeile 2
23402. 00000 -  "refresh was aborted because of conflicts caused by deferred txns"
*Cause:    There are outstanding conflicts logged in the DefError table
           at the materialized view's master.
*Action:   Resolve the conflicts in the master DefError table and
           refresh again after the table is empty.  Alternatively,
           refresh with refresh_after_errors set to TRUE, which
           will proceed with the refresh even if there are conflicts
           in the master's DefError table.  Proceeding despite conflicts
           can result with an updatable materialized view's changes appearing
           to be temporarily lost (until a refresh succeeds after the
           conflicts are resolved).
```

Hier geschieht also genau dasselbe, weil beide eine Zeile mit demselben Primary Key einfügen wollen. Auch hier betrachten wir den Eintrag in der DefError Tabelle. Es ist ersichtlich, dass der Konflikt auftritt, weil ein Constraint verletzt wurde.

```
DEFERRED_TRAN_ID ORIGIN_TRAN_DB ORIGIN_TRAN_ID CALLNO DESTINATION START_TIME
    ERROR_NUMBER ERROR_MSG   RECEIVER
---------------- ------------- ------------- ------- ------------- ----------
    ----------- ----------- ----------
2.11.89518   GANYMED.SIRIUS.FHNW.CH  2.7.120248  0 TELESTO.JANUS.FHNW.CH   28.04.13
    -1 ORA-00001: Unique Constraint (MVDBS10.FL_PK) verletzt  REPADMIN
```

Da wir später eine Regel zur Konfliktauflösung definieren, brechen wir hier ab. Wir haben gesehen, dass ein Konflikt auftritt.

3.2.3 Deletes

Um herauszufinden, wie sich die Replikation verhält wenn Konflikte auftreten, erstellen wir für beide Sites dasselbe SQL Query, welches dieselbe Zeile und dieselben Werte löscht. Bei der Replikation sollte ein Konflikt auftreten.

SQL Query Master Site und Materialized View (ab Zeile 4 Output):

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

1 Zeilen geloescht.
```

Nun steht in beiden Logs dasselbe.

Die Zeile mit dem Primary Key F6 wurde geändert mit einem Insert und neuen Werten.

```
1 FNR SNAPTIME$$ DMLTYPE$$ OLD_NEW$$ CHANGE_VECTOR$$ XID$$
2 ——— ———————— ————————— ————————— ——————————————— ————————
3 F6  01.01.00   D          O         00              3.1E+15
```

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

Wiederum geben wir die Änderungen für die Replikation frei:

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

Wenn wir nun die Replikation manuell starten, erhalten wir folgende Meldung:

```
1  Fehlerbericht:
2  ORA-23402: Wegen Konflikten bei verzoegerten Transaktionen wurde die Aktualisierung
       abgebrochen
3  ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2558
4  ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2771
5  ORA-06512: in "SYS.DBMS_IREFRESH", Zeile 685
6  ORA-06512: in "SYS.DBMS_REFRESH", Zeile 195
7  ORA-06512: in Zeile 2
8  23402. 00000 -  "refresh was aborted because of conflicts caused by deferred txns"
9  *Cause:    There are outstanding conflicts logged in the DefError table
10            at the materialized view's master.
11 *Action:   Resolve the conflicts in the master DefError table and
12            refresh again after the table is empty.  Alternatively,
13            refresh with refresh_after_errors set to TRUE, which
14            will proceed with the refresh even if there are conflicts
15            in the master's DefError table.  Proceeding despite conflicts
16            can result with an updatable materialized view's changes appearing
17            to be temporarily lost (until a refresh succeeds after the
18            conflicts are resolved).
```

Hier geschieht also genau dasselbe, weil beide eine Zeile mit demselben Primary Key löschen wollen. Auch hier betrachten wir den Eintrag in der DefError Tabelle. Es ist ersichtlich, dass der Konflikt auftritt, weil ein Constraint verletzt wurde.

```
1 DEFERRED_TRAN_ID ORIGIN_TRAN_DB ORIGIN_TRAN_ID CALLNO DESTINATION START_TIME
      ERROR_NUMBER ERROR_MSG   RECEIVER
2 ————————————————— ————————————— ————————————— ——————— ————————————— ——————————————
      ———————————— ————————————— ————————————
3 8.20.89509    GANYMED.SIRIUS.FHNW.CH  10.3.97163   0 TELESTO.JANUS.FHNW.CH    28.04.13
      100 ORA-01403: Keine Daten gefunden  REPADMIN
```

Da wir später eine Regel zur Konfliktauflösung definieren, brechen wir hier ab. Wir haben gesehen, dass ein Konflikt auftritt.

