Firebird White Paper

Firebird 4 Replication

Fikret Hasovic, July 2021

What is database replication?

Database replication is the frequent copying of data from a database from one server to a database in another, so that all users share the same level of information. The result is a distributed database in which users can quickly access data relevant to their tasks.

A quick introduction

Firebird 4 introduces built-in support for uni-directional ("primary-replica") logical replication. Logical here means record-level replication, as opposed to physical (page-level) replication.

Events that are tracked for replication include inserted/updated/deleted records, sequence changes and DDL statements.

Replication is transactional and the commit order is preserved. Any table that is to be replicated must have a primary key or, at least, a unique key.

Both *synchronous* and *asynchronous* modes are available.

Synchronous Mode

In synchronous replication, the primary (master) database is permanently connected to the replica (slave) database(s) and changes are replicated immediately. Effectively the databases are in sync after every commit, which could have an impact on performance due to additional network traffic and round-trips.

You can have more than one synchronous replica, if necessary.

Asynchronous Mode

In asynchronous replication, changes are written into local journal files that are transferred over the wire and applied to the replica database. The impact on performance is much lower, but imposes a delay — so called *replication lag* — while changes wait to be applied to the replica database; i.e. the replica database is always "catching up" the master database.

Access Modes

There can be two access modes for replica databases: *read_only* and *read_write*.

With a read_only replica, only queries that do not modify data are allowed. Modifications are limited to the replication process only.

A read_write replica allows execution of any query, so potential conflicts can exist and they must be resolved by users or database administrators.

For a detailed explanation and full documentation, please consult the Release Notes.

Let's do some exercises now…

Setting up the replication

The setup involves tasks on both the primary and replica sides.

Setting up the Primary Side

Replication is configured using a single configuration file, `replication.conf`, on the host serving the primary database. Both global and per-database settings are possible within the same file. The available options are listed inside `replication.conf`, along with commented descriptions of each.

Inside the database, replication should be enabled using the following DDL statement:

```
ALTER DATABASE ENABLE PUBLICATION
```

Defining a Custom Replication Set

Optionally, the replication set (aka publication) should be defined. It includes tables that should be replicated. This is done using the following DDL statements:

```
-- to replicate all tables (including the ones created later)
ALTER DATABASE INCLUDE ALL TO PUBLICATION

-- to replicate specific tables
ALTER DATABASE INCLUDE TABLE T1, T2, T3 TO PUBLICATION
```

Tables may later be excluded from the replication set:

```
-- to disable replication of all tables (including the ones created later)
ALTER DATABASE EXCLUDE ALL FROM PUBLICATION

-- to disable replication of specific tables
ALTER DATABASE EXCLUDE TABLE T1, T2, T3 FROM PUBLICATION
```

Tables enabled for replication inside the database can be additionally filtered using two settings in `replication.conf: include_filter and exclude_filter`.

Synchronous/Asynchronous Modes

Synchronous Mode

Synchronous replication can be turned on by setting the `sync_replica` specifying a connection string to the *replica* database, prefixed with username and password. Multiple entries are allowed so you can define many replicas.

Asynchronous Mode

For asynchronous replication the journaling mechanism must be set up. The primary parameter is `journal_directory` which defines the location of the replication journal. Specifying this location turns on asynchronous replication and tells the Firebird server to start producing the journal segments.

A Minimal Configuration

A minimal primary-side configuration would look like this:

```
database = c:\db\MASTER.FDB
{
    journal_directory = c:\db\journal_directory\
    journal_archive_directory = c:\db\journal_archive_directory\
}
```

Archiving is performed by the Firebird server copying the segments from `c:\db\journal_directory\` to `c:\db\journal_archive_directory\.`

You can, however, create the setup with user-defined archiving. Custom archiving, through use of the setting `journal_archive_command` allows use of any system shell command, including scripts or batch files, to deliver segments to the replica side. It could use compression, FTP, or whatever else is available on the server.

The same setup as above, with archiving performed every 10 seconds:

```
database = c:\db\MASTER.FDB
{
    journal_directory = c:\db\journal_directory\
    journal_archive_directory = c:\db\journal_archive_directory\
    journal_archive_timeout = 10
}
```

You can check `replication.conf` (in the Firebird root directory) for many other possible settings.

Applying the Primary Side Settings

To take into effect changes applied to the primary-side settings, all users connected to a database must be disconnected (or a database must be shutdown). After that, when all users connect again, it will use the updated configuration.

Setting up the Replica Side

The `replication.conf` file is also used for setting up the replica side. Setting the parameter `journal_source_directory` specifies the location that the Firebird server scans for the transmitted segments. In addition, the DBA may specify explicitly which source database is accepted for replication, by setting the parameter `source_guid`.

