Bidirektionale Replikation für InterBase und Firebird

Der Open Source Datenbankserver Firebird[1] und sein kommerzieller Bruder Borland InterBase[2] sind seit Jahren eine bewährte und stabile Plattform für Datenbankanwendungen aller Art. Aufgrund des gemeinsamen Vorfahren in Form des Quellcodes von InterBase 6, lassen sich viele Lösungen auf beiden Plattformen problemlos benutzen. Trotzdem hat gerade in letzter Zeit der Firebird Server mit der Version 2 neue Maßstäbe gesetzt und viele hilfreiche Funktionen eingeführt, die man bei InterBase 7.5 noch vermisst. Bei beiden Plattformen ist jedoch eine Replikation nicht im Lieferumfang. Der folgende Artikel zeigt jedoch, wie man mit Bordmitteln und unter Mithilfe des Programms IBExpert[3] eine solche Replikation erstellen kann.

Was bedeutet Replikation?

Wikipedia liefert eine kurze und griffige Erklärung: "Eine Replikation ist eine Verdopplung bzw. Vervielfältigung von Daten. Der Datenbestand der replizierten Daten ist in der Regel mit dem Original identisch". Man unterscheidet jedoch zwischen synchroner Replikation und asynchroner Replikation. Während die synchrone Replikation dafür sorgt, dass ein Datenbankserver im Fehlerfall durch einen replizierten Backupserver umgehend ersetzt werden kann und die Anwender ohne Unterbrechung weiterarbeiten können, ist eine asynchrone Replikation sinnvoll, wenn es um den Datenbestand von Rechnern geht, die sich nicht oder nicht immer im selben Netzwerk befinden. Typischerweise wird eine asynchrone Replikation für die Laptops von Außendienstmitarbeitern benutzt oder bei der Anbindung von Filialen, bei denen eine Verbindung per Standleitung nicht in Frage kommt.

Weitere Anwendungen für replizierte Daten sind vielfältig, zum Beispiel kann ein Cluster aufgebaut werden, bei dem mehrere Datenbankserver zusammengeschaltet werden und die Last verteilt wird. Obwohl es auch in der Firebird und InterBase Welt verschiedene kommerzielle Anbieter für Replikationslösungen gibt, ist eine selbst implementierte Lösung vorteilhaft, weil man wesentlich genauer auf die individuellen Anforderungen eingehen kann und ganz nebenbei auch noch Lizenzkosten spart.

Grundlagen

Basis für eine replikationsfähige Datenbank sollte immer ein konsequent erstelltes Datenmodell sein. Die vom Autor bevorzugte Lösung basiert auf einem Primärschlüsselfeld ID vom Datentyp `BIGINT` in jeder Tabelle und konsequenter Benennung von Fremdschlüsselfeldern in der Form `TABELLE_ID`. Sämtliche Primärschlüssel werden immer aus einem gemeinsamen Generator erzeugt.

Das ist zwar auf den ersten Blick ungewöhnlich, bietet aber gerade für die Replikation und mögliche spätere Erweiterungen deutliche Vorteile. Wenn vorhandene Datenbanken nachträglich replikationsfähig gemacht werden sollen, bietet sich wahlweise die Ergänzung der vorhandenen Tabellen um ein Replikations-ID-Feld oder über Trigger gefüllte Paralleltabellen an.

Die vorgestellten Mechanismen basieren jedoch auf dem bevorzugten ID-Modell mit gemeinsamem Generator. Sämtliche SQL-Befehle sind auf dem frei erhältlichen Firebird Server angepasst. Für den Einsatz auf dem InterBase Server sind notwendige Änderungen teilweise explizit erwähnt.

