Managing a firebird server: Daily backups

Michaël Van Canneyt

August 29, 2011

Abstract

Firebird is an open source database which is less known than its popular cousin MySQL. It deserves attention, because it has been around for much longer than MySQL and has an extremely rich feature list and is not encumbered by a murky license scheme. Like all databases, firebird databases must be backed up regularly. In this article we take a look at how to organize backups for a firebird server.

1 Introduction

Firebird is more than 25 years old. In these 25 years it has evolved to a very robust, stable database which van run 24/7 virtually unattended on almost all modern platforms. It can be downloaded from

```
http://www.firebirdsql.org/
```

Install scripts, packages or programs are available for all supported platforms.

As a database, it has a small memory footprint and installation footprint, yet it has all features of a database such as Oracle. (in fact, it can successfully emulate oracle). For a system administrator, it would be very easy to forget that Firebird is installed since it is so unobtrusive.

However, every system administrator knows that accidents happen, and therefor, all files on a system - and in particular database files - must be backed up.

For firebird, as for many other databases, it is not a good idea just to copy the file(s) that make up a database to a backup system. If the files are in use, chances are that the copy will be corrupt, if the copy was made during a write operation of the database engine.

To make backups, firebird offers 2 backup tools: gbak, and nbackup. The former is used to create complete backups of a database, the latter is used to create incremental backups. In this article the former tool is used.

2 Creating a backup

The gbak tool is simple to use:

```
gbak -USER SYSDBA -PASSWORD Secret /path/to/db /path/to/backup
```

it requires the database filename and credentials, and a file in which to write the backup. Nothing could be more easy. There are some options to backup only metadata, but these are of no use when creating backups for safe keeping. The above command will back up the database in it's current state, that is, the state it was in when the backup was started. People can continue working while the backup is in progress, the backup does not interfere with the normal working of the database. Changes applied to the database after the backup was started, are of course not included in the backup.

If only one database exists on the system, then it would be sufficient to insert the above line in the crontab of a user, and make sure the backup is written somewhere where the regular backup tool finds it.

If disaster strikes, one need but restore the system, copy the backup file back on the system, and restore the database with the following command:

```
gbak -CRE -USER SYSDBA -PASSWORD Secret /path/to/backup /path/to/db
```

The username should be the actual owner of the database, otherwise the owner won't be able to write in the database.

3 Caveat: corrupted backups

The above does not tell the full story. A firebird backup can be corrupt. That is, the backup tool will create a backup file, but this backup file cannot be restored into a database. There can be 2 reasons for this:

1. The backup was corrupted by the copying process.
2. The backup was created corrupt in the first place.

While the former cannot be helped much, the latter can be checked for and prevented.

How can a backup be created corrupt ? It seems strange that a backup created by Firebird's own backup tool can be corrupt. Nevertheless, it is very easy to create a backup that is corrupt. Consider the following sequence of SQL commands:

```
CREATE TABLE TESTRESTORE (
  ANAME VARCHAR(50) NOT NULL
);
COMMIT;
INSERT INTO TESTRESTORE VALUES ('Michael Van Canneyt');
COMMIT;
ALTER TABLE TESTRESTORE ADD MYKEY INT NOT NULL;
COMMIT;
```

This does 3 things:

1. Create a table with a required field (`AName`).
2. Insert a record in the table with a value for the required the field
3. Add a required field (`MyKey`) to the table.

At this point, the records that are present in the table have no value for the required field. If the database is backed up, there is no value to back up, and none is written to the backup file. When the database is restored, first the table is restored in its current state (2 required fields: `AName` and `MyKey`), and then the values are inserted. Since the row in the backup contains no value for the field `MyKey`, the engine complains that a value is missing for `MyKey` and the restore fails.

The solution is simple: when the required field is added to the table, existing rows should be updated so they have a value for the `MyKey` field:

```
UPDATE TESTRESTORE SET MYKEY=0;
COMMIT;
```

For a table that contains many rows, this operation can be time consuming. This is the reason the firebird designers do not automatically do this when adding a required field (one can argue that it is pointless not to do this, since all backups will be invalid if it is not done).

There may be other ways to end up with a corrupt backup from a working database, but the above one is the most obvious.

The 2 reasons mentioned here make it clear that it is important to check whether the backups of firebird databases are valid. (incidentally, MySQL's backup tool chooses and inserts a value for `MyKey` all by itself when creating the backup).

4 A backup script

For a database server with many database files, it is handy to have a script that automates the backups, and tests the created backups. This script can then be inserted in the crontab. The script has the following features:

- Creates backups of a list of databases.
- Only creates backups if the database has changed since the last backup.
- Removes 'old' backups.
- Keeps a log of all operations.
- Can be configured to test the backups.
- Can be configured to compress the backups.
- Can be configured to run an extra command on all backups. For instance, to copy them to a NAS).
- Can be configured to send a separate mail with a list of failures.