3.3 Regel zur Konfliktauflösung

Wir verbinden uns mit dem Benutzer repadmin auf den Server Telesto und führen folgende Skripts aus, um die Konfliktauflösung mittels *Overwrite* zu aktivieren. Quelle: `http://docs.oracle.com/cd/B28359_01/server.111/b28327/rarconflictres.htm`

Zuerst müssen wir die Replikationsgruppe inaktiv setzen, damit wir Änderungen vornehmen können an der Konfiguration.

```
BEGIN
    DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
        gname => 'mvdbs10_repg');
END;
```

Anschliessend müssen wir eine Column Group erstellen, da wir diese benötigen um das Overwrite zu aktivieren:

```
BEGIN
    DBMS_REPCAT.MAKE_COLUMN_GROUP (
        sname => 'mvdbs10',
        oname => 'filialen',
        column_group => 'dep_cg',
        list_of_column_names => 'fnr');
END;
```

Nun können wir die Konfliktauflösung definieren. Wir erstellen eine Overwrite Konfliktauflösung.

```
BEGIN
    DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
        sname => 'mvdbs10',
        oname => 'filialen',
        column_group => 'dep_cg',
        sequence_no => 1,
        method => 'OVERWRITE',
        parameter_column_name => 'fnr');
END;
```

Nun müssen wir den Replikationssupport neu generieren.

```
BEGIN
     DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
        sname => 'mvdbs10',
        oname => 'filialen',
        type => 'TABLE',
        min_communication => TRUE);
END;
```

Und zuletzt muss die Replikationsgruppe wieder aktiviert werden.

```
BEGIN
    DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
        gname => 'mvdbs10_repg');
END;
```

3.4 Mit Konflikt und Konfliktauflösung

3.4.1 Updates

Wir machen genau dasselbe wie oben mit Konflikt.

Wenn wir nun die Replikation manuell starten, erhalten wir folgende Meldung:

```
anonymer Block abgeschlossen
```

Es scheint als hat alles funktioniert. In der Tabelle DefError ist keine Meldung. Die Änderungen wurden übernommen. Da wir aber auf beiden Sites dasselbe Query ausgeführt haben, sehen wir nicht welche Änderung übernommen wurde. Daher erstellen wir nun zwei unterschiedliche Querys, welche die gleiche Zeile anpassen, aber unterschiedliche Werte hineinschreiben:

Master Site Query:

```
1 UPDATE filialen
2 SET ort='Brugg'
3 WHERE fnr='F1';
```

Materialized View Site Query:

```
1 UPDATE filialen
2 SET ort='Biel'
3 WHERE fnr='F1';
```

Wir führen zuerst das Query auf der Master Site aus und schliessen es ab mit 'commit'. Anschliessend führen wir das Query auf der Materialized View Site aus und bestätigen dies ebenfalls mit 'commit'. Wenn wir die Tabelle Filialen anschauen, sehen wir, dass der Wert Biel übernommen wurde. Dieses Resultat entspricht unserer Konfliktauflösung, da wir Overwrite definiert haben. Der letzte Commit überschreibt die vorhergehenden Änderungen.

```
1 FNR ORT          PLZ
2 ——— ———————— ————
3 F1   Biel        4056
4 F3   Bern        3014
5 F2   Zuerich      8050
6 F4   Basel       4002
```

Die Logs der beiden Views sehen vor der Replikation genau gleich aus wie vorher, als wir die Konfliktauflösung noch nicht definiert haben.