Um eine Replikation aufbauen zu können, ist es zunächst mal wichtig, dass lückenlos sämtliche Datenänderungen in der Datenbank transaktionssicher protokolliert werden. Während andere, angeblich transaktionssichere Datenbanksysteme teilweise doch Lücken beim Rollback deutlich werden lassen, ist der verwendete Interbase und Firebird Server auch bei Trigger-Operationen immer transaktionssicher. Daher werden für die vorhandenen Tabellen entsprechende Trigger erstellt, die Insert-, Update- und Delete-Operationen auf jeder Tabelle protokollieren.
Geschrieben wird das Protokoll in folgende Tabelle:

```
CREATE TABLE IBE$LOG (
  ID  BIGINT NOT NULL PRIMARY KEY,
  USR VARCHAR(30) default current_user,
  TS  TIMESTAMP default current_timestamp,
  SQL VARCHAR(32000),
  IDX BIGINT,
```

```
  DAT BLOB SUB_TYPE 0 SEGMENT SIZE 16384
 );
```

Obwohl der Einsatz von sehr großen VARCHAR-Feldern nicht immer empfehlenswert ist, vereinfacht dieser das vorgestellte Modell. Für das ID-Feld kann mit IBExpert ein Autoinkrement-Trigger erzeugt werden, der seinen Wert aus einem Generator namens `ID` holen soll. `USR` und `TS` werden automatisch mit Benutzername und Zeitstempel gefüllt. Im Feld SQL wird der vollständige SQL-Quelltext gespeichert, der die identische Insert-, Update- oder Delete-Operation ausführen würde. Dieser wird später im Rahmen der Replikation zwischen den beteiligten Datenbanken ausgetauscht und auf dem replizierten System ausgeführt. Das Feld `IDX` ist als Hilfsfeld für den zugehörigen Primärschlüssel vorgesehen. Darüber kann später einfach ermittelt werden, welche Historie der Datensatz mit der ID 123 hat. Im Feld-`DAT` werden durch gesonderte Trigger geänderte Blobdaten für die Replikation gespeichert.

Damit vergebene Primärschlüssel auch global konfliktfrei sind, werden für alle beteiligten Server die ID-Generatoren auf unterschiedliche Startwerte gesetzt; Server A beginnt bei 1 Milliarde, Server B bei 2 Milliarden etc. Da ein Generator ein 64 Bit Integer Wert zurückgibt, könnten 16 Milliarden Replikationspartner jeweils 1 Milliarde global eindeutige IDs konfliktfrei erzeugen. Alternativ kann man bei Verringerung der Anzahl der Replikationspartner den Offset zwischen den IDs entsprechend erhöhen. Die oft gewählte Alternative auf Basis von GUIDs ist aus der Sicht des Autors eher nachteilig, weil auf Basis der IDs auch andere Lösungen möglich sind, zum Beispiel, dass Daten nur auf Servern geändert werden dürfen, auf denen sie auch entstanden sind.

Transaktionslog

Um für spätere Datenmodelländerungen gewappnet zu sein, ist es sinnvoll, das Erstellen der Trigger zu automatisieren. Aufgrund der zur Verfügung stehenden Befehle ist das für Firebird innerhalb einer Stored Procedure möglich. Der fehlende `EXECUTE STATEMENT` Befehl in InterBase macht es erforderlich, den Quelltext mit Hilfe von IBExperts IBEBlock Technologie auszuführen, weil auf diesem Wege auch der InterBase Server solche Sprachelemente beherrscht.

Die Prozedur `INITLOG` beginnt zunächst mit einer Schleife, bei der alle Tabellennamen ohne Dollarzeichen aus der Systemtabelle `RDB$RELATIONS` ausgelesen wird:

```
select f_rtrim(rdb$relation_name) from rdb$relations
where rdb$relation_name not containing '$'
 INTO :V$RELATION_NAME
```

In der folgenden Anweisung wird dann der Quellcode für den ersten After-Insert-Trigger der ersten gefundenen Tabelle begonnen:

```
sql='RECREATE TRIGGER IBE$'||V$RELATION_NAME||'_AI FOR
'||V$RELATION_NAME||' '||f_crlf()||
  'ACTIVE AFTER INSERT POSITION 32000 '||f_crlf()||
  'AS '||f_crlf()||
  'declare variable sql varchar(32000); '||f_crlf()||
  'begin '||f_crlf()||
  ' SQL=''INSERT INTO '||V$RELATION_NAME||'(';
```

Mit Hilfe der UDF `f_crlf` aus der Bibliothek FreeAdhocUDF[4] wird ein Zeilenvorschub im Triggerquelltext eingefügt, ohne den der Trigger zwar problemlos funktionieren würde, jedoch sehr unübersichtlich wäre.