The last item is useful in that it means that the sysadmin does not have to check the full (daily) log: he will be notified if anything goes wrong, so he can take the necessary steps to correct the reasons for failure.

The restore testing is put in a separate script, so it can be run separately. The script has the following features:

- It can uncompress a compressed backup in a temporary file.
- It tests the backup process of a list of databases (it's main function)
- It can compress the backup if it was not compressed to begin with.
- It can be configured to send a mail with a list of failures.

The first script takes no arguments, and is configured through some variables at the start of the script file:

```
BaseDir=/home/firebird
DatabaseListFile=/home/backup/databases.lst
BackupDir=/home/backup/backups
LogDir=/home/backup/logs
DBUser=SYSDBA
DBPassword=Secret
Old=20
UseDirs=y
Owner=www-data:www-data
TestValidity=Y
FailMail=michael@freepascal.org
```

The meaning of these variables is pretty straightforward:

BaseDir Directory where all Firebird database files are.

DatabaseListFile list with database filenames to back up. Names are relative to the `BaseDir` directory.

BackupDir Directory where backups are written. This directory can be subsequently backed up by the file backup tool.

LogDir Where to write a log of operations.

DBUser The Firebird user to use when creating backups.

DBPassword The password of the firebird user.

Old Number of days after which a backup file is deleted.

UseDirs if set to `y` then a separate directory will be created for the backups of each database. Any other value will create all backups in the same directory.

Owner If this is set, the resulting files will be chown-ed to this user (this works only if the script is run as root).

TestValidity if set to `Y` then the backups will be restored in a temporary database to see if they are valid.

FailMail if non-empty, a mail will be sent with a list of failures. No mail is sent if there are no failures.

The script relies on some standard unix commands to be in the PATH: awk, rm, chown, cat, mail, du, mktemp, gzip and bzip2. The location of the firebird GBAK and ISQL tools can be configured.

The script itself (called `backupfb`) is rather straightforward. After the declarations of the above variables, it starts with setting some variables which it needs:

```
TheDate=`date +%Y-%m-%d`
LogFile="$LogDir/$TheDate.log"
Errors="N"
FailList=`mktemp`
RestoreList=`mktemp`
```

The faillist and restorelist are temporary files which will contain the names of the failed and correct database backups. Then, the script defines a function to compress a backup. It will also set the name of the owner to the value specified in the `Owner` variable.

```
compressbackup () {
  retval=0
  dbbackfile=$1
  StartTime=`date +%H:%M:%S`
  echo "$StartTime Compressing backup file $dbbackfile"
  oldsize=`du -h  $dbbackfile | awk '{print $1}'`
  bzip2 -f $dbbackfile
  if [ $? != 0 ]; then
    echo "ERROR: compressing of backup file $dbbackfile failed !"
    retval=1
  else
    newsize=`du -h  $dbbackfile.bz2 | awk '{print $1}'`
    EndTime=`date +%H:%M:%S`
    echo "$EndTime Compressed backup from $oldsize to $newsize."
    if [ "$Owner" != "" ]; then
       echo "Setting backup file ownership to $Owner"
       chown $Owner $dbbackfile.bz2
       retval=2
    fi
  fi
}
```

Stripped of the echo commands, this is actually a simple function. The output of everything is sent to a log file, hence the rather verbose output. The function returns 0 if successfull, or nonzero if an error occurred.

Next, it defines a function that does the actual backup. The first part determines the filename of the backup:

```
backupdatabase () {
  retval=0
  db=$1
  dbback=`echo $db | tr / _`
  dbback=`basename $dbback .gdb`
  dbback=`basename $dbback .fdb`
  dbback=`basename $dbback .fb`
  dbback="$BackupDir/$dbback"
  if [ "$UseDirs" = "Y" ]; then
    dbbackfile="$dbback/$TheDate.fbk"
    lastfile="$dbback/last"
    if [ ! -d $dbbackfile ]; then
      echo "Creating backup directory $dbback"
      mkdir -p $dbback
      if [ $? != 0 ]; then
        echo "Error: failed to create backup directory $dbback"
      fi
    fi
  else
    dbbackfile="$dbback-$TheDate.fbk"
    lastfile="$dbback.last"
  fi
```

As can be seen, it also creates a 'lastfile'. This file will be created if the backup succeeds.