3.4.2 Inserts

Wir machen genau dasselbe wie oben mit Konflikt.

Auch hier erstellen wir wieder zwei unterschiedliche Querys, welche Werte mit dem selben Primary Key einfügen.

Master Site Query:

```
1 INSERT INTO filialen (fnr, ort, plz) values ('F6', 'Grenchen', 9000);
```

Materialized View Site Query:

```
1 INSERT INTO filialen (fnr, ort, plz) values ('F6', 'Genf', 9000);
```

Auch hier führen wir zuerst das Query auf der Master Site aus, schliessen es mit commit ab und machen anschliessend dasselbe auf der Materialized View Site.

Bei der Replikation erhalten wir aber einen Fehler:

```
Fehlerbericht:
ORA-23402: Wegen Konflikten bei verzoegerten Transaktionen wurde die Aktualisierung
    abgebrochen
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2558
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2771
ORA-06512: in "SYS.DBMS_IREFRESH", Zeile 685
ORA-06512: in "SYS.DBMS_REFRESH", Zeile 195
ORA-06512: in Zeile 2
23402. 00000 - "refresh was aborted because of conflicts caused by deferred txns"
*Cause:    There are outstanding conflicts logged in the DefError table
           at the materialized view's master.
*Action:   Resolve the conflicts in the master DefError table and
           refresh again after the table is empty.  Alternatively,
           refresh with refresh_after_errors set to TRUE, which
           will proceed with the refresh even if there are conflicts
           in the master's DefError table.  Proceeding despite conflicts
           can result with an updatable materialized view's changes appearing
           to be temporarily lost (until a refresh succeeds after the
           conflicts are resolved).
```

Wir schauen uns den Eintrag in der DefError Tabelle an und erkennen, dass es genau derselbe Fehler ist wie vor der Konfliktauflösung. Dies erklärt sich, da ein Unique Constraint gesetzt ist und ein Insert mit demselben Primary Key deshalb nicht möglich ist. Das Overwrite funktioniert hier also nicht, da der Constraint stärker gewichtet ist als das Overwrite der Replikation.

```
DEFERRED_TRAN_ID ORIGIN_TRAN_DB ORIGIN_TRAN_ID CALLNO DESTINATION START_TIME
    ERROR_NUMBER ERROR_MSG   RECEIVER
---------------- ------------- ------------- ------- ------------ ----------
    ------------ ------------ ----------
2.11.89518   GANYMED.SIRIUS.FHNW.CH  2.7.120248  0 TELESTO.JANUS.FHNW.CH   28.04.13
    -1 ORA-00001: Unique Constraint (MVDBS10.FL_PK) verletzt  REPADMIN
```

Nun erstellen wir eine Regel für die Konfliktlösung von INSERT Konflikten.

```
BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION (
       sname => 'mvdbs10',
       oname => 'filialen',
       constraint_name => 'FL_PK',
       sequence_no => 1,
       method => 'DISCARD',
       parameter_column_name => 'fnr');
END;
```

Bei den Uniqueness Konfliktlösungen gibt es drei Möglichkeiten.

APPEND SITE NAME Beim Primary Key wird der Name der Site hinzugefügt. Voraussetzung ist das der Typ der Spalte CHAR oder VARCHAR2 ist.

APPEND SEQUENCE Beim Primary Key wird eine generierte Sequenznummer hinzugefügt. Voraussetzung ist das der Typ der Spalte CHAR oder VARCHAR2 ist.

DISCARD Der Konflikt mit dem Constraint wird ignoriert und die Zeile nicht eingefügt.

In unserem Fall bedeutet dies das der INSERT der Master Site eingetragen und repliziert wird. Wir erhalten keinen Replikationsfehler mehr, jedoch wird nur der erste Eintrag gespeichert und der zweite einfach ignoriert. Keine schöne, aber eine Konfliktfreie Lösung.

```
FNR ORT          PLZ
--- ------------ ----
F6  Grenchen     9000
F1  Biel         4056
F3  Bern         3014
F2  Zurich       8050
F4  Basel        4002
```

3.4.3 Deletes

Wir machen genau dasselbe wie oben mit Konflikt.