In der folgenden Schleife werden zur aktuellen Tabelle alle Felder aus den Tabellen `RDB$RELATION_FIELDS` und `RDB$FIELDS` ausgelesen, deren Typ ungleich 261 ist. Der Typ 261 steht für Blobfelder und diese müssen später gesondert behandelt werden.

```
komma='';
 for select f_rtrim(rdb$relation_fields.rdb$field_name)
 from rdb$relation_fields
 join rdb$fields on
```

```
rdb$relation_fields.rdb$field_source=rdb$fields.rdb$field_name and
rdb$fields.rdb$field_type<>261
 where rdb$relation_name=:v$relation_name
 into :v$field_name
 do
 begin
 sql=sql||komma||v$field_name;
 komma=',';
 end
 sql=sql||') values (';
 komma='';
```

Durch die vorher leere und in der Schleife gesetzte Variable Komma entsteht eine kommagetrennte Liste aller Feldnamen, wie diese für einen Insert-Befehl erforderlich ist. Es folgt ein weiterer Durchlauf durch die Feldliste, bei dem die Instanzvariablen `NEW.` mit passenden Anführungszeichen für den zweiten Teil des Triggerquellcodes zusammengestellt werden. Dieser Teil ist aus Platzgründen nur im Beispielscript[5] zu finden.

Es folgt noch der Befehl, um den SQL Befehl aus dem Trigger heraus in die Tabelle `IBE$LOG` zu schreiben. Mit dem anschließenden Aufruf über den Befehl `EXECUTE STATEMENT :SQL` wird aus der Prozedur heraus der Triggerquelltext ausgeführt und damit der Trigger erzeugt.

```
 sql=sql||')'';'||f_crlf()||
   ' insert into ibe$log(sql,idx) values (:sql,new.id);'||f_crlf()||
   'end;';
 execute statement :sql;
```

Blobdaten

In den weiteren Teilen des Scripts wird in ähnlicher Weise der Update- und Delete-Trigger zusammengestellt und erzeugt. Am Ende werden dann noch extra Trigger für jedes Blob Feld erzeugt, weil hier auch nur dann Daten protokolliert werden sollen, wenn die auch wirklich geändert wurden. Für diesen Zweck werden aus den Systemtabellen die Feldnamen und Tabellennamen ausgelesen, deren Typ 261 ist.

```
FOR
 select

f_rtrim(rdb$relation_fields.rdb$relation_name),f_rtrim(rdb$relation_fields.
rdb$field_name)
 from rdb$relation_fields
 join rdb$fields on
rdb$relation_fields.rdb$field_source=rdb$fields.rdb$field_name
 where rdb$relation_fields.rdb$relation_name not containing '$'
 and rdb$fields.rdb$field_type=261
 INTO :V$RELATION_NAME, :V$FIELD_NAME
 DO
 BEGIN
 sql='RECREATE TRIGGER IBE$'||V$RELATION_NAME||V$FIELD_NAME||'_AI FOR
'||V$RELATION_NAME||'
 '||f_crlf()||
 'ACTIVE AFTER INSERT POSITION 32000 '||f_crlf()||
 'AS '||f_crlf()||
 'begin '||f_crlf()||
 ' if (new.'||V$FIELD_NAME||' is not null) then insert into
ibe$log(sql,idx,dat) values

(''''||V$RELATION_NAME||'.'||V$FIELD_NAME||''',new.id,new.'||V$FIELD_NAME||'
);'||f_crlf()||
 'end;';
 execute statement :sql;
```

```
 ....
```

Das Transaktionslog kann nun mit der Ausführung der Firebird Prozedur `INITLOG` oder des passenden IBEBlock Befehls für Interbase in der Datenbank aktiviert werden. Wenn Datenmodelländerungen anstehen, empfiehlt es sich, dieses Transaktionslog wieder zu deaktivieren, weil auf diesem Wege alle Referenzen auf die verwendeten Tabellen wieder gelöscht werden. Zu diesem Zweck ist im Beispielscript eine Prozedur `DROPLOG` implementiert.