In the next part the script checks whether a backup must actually be made, by comparing the timestamp of the database with the timestamp of the 'last' file:

```
  needbackup=no
  echo Examining $db
  if [ -e $lastfile ]; then
    if [ $lastfile -ot $BaseDir/$db ]; then
      echo Backup older than database.
      needbackup=yes
    else
      echo Backup newer than Database. Skipping.
    fi
  else
    echo First time backup.
    needbackup=yes
  fi
```

Finally, the backup process is started:

```
  if [ $needbackup = yes ]; then
    if [ -f $dbbackfile ]; then
      echo $dbbackfile exists already. Removing old copy.
      rm -f $dbbackfile
    fi
    StartTime=`date +%H:%M:%S`
    echo "$StartTime : Backing up to $dbbackfile..."
    $GBAK -B -USER $DBUser -PASSWORD $DBPassword $BaseDir/$db $dbbackfile
    if [ $? != 0 ]; then
      retval=1
      EndTime=`date +%H:%M:%S `
      echo "ERROR: $EndTime : backup failed !"
      echo $db >> $FailList    else
      EndTime=`date +%H:%M:%S `
      echo "$EndTime : backup completed."
      touch $lastfile
      if [ $TestValidity = 'Y' ]; then
        echo $dbbackfile >>$RestoreList
      else
        # Compress backup
        compressbackup $dbbackfile
      fi
    fi
  fi
  return $retval
}
```

Note that the backup is only compressed by this function if the validity must not be checked. If the validity of the backup is checked, then the validation script will compress the backup.

The above function is now called by the main part of the script, which is executed in a sub shell (to catch the output) and starts out by removing old databases

```
(
echo Using list file $DatabaseListFile.
echo Using Backup Directory $BackupDir.
echo Checking old backups
for olddb in `find $BackupDir -mtime +$Old -name '*.bz2'`
do
```

```
  echo Removing old database $olddb:
  ls -l $olddb
  rm -f $olddb
done
```

Next, the actual backups are made:

```
echo Backing up databases from directory $BaseDir
for db in `cat $DatabaseListFile`
do
  echo Backing up $db
  backupdatabase $db
  if [ "$?" = "1" ]; then
    Errors=Y
  fi
done
```

Note that if an error is detected, the 'Errors' variable is set. This is used to send the notification mail:

```
if [ $Errors = "Y" ]; then
  echo "Errors detected"
  if [ ! -z "$FailMail" ]; then
    ErrCount=`wc -l $FailList`
    (echo "The following backups failed:" ;
     cat $FailList
     echo "Check the log file $LogFile for details:"
     ) | mail -s "Attention : $ErrCount failed firebird backups" $FailMail
  fi
fi
```

Finally, if validity testing of the backups is requested, the check script is called, passing it the filename with the list of succeeded backups and the email address to mail to in case of an error:

```
if [ $TestValidity = "Y" ]; then
  echo "Testing Restore of generated backups"
  testfbrestore -c "$RestoreList" "$FailMail"
fi
```

The last thing to do is some clean-up:

```
echo "Removing error list";
rm "$FailList"
if [ $? != 0 ]; then
  echo "Failed to remove list of errors $FailList"
fi
echo "Removing restore list"
rm "$RestoreList"
if [ $? != 0 ]; then
  echo "Failed to remove list of databases to restore $RestoreList"
fi
) | tee $LogFile
```

The last line saves the output of the subshell to the logfile, which is then set to the owner as configured in the beginning of the script:

```
if [ "$Owner" != "" ]; then
  echo "Setting log file ownership to $Owner"
  chown $Owner $LogFile
fi
```

The restore script is very similar to the backup script, and will not be presented in depth.

When run, the script outputs something like this:

```
Using list file /backups/databases.lst.
Using Backup Directory /backups.
Checking old backups
Backing up databases from directory /firebird
Backing up fpc.fb
Creating backup directory /backups/fpc
Examining fpc.fb
First time backup.
23:24:32 : Backing up to /backups/fpc/2011-08-29.fbk...
23:24:32 : backup completed.
Testing Restore of generated backups
Attempting restore of /backups/fpc/2011-08-29.fbk
Backup /backups/fpc/2011-08-29.fbk can be used as-is
23:24:32 Attempting restore of /backups/fpc/2011-08-29.fbk
  (/backups/fpc/2011-08-29.fbk) to /tmp/tmp.W3vLEcvpCX
gbak: ERROR:validation error for column MYKEY, value "*** null ***"
gbak: ERROR: warning -- record could not be restored
gbak:Exiting before completion due to errors
23:24:33 Backup restore of /backups/fpc/2011-08-29.fbk failed!
Removing restored database /tmp/tmp.W3vLEcvpCX using rm
23:24:33 Compressing backup file /backups/fpc/2011-08-29.fbk
23:24:33 Compressed backup from 176K to 60K.
Setting backup file ownership to michael:michael
Removing error list
Removing restore list
Setting log file ownership to www-data:www-data
```

One of the reasons to change the owner of the backups and log files is for instance to set them to the user which is used by the webserver: this allows the webserver to be used to manipulate the logs and backups. Another reason would be to make them accessible by FTP, so they can be downloaded and subsequently deleted from another machine. The scenario in which the files are manipulated by the webserver will be explored in a future contribution.