A Sample Replica Setup

A configuration for a replica could looks like this:

```
database = c:\db\REPLICA.FDB
{
    journal_source_directory = c:\db\journal_archive_directory\
    source_guid = {6F9619FF-8B86-D011-B42D-00CF4FC964FF}
}
```

Read through the `replication.conf` for other possible settings.

Applying the Replica Side Settings

To take into effect changes applied to replica-side settings, the Firebird server must be restarted.

Creating a Replica Database

Task 1 — Make the initial replica

Any physical copying method can be used to create an initial replica of the primary database, but the simplest way is file-level copy while the Firebird server is shut down.

Task 2 — Activate the replica access mode

Activating the access mode - for the copied database - involves the command-line utility *gfix* with the new `-replica` switch and either `read_only` or `read_write` as the argument:

- **To set the database copy as a read-only replica**

  Start *gfix* from the Firebird 4 root directory in the command line:

  ```
  gfix -replica read_only <database>
  ```

  If the replica is read-only then only the replicator connection can modify the database. This is mostly intended for high-availability solutions, as the replica database is guaranteed to match the primary one and can be used for fast recovery. Regular user connections may perform any operations allowed for read-

only transactions: select from tables, execute read-only procedures, write into global temporary tables, etc. Database maintenance such as sweeping, shutdown, monitoring is also allowed.

A read-only replica can be useful for distributing read-only load, for example, analytics, away from the master database. Read-only connections have the potential to conflict with replication if DDL statements that are performed on the master database are of the kind that requires an exclusive lock on metadata.

- **To set the database copy as a read-write replica**

  Again using *gfix* in the command line:

  ```
  gfix -replica read_write <database>
  ```

  Read-write replicas allow both the replicator connection and regular user connections to modify the database concurrently. With this mode, there is no guarantee that the replica database will be in sync with the master one. Therefore, use of a read-write replica for high availability conditions is not recommended unless user connections on the replica side are limited to modifying only tables that are excluded from replication.

Task 3 — Converting the replica to a regular database

A third `gfix -replica` argument is available for "switching off" replication to a read-write replica when conditions call for replication flow to be discontinued for some reason. Typically, it would be used to promote the replica to become the primary database after a failure; or to make physical backup copies from the replica.

```
gfix -replica none <database>
```

Real-world example and exercise

Let's manually download the 64-bit version and run it as an application (you might already have Firebird 2.5 and/or Firebird 3 running as a service locally), but you can use your preferred method of installation.

We extract our Firebird-4.0.0.2496-1-x64.zip archive to some custom location, for example, `C:\Firebird\Firebird-4.0.0.2496-1-x64\`. As mentioned above, let's run it as an application by executing `firebird.exe -a` in the command line from the directory above. Make sure to change the default Firebird 4 port to some custom value if you already have older (or the same) Firebird instances running by adding the following to the `firebird.conf`:

```
RemoteServicePort = 3054
IpcName = FIREBIRD4
```

Also, since we are installing from a zip file, we need to manually initialize the *Security Database*:

1. **Stop the Firebird server**. Firebird 4 caches connections to the security database aggressively. The presence of server connections may prevent *isql* from establishing an embedded connection.
2. In a suitable shell, start an *isql* interactive session using the command-line `isql.exe` from the Firebird root directory, opening the *employee* database via its alias:

```
isql -user sysdba employee
```

3. Create the SYSDBA user:

   You can do this on the command line or in IBExpert: *Tools / SQL Editor*:

```
SQL> create user SYSDBA password 'masterkey';
SQL> commit;
SQL> quit;
```

4. To complete the initialization, start the Firebird server again. Now you will be able to perform a network login to databases, including the security database, using the password you assigned to SYSDBA.

Common for all scenarios:

Before we start the replication process, we need to create physical copy of the master database, for example by simple copy/paste whilst the server is not running, so copy `MASTER.FDB` and paste it with new name i.e. `REPLICA.FDB` or any other name you choose.

Next step is to run

```
gfix -replica read_only c:\db\REPLICA.FDB -user sysdba -pass masterkey
```

Connect to the master database and execute the following DDL:

```
ALTER DATABASE INCLUDE ALL TO PUBLICATION;
ALTER DATABASE ENABLE PUBLICATION;
```

Scenario 1:

Let's try replication testing on this single Firebird 4 instance, and let's use the *Synchronous* mode:

Now we need to modify `replication.conf` file (located in the root Firebird folder, together with the `firebird.conf` file) by adding the following:

```
database = c:\db\MASTER.FDB
{
    sync_replica = SYSDBA:masterkey@localhost/3054:c:\db\REPLICA_SYNC.FDB
}
```

Since we have a single Firebird 4 instance, that is all that is needed.

