Bidirectional replication for InterBase and Firebird

The open source database server, Firebird[1], and its commercial partner, Borland InterBase[2], have long been established as a proven and stable platform for all sorts of database applications. Because of the common ancestry in the form of the InterBase 6 source code, many solutions can be implemented on both platforms without any problems. However Version 2 of the Firebird Server has recently set new standards, introducing many helpful functions that are unfortunately missing in InterBase 7.5. However a replication facility is not included in either platform. This article illustrates how a replication can be created with the aid of IBExpert[3].

What is replication?

The German-language Wikipedia offers a concise definition: "Replication is a duplication of data. The data base of the replicated data is, as a rule, identical with the original."

We need to distinguish between synchronous replication and asynchronous replication. Whilst the synchronous replication ensures that in the case of a fault or error, the database server can be immediately replaced by the replicated backup server and users can continue work without any disruption, an asynchronous replication makes sense when the databases and their servers are not always in the same network. An asynchronous replication is typically used for field staff and their laptops, or when branches of a company are not always connected to the main server by a dedicated line.

There are many further applications for replicated data, for example, a cluster can be constructed, by which multiple database servers can be interconnected to distribute the burden. Although there are various commercial suppliers offering replication solutions in the Firebird and InterBase world, a customized implementation has the advantage that it is possible to fulfil considerably more individual needs and wishes, at the same time saving license fees.

Fundamentals

The basis for a replicable database should always be a consequently constructed data model. The author's preferred solution is based on a primary key ID field, datatype `BIGINT` in every table and a consequent naming convention of foreign key fields in the form: `TABELLE_ID`. All primary keys are always created from a single generator.

This may initially appear somewhat unusual, but it does offer distinct advantages for replication and for any other subsequent extensions. Should existing databases need to be made replicable, existing tables can optionally be supplemented by a replication ID field or parallel tables filled using triggers.

The mechanisms presented here are based on the preferred ID model with a common generator. All SQL commands are accommodated on the freely available Firebird Server. Necessary alterations for deployment on the InterBase server are explicitly mentioned.

In order to construct a replication, it is initially vital that absolutely all data alterations in the database are logged securely. Whilst other, supposedly transaction-safe database systems clearly produced gaps when rolling back, the Firebird and InterBase server are always transaction safe even in the case of trigger operations. Therefore corresponding triggers are created for existing tables, which log all insert, update and delete operations on each table.

The log is written in the following table:

```
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
 );
```

Although it is not always recommendable to use very large VARCHAR fields, this simplifies the model presented here. An autoincrement trigger can be created using IBExpert for the ID field, the value of which should be fetched from a generator, called ID. USR and TS are automatically filled in with the user name and timestamp. The complete SQL source code is stored in the SQL field, which will execute the identical insert, update and delete operations. This will be later exchanged between the databases concerned as part of the replication, and executed on the replicated system. The IDX field is designed to be an auxiliary field for the associated primary key. This can later be used to easily ascertain the history of a data set with the ID 123. Altered blob data is stored by means of special triggers for the replication in the DAT field.

To avoid global conflict of allocated primary keys, all ID generators are set at different start values on all servers concerned; Server A starts at 1 billion, Server B at 2 billion etc. As generators return a 64 Bit value, 16 billion participating replication servers could each generate 1 billion globally unique IDs without any conflict. Alternatively the offset between the IDs on each server can of course be increased accordingly by reducing the number of replication servers involved. The author considers the popular alternative method based on GUIDs disadvantageous, because the ID method can also be used for other solutions, for example, that data may only be altered on the server where it was created.

Transaction Log

It is wise to automate trigger creation, so as to be armed for later data model alterations. Due to the commands available in Firebird, it is possible to do this within a stored procedure. The absence of the EXECUTE STATEMENT command in InterBase means that the source code needs to be executed using IBExpert's IBEBlock technology, as this method enables the InterBase server to handle such language elements.

The INITLOG procedure initially begins with a loop, extracting all table names from the system table, RDB$RELATIONS, which do not contain the dollar sign:

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

Then the source code for the first AFTER INSERT trigger for the first table found begins in the following statement:

```
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||'(';
```

Using the f_crlf UDF, from the FreeAdhocUDF library[4], a line feed is inserted into the trigger source code, without which the trigger would function, but nevertheless be extremely confusing.

In the following loop all fields in the current table are selected from the RDB$RELATION_FIELDS and RDB$FIELDS tables, whose type does not equal 261. Type 261 is for blob fields, which need to be treated separately later on.

```
 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='';
```

A comma-separated list of all field names is generated due to the previously empty variable and the comma variable defined in the loop, as required for an `INSERT` command. Then another sweep is made through the field list, in which the instance variable `NEW.` is prepared with the appropriate exclamation marks for the second part of the trigger source code. This part, due to lack of space here, can be found in the sample script[5].

This is now followed by the command to write the SQL command out of the trigger into the table `IBE$LOG`. With the subsequent request using the command, `EXECUTE STATEMENT :SQL`, the trigger source code is executed from the procedure, so creating the trigger.

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

Blob data

In the subsequent parts of the script, the update and delete triggers are constructed and generated in a similar way. Finally extra triggers are then created for each blob field, because only data should be logged which has actually been altered. For this purpose all field and table names with the type 261 are selected.

```
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;
  ....
```

The transaction log can now be activated in the database by executing the Firebird procedure `INITLOG` or in InterBase using the appropriate IBEBlock command. If data model alterations are to be made, it is wise to first deactivate this transaction log, as this way all references to the tables used will be deleted again. To this effect, the `DROPLOG` procedure is implemented in the sample script.

Replicating the transaction log

The actual replication, i.e. the data exchange from the transaction log in the correct order, now begins with an IBEBlock. An IBEBlock is a special extension within the IBExpert product family, which enables additional commands for the handling of scripts. An IBEBlock also offers commands for InterBase, which are not otherwise possible within a procedure, for example, the `EXECUTE STATEMENT` command. Furthermore it is possible to make a connection to multiple databases in an IBEBlock script. Replication can also optionally be carried out with all ODBC databases using the integrated ODBC port. Such IBEBlock commands may also be fully incorporated into your own applications using the DLL or EXE distribution licenses.

IBEBlock first makes the connections to the databases involved:

```
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';
```

After the connections have been made it is possible to switch backwards and forwards between any of the databases, using the `USE` command. The following loop now selects all entries in the `IBE$LOG` table in the source or reference database and inserts them into the `IBE$LOG` table in the target database. In order to avoid re-replicating data that has already been transferred, a table, in this example `IBE$TRANS,` is referenced, in which the ID from `IBE$LOG` is entered following successful data transmission. The user `REPL` was used for the replication, because this way it is possible to recognize which data have come via the replication and therefore do not need to be replicated back again.

```
  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
```

The approach to be taken when replicating blob data can be found in the sample script[5]. This also demonstrates the procedure for bidirectional replication. Using this technology little effort is needed to supplement a system, which is capable of exchanging data for asynchronous replication using packed blob data and is sufficient for large data quantities, even when low band widths are used. It is also possible on a quick backbone to construct an extremely rapid and reliable database cluster using the InterBase/Firebird Event Alerter technology.

The customizable scripts can be implemented for partial replication, by using any number and combination of rules. In this way it is possible to distribute data quantities to various servers according to logical criteria. For example, the customer base can be distributed to all servers, whilst the order data is only copied to country-specific databases or servers. Or the inverse direction can be used to combine and consolidate data from multiple databases.

---

[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