How to analyse and repair a corrupted database

You have to try very hard to corrupt a Firebird database — it's designed to survive the hard knocks that break databases in other systems. You will know a database has been corrupted if you cannot connect to it or back it up and a message in the firebird log, or from gbak –b[ackup], tells you that there is corruption, or some other message reports a checksum error.

This topic describes the steps you need to take, primarily using the command-line tools gfix and gbak, to try to identify some kinds of corruption and repair the damage. Be aware, however, that there are certain kinds of severe corruption that this procedure cannot fix.

Recognising possible corruption

Evidence of possible corruption, although rare enough, might appear during the database validation that you do as part of your routine housekeeping to detect minor anomalies and recycle misallocated space. You might have run a validation to check whether some misbehaviour could be associated with structural damage, perhaps prompted by one of the following circumstances:

- A corrupt database or consistency check error appearing in the firebird.log or in a client application
- A backup that ends abnormally
- Power failure or brownout without UPS protection or with suspected UPS failure
- Suspected or system-reported hard disk, network, or memory faults
- A database shadow taking over from a dead database after a disk crash
- A production database is about to be moved to another platform or storage system
- Suspected compromise of the network or database by a malicious attack

Preparing for analysis and repair

If you suspect you have a corrupt database, it is important to follow a proper sequence of recovery steps in order to avoid further corruption. Because some of the steps make irreversible changes to structures within the database, it is essential to isolate the original database file from these operations and work with a file copy.

Take the database off-line as soon as evidence of a possible corruption appears and isolate the file to prevent any further connections. If it is not practicable to move the file then rename it.

Important: Never attempt a recovery task by operating directly on the production database file.

1. Get the database off-line.

- Skip this step if the server is not running or is unable to accept connections.
- If the server is running and is still accepting connections then ask all users to cancel their work and log out. If necessary, use gfix -force with a sufficient timeout to let them try to get off gracefully.
- If logged-in users are unable to log out, the last resort will be to stop the server (Superserver or

Superclassic) or kill the processes (Classic).

CAUTION: If the Superserver or Superclassic server is serving other databases, be sure to get those databases off-line before you stop the server.

2. Make a working copy of the database. Once the database is off-line, take a file copy of it using a reliable file system utility and store it to disk in the location where you intend to work with it. Take note:

- Do not locate the copy in the same directory as the suspect database.
- If you suspect hard disk damage, make at least two copies and place them on a separate physical drive. Work on one and keep the other as a fall-back.
- Make sure enough space is available to accommodate the copied file.
- Do not start the analysis and repair steps until you are certain the copying operation has finished.
- Make sure you have sufficient disk space available to both create a backup and restore the backup to a new version of the databases.

3. Make an alias for the working datababase in aliases.conf. For example,

```
tempdb.fdb = /var/databases/copy1.fdb
```

4. Set the environment variables ISC_USER and ISC_PASSWORD.

```
 This step is optional but, because the procedure involves the command-line
utilities gfix and gbak, requiring repeated logins as SYSDBA, it will save
some typing.
 How they are set depends on platform. For example, on Windows you can can
use the SET command in the command shell or the specialised applet under
Control Panel -> Adminstration Tools -> Advanced.
 The arguments are:

 ISC_USER=SYSDBA
 ISC_PASSWORD=masterkey

 but substituting the correct SYSDBA password, of course.
```

```
 CAUTION: Having these variables set can create a security vulnerability. If
you have other databases that need to be kept running while you are
repairing this one, either:
 Take the long road: skip this step and be prepared to do the typing; or
 Use Superuser or Administrator privileges to masquerade as SYSDBA. This is
possible on Linux and configurable on Windows if the server is Firebird 2.1
or higher.
 Now you should be ready to begin the analysis and repair procedure.
```

Now you should be ready to begin the analysis and repair procedure.