Now execute the `common` procedure with the name `REPLICA_SYNC.FDB`.

It's time to run firebird now, as we explained above:

```
firebird.exe -a
```

Now your database should start replicating. You can try adding or deleting some data, and all operations should be duplicated immediately in your replica database.

Scenario 2:

Let's use the *Asynchronous* mode:

This time we need to modify `replication.conf` file by adding following:

```
database = c:\db\MASTER.FDB
{
 journal_directory = c:\db\journal_directory\
 journal_archive_directory = c:\db\journal_archive_directory\
 journal_archive_timeout = 10
 verbose_logging = true
}
database = c:\db\REPLICA_ASYNC.FDB
{
 journal_source_directory = c:\db\journal_archive_directory\
 verbose_logging = true
}
```

Note that we have two database entries this time, and reason is that this single Firebird 4 instance is responsible for dealing with journal files.

Now execute the Common procedure with the name `REPLICA_SYNC.FDB`.

In this case you can notice that logging is enabled, so you can easily check the status of replication process in `replication.log` located in your Firebird 4 install directory.

Scenario 3:

You can, if you wish, enable both *sync* and *async* replication modes by combining the replication config:

```
database = c:\db\MASTER.FDB
{
 sync_replica = SYSDBA:masterkey@localhost/3054:c:\db\REPLICA_SYNC.FDB
 journal_directory = c:\db\journal_directory\
 journal_archive_directory = c:\db\journal_archive_directory\
 journal_archive_timeout = 10
 verbose_logging = true
}
database = c:\db\REPLICA_ASYNC.FDB
{
 journal_source_directory = c:\db\journal_archive_directory\
 verbose_logging = true
}
```

Don't forget to execute the Common procedure with the name `REPLICA_SYNC.FDB`.

Scenario 4:

Let's introduce one more Firebird 4 instance, for this exercise on the `localhost`.

We will replicate the firebird install, set it up to listen on different port i.e.

```
RemoteServicePort = 3055
IpcName = FIREBIRD4_2
```

and you can define *async* replication setup as follows:

1. On the 3054 instance, modify the `replication.conf` file:

```
database = c:\db\MASTER.FDB
{
 journal_directory = c:\db\journal_directory\
 journal_archive_directory = c:\db\journal_archive_directory\
 journal_archive_timeout = 10
 verbose_logging = true
}
```

2. On the 3055 instance, modify the `replication.conf` file:

```
database = c:\db\REPLICA_ASYNC.FDB
{
 journal_source_directory = c:\db\journal_archive_directory\
 verbose_logging = true
}
```

As usual, don't forget to execute the Common procedure with the name `REPLICA_ASYNC.FDB`.

Now, after restarting both instances, you can have one instance initiating the replication process and creating journal files, and the second instance processing journal files and applying them to the replica database.

Note that, if you are using two different servers, `journal_archive_directory` and `journal_source_directory` (respectively per instance) must point to some network drive accessible by both instances, and be prepared to expect some delay in processing the journal files.

Scenario 5:

Sync replication on the secondary instance, and define the async replication setup as follows:

```
database = c:\db\MASTER.FDB
{
    sync_replica = SYSDBA:masterkey@localhost/3055:c:\db\REPLICA_SYNC.FDB
}
```

On the secondary instance there is nothing to configure.

As usual, don't forget to execute the Common procedure with the name `REPLICA_ASYNC.FDB`.

Scenario 6:

There is a possibility to enable multiple synchronous replicas, by listing them in `replication.conf`:

```
sync_replica = SYSDBA:pwd1@myserver1:c:\db1\REPLICA_SYNC.FDB
sync_replica = SYSDBA:pwd2@myserver2:c:\db2\REPLICA_SYNC.FDB
sync_replica = SYSDBA:pwd3@myserver3:c:\db3\REPLICA_SYNC.FDB
```

As usual, don't forget to execute the Common procedure with the name `REPLICA_SYNC.FDB`.

Using the replica as the main database

If you want to promote the replica to become the primary database (for example following a failure), you should use

```
gfix -replica none <database> -user sysdba -pass masterkey
```

As already mentioned above, the read-only replica can be used as a hot stand-by server and for distributing read-only load away from the master database.

However, as mentioned above, read-write replicas allow both the replicator connection and regular user connections to modify the database concurrently, so there is no guarantee that the replica database will be in sync with the master one.

So, bi-directional replication is not supported natively, unfortunately.