Auch hier erstellen wir wieder Queries, welche die Zeile mit demselben Primary Key löschen.

Master Site Query:

```
DELETE FROM filialen
WHERE ort='Grenchen' and fnr='F6';
```

Materialized View Site Query:

```
DELETE FROM filialen
WHERE ort='Grenchen' and fnr='F6';
```

Auch hier führen wir zuerst das Query auf der Master Site aus, schliessen es mit commit ab und machen anschliessend dasselbe auf der Materialized View Site.

Bei der Replikation erhalten wir aber einen Fehler:

```
Fehlerbericht:
ORA-23402: Wegen Konflikten bei verzoegerten Transaktionen wurde die Aktualisierung
    abgebrochen
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2558
ORA-06512: in "SYS.DBMS_SNAPSHOT", Zeile 2771
ORA-06512: in "SYS.DBMS_IREFRESH", Zeile 685
ORA-06512: in "SYS.DBMS_REFRESH", Zeile 195
ORA-06512: in Zeile 2
23402. 00000 -  "refresh was aborted because of conflicts caused by deferred txns"
*Cause:    There are outstanding conflicts logged in the DefError table
           at the materialized view's master.
*Action:   Resolve the conflicts in the master DefError table and
           refresh again after the table is empty.  Alternatively,
           refresh with refresh_after_errors set to TRUE, which
           will proceed with the refresh even if there are conflicts
           in the master's DefError table.  Proceeding despite conflicts
           can result with an updatable materialized view's changes appearing
           to be temporarily lost (until a refresh succeeds after the
           conflicts are resolved).
```

Wir schauen uns den Eintrag in der DefError Tabelle an und erkennen, dass es genau der selbe Fehler ist wie vor der Konfliktauflösung. Dies erklärt sich, da nach dem Ausführen des ersten Querys die Zeile mit dem Primary Key F6 nicht mehr vorhanden ist und daher keine Daten gefunden wurden. Das Overwrite der Replikation kann nicht ausgeführt werden, da der Datensatz nicht gefunden wird.

```
DEFERRED_TRAN_ID ORIGIN_TRAN_DB ORIGIN_TRAN_ID CALLNO DESTINATION START_TIME
    ERROR_NUMBER ERROR_MSG   RECEIVER
---------------- -------------- ------------- ------- ------------- ----------
    ------------ ------------ ----------
1.32.84171   GANYMED.SIRIUS.FHNW.CH  6.12.129681  0 TELESTO.JANUS.FHNW.CH   28.04.13
    100 ORA-01403: Keine Daten gefunden  REPADMIN
```

Oracle bietet auch für DELETE Konflikte ein Prozedur an. Es gibt aber von Oracle keine Delete Methoden. Das bedeutet die Methode muss vom DB Administrator selbst geschrieben werden. Das hinzufügen einer Delete Konfliktlösung ist gleich wie bei Update und Insert.

```
1 DBMS_REPCAT.ADD_DELETE_RESOLUTION (
2     sname                       IN    VARCHAR2,
3     oname                       IN    VARCHAR2,
4     sequence_no                 IN    NUMBER,
5     parameter_column_name       IN    VARCHAR2 | DBMS_REPCAT.VARCHAR2s,
6     function_name               IN    VARCHAR2,
7     comment                     IN    VARCHAR2       := NULL
8     method                      IN    VARCHAR2       := 'USER FUNCTION');
```

Quelle: `http://docs.oracle.com/cd/B28359_01/server.111/b28327/rarrcatpac.htm#i94790`

3.4.4 Ergebnis

Wird als Konfliktlösung die Methode Overwrite gewählt, werden bei UPDATE Statements mit Konflikten jeweils der Eintrag gespeichert der zuletzt ausgeführt wurde. Bei der INSERT Konfliktlösung muss einem bewusst werden was die Auswirkungen der Lösung auf die Primary Keys sind. Eine Delete Konfliktlösung ist zwar möglich, aber sie muss selbst implementiert werden was je nach komplexität einen grossen Aufwand bedeutet.