Replizieren des Transaktionslogs

Die eigentliche Replikation, also der Austausch der Daten aus dem Transaktionslog in der richtigen Reihenfolge, beginnt nun mit einem ibeblock. Ein IBEBlock ist eine spezielle Erweiterung innerhalb der IBExpert Produktfamilie, die zusätzliche Befehle für die Scriptverarbeitung ermöglicht. Ein IBEBlock bietet auch für InterBase Befehle an, die innerhalb einer Prozedur nicht möglich sind, zum Beispiel der Befehl `EXECUTE STATEMENT`. Weiterhin ist es möglich, in einem IBEBlock Script eine Verbindung zu mehreren Datenbanken herzustellen. Über die integrierte ODBC-Schnittstelle kann die Replikation auch optional mit jeder ODBC-fähigen Datenbank durchgeführt werden. Solche IBEBlock Befehle können mit der DLL oder EXE Distributionslizenz auch unlimitiert in eigene Anwendungen eingebaut werden.

Zu Beginn werden im IBEBlock zunächst mal die Verbindungen zu den beteiligten Datenbanken erzeugt:

```
execute ibeblock
 as
 begin
 create connection src dbname 'localhost:c:\src.fdb'
 password 'repl' user 'REPL'
 clientlib 'fbclient.dll';

 create connection dest dbname 'localhost:c:\dest.fdb'
 password 'repl' user 'REPL'
 clientlib 'fbclient.dll';
```

Nachdem die Verbindungen erzeugt wurden, kann im Script mit dem Befehl `USE` beliebig zwischen den verschiedenen Datenbanken hin und her gesprungen werden. In der folgenden Schleife werden nun alle Einträge aus der Tabelle `IBE$LOG` in der Quelldatenbank ausgelesen und in die Tabelle `IBE$LOG` der Zieldatenbank eingefügt. Um bereits übertragene Daten nicht erneut zu replizieren, wurde in diesem Beispiel eine Tabelle `IBE$TRANS` referenziert, in die nach erfolgreicher Übertragung der Daten die ID aus der Tabelle `IBE$LOG` eingetragen wird. Der Benutzer `REPL` wurde für die Replikation benutzt, weil auf diesem Wege erkannt werden kann, welche Daten auf dem Replikationswege gekommen sind und aus diesem Grund nicht wieder zurück repliziert werden müssen.

```
 use src;
 for select id, usr, ts, sql, idx, dat
 from ibe$log where usr<>'REPL'
 and not exists (select ibe$trans.id from ibe$trans where
ibe$trans.id=ibe$log.id)
 into :id, :usr, :ts, :sql, :idx, :dat
 do
 begin
  use dest;
  insert into ibe$log(id, ts, sql, idx, dat)
  values (:id, :ts, :sql, :idx, :dat);
  if (sql not starting with 'BLOB ') then execute statement :sql;
  commit;
  use src;
  insert into ibe$trans(id) values (:id);
  commit;
```

```
end
```

Die Vorgehensweise bei der Replikation von Blobdaten entnehmen Sie bitte dem Beispielscript[5]. Ebenso kann dem Beispielscript die Vorgehensweise bei der bidirektionalen Replikation entnommen werden. Auf Basis dieser Technologie lässt sich mit wenig Aufwand ein System ergänzen, bei dem der Austausch der Daten für asynchrone Replikation durch komprimierte Blobdaten trotz geringer Bandbreiten auch für große Datenmengen ausreicht. Ebenso kann auf einem schnellen Backbone mit Hilfe der InterBase/Firebird Eventalerter Technologie ein sehr schneller und zuverlässiger Datenbankcluster aufgebaut werden.

Durch die anpassbaren Scripte lassen sich auch beliebige Regeln zur teilweisen Replikation implementieren. So können Datemengen auf verschiedene Server nach logischen Kriterien verteilt werden. So kann zum Beispiel der Kundenstamm auf alle Server verteilt werden, während die Auftragsdaten nur auf länderspezifische Datenbanken oder Server kopiert werden. Ebenso kann der umgekehrte Weg zur Kombination und Verdichtung von Daten mehrerer Datenbanken benutzt werden.

---

[1] www.firebirdsql.org
[2] www.borland.com
[3] www.ibexpert.com
[4] www.ibexpert.com/download/udf/FreeAdhocUDF.zip
[5] www.h-k.de/download/repl2006.zip