Steps for recovery using command-line tools

For our purposes here, let's assume that

- you are set up so that you don't have to supply login credentials
- you have aliased your working copy (in aliases.conf) as copy1.fdb

The steps described below involve use of privileged options of the command-line tools gfix and gbak. They will not work if you do not have SYSDBA or equivalent privileges. This procedure can mend some forms of corruption and return the database to a state wherein it becomes usable. It usually results in some data loss, since its strategy is to disable and remove troublesome records and even pages, in some situations.

---

1. Check for database corruption. The gfix switches –v[alidate] and –f[ull] are used first, to check record and page structures. This checking process reports corrupt structures and releases unassigned record fragments or orphan pages (i.e., pages that are allocated but unassigned to any data structures).

```
fix -v -full copy1.fdb
```

The switch –n[o update] can be used with –v, to validate and report on corrupt or misallocated structures without attempting to fix them:

```
fix -v -n copy1.fdb
```

If persistent checksum errors are getting in the way of validation, include the –i[gnore] switch to have the validation ignore them:

```
fix -v -n -i copy1.fdb
```

- If errors were reported, the next step is for you.
- If the outcome of all this is that no errors were reported, skip the next two steps: attempting to mend databases without broken structures can actually create breakages.

2. Mend corrupted pages

If gfix validation reports damage to data, the next step is to mend (or repair) the database by putting those structures out of the way. The –m[end] switch marks corrupt records as unavailable, so they will be skipped during a subsequent backup. Include a –f[ull] switch to request mend for all corrupt structures and an –i[gnore] switch to bypass checksum errors during the mend.

```
fix -mend -full -ignore copy1.fdb
```

or, briefly

```
fix -m -f -i copy1.fdb
```

3. Validate after –mend

After the –mend command completes its work, again do a full validation to check whether any corrupt structures remain:

```
gfix -v -full copy1.fdb
```

4. Clean and recover the database

Whether you skipped steps 2 and 3 or not, now do a full backup and restore using gbak, even if errors are still being reported. Include the –v[erify] switch to watch progress. In its simplest form, the backup command would be (all in one command):

```
bak -b -v -i copy1.fdb [pathto]copy1.fbk
```

The following should help to resolve some of the complications that might show up during gbak:

a. Garbage collection problems might cause gbak to fail. If this happens, run it again, adding the –g switch to signify no garbage collection:

```
gbak -b -v -i -g copy1.fdb{path}copy1a.fbk
```

b. Corruption in limbo transactions. If there is corruption in record versions associated with a limbo transaction, you may need to include the –l[imbo] switch:

```
gbak -b -v -i -g -l {path}copy1a.fbk
```

5. Restore the cleaned backup as a new database.

Now create a new database from the backup, using the –v[erify] switch to watch what is being restored:

```
gbak -create -v [pathto]copy1a.fbk [pathto]reborn.fdb
```

If the restore throws up further errors, you may need to consider further attempts using gbak -c with other switches to eliminate the sources of these problems. For example:

a. Restore with inactive indexes.

The –i[nactive] switch will eliminate problems with damaged indexes, by restoring without activating any indexes. Afterwards, you can activate the indexes manually, one at a time, until the problem index is found.

b. Restore tables one at a time.

The –o[ne_at_a_time] switch will restore and commit each table, one by one, allowing you restore good tables and bypass the problem ones.

6. Validate the restored database.

Verify that restoring the database fixed the problems by validating the restored database with the –n[o_update] switch:

```
gfix -v -full [pathto]reborn.fdb
```

If identifiable damage to page and record structures were found and fully eliminated by this procedure, you should now be able to log in to the “reborn” database.

All that remains now is to store the good backup in a safe place, move the repaired copy back to its proper directory and rename it.

If the preceding steps do not work, but you are still able to access the copy of the corrupt database, you may still be able to transfer table structures and data from the damaged database to a new one, a technique known as data pumping. Several free and commercial tools are available. Data pumping on pre-Firebird 2 servers can also be done using the QLI (Query Language Interpreter) command line tool that can be found in Firebird's /bin directory.