Database corruption

The following articles provide important information regarding the causes leading to database corruption, as well as ways to recover a corrupt database. We would like to thank the authors for allowing us to publish their articles here.

How to corrupt a database Although Firebird is extremely stable and secure, there are a few things that you should NOT do, as these could result in corrupting the database!

The following tips have been taken from the Firebird Quick Start Guide, © IBPhoenix Publications 2002, 2003. Many thanks to Paul Beach (https://www.ibphoenix.com)!

Modifying metadata tables

Firebird stores and maintains all of the metadata for its own and your user-defined objects in a Firebird database! More precisely, it stores them in relations (tables) right in the database itself. The identifiers for the system tables, their columns and several other types of system objects begin with the characters 'RDB$'.

Because these are ordinary database objects, they can be queried and manipulated just like your user-defined objects. However, just because you can does not say you should. The Firebird engine implements a high-level subset of SQL (DDL - please refer to Data Definition Language for further information) for the purpose of defining and operating on metadata objects, typically through CREATE, ALTER and DROP statements.

It cannot be recommended too strongly that you use DDL - not direct SQL operations on the system tables - whenever you need to alter or remove metadata. Defer the 'hot fix' stuff until your skills in SQL and your knowledge of the Firebird engine become very advanced. A wrecked database is neither pretty to behold nor cheap to repair.

Disabling forced writes

Firebird is installed with forced writes (synchronous writes) enabled by default. Changed and new data are written to disk immediately upon posting.

It is possible to configure a database to use asynchronous data writes - whereby modified or new data are held in the memory cache for periodic flushing to disk by the operating system's I/O subsystem. The common term for this configuration is forced writes off (or disabled). It is sometimes resorted to in order to improve performance during large batch operations.

The big warning here is - do not disable forced writes on a Windows server. It has been observed that the Windows server platforms do not flush the write cache until the Firebird service is shut down. Apart from power interruptions, there is just too much that can go wrong on a Windows server. If it should hang, the I/O system goes out of reach and your users' work will be lost in the process of rebooting.

- Windows 9x and ME do not support deferred data writes.

Disabling Forced Writes on a Linux server

Linux servers are safer for running an operation with forced writes disabled temporarily. Do not leave it disabled once your large batch task is completed, unless you have a very robust fall-back power system.

Restoring a backup to a running database

One of the restore options in the GBAK utility (gbak -r[estore]) allows you to restore a gbak file over the top of an existing database. It is possible for this style of restore to proceed without warning while users are logged in to the database. Database corruption is almost certain to be the result.

Be aware that you will need to design your Admin tools and procedures to prevent any possibility for any user (including SYSDBA) to restore to your active database if any users are logged in. If is practicable to do so, it is recommended to restore to spare disk space using the gbak -c[reate] option and *test the restored database* using isql [or IBExpert]. If the restored database is good, shut down the server. Make a file system copy of the old database and then copy the restored [Database structure | database file] (or files) over their existing counterparts.

Allowing users to log in during a restore

If you do not block access to users while performing a restoreusing gbak -r[estore] then users may be able to log in and attempt to do operations on data. Corrupted structures will result.

Recovering corrupt databases

The following is an excerpt from the successful Russian book, “The InterBase® World” first published in September 2002, with a second edition following in April 2003. The authors of the book are Alexey Kovyazin, developer of IBSurgeon (https://www.ibsurgeon.com) and well-known Russian InterBase® specialist, and Serg Vostrikov, CEO of the Devrace company (https://www.devrace.com).

Here the authors would like to offer you a draft copy of one chapter of this book devoted to recovery of Firebird/InterBase® databases.

They would like to pass on their thanks to all who helped create this guide: Craig Stuntz, Alexander Nevsky, Konstantin Sipachev, Tatjana Sipacheva and all the other kind and knowledgeable members of the Firebird and InterBase® community.

Main causes of database corruption

Unfortunately there is always a probability that any information stored will be corrupted and some of this information will be lost. Databases are not an exception to this rule. In this chapter we will consider the principal causes that lead to Firebird InterBase®/database corruption, some methods of repairing databases and extracting information from them. We will also make recommendations and offer precautions that will minimize the probability of information loss.

First of all, if we speak about database repair we should perhaps first define “database corruption”. A database is usually described as damaged if, when trying to extract or modify some information, errors appear and/or the information to be extracted turns out to be lost, incomplete or incorrect. There are cases when database corruption is hidden and can only be found by testing with special facilities. However there are also real database corruptions, when it is impossible to connect to the database, when adjusted programs send strange errors to the clients (without any data manipulation having occurred), or when it is impossible to restore the database from a backup copy.

Principal causes of database corruption are:

1. Abnormal termination of the server computer, especially an electrical power interruption. For the IT-industry it can be a real blow and that is why we hope there is no need to remind you once again about the necessity of having a source of uninterrupted power supply on your server.

2. Defects and faults on the server computer, especially the HDD (hard disk drive), disk controllers, the computer's main memory and the cache memory of Raid controllers.

3. An incorrect connection string to a multi-client database with one or more users (in versions prior to 6.x). When connecting via TCP/IP, the path to the database must be pointed to a server name: drive:/path/databasename/

For servers on UNIX platforms: servername: /path/databasename/

Using a NetBEUI protocol: servernamedrive:pathdatabasename.

Even when connecting to a database from the computer, on which the database is located and where the server is running, the same specification should be used, renaming the servername as localhost. It is not possible to use mapped drives in the connection specification. If you break one of these rules, the server thinks that it is working with different databases and database corruption is guaranteed.

4. File copy or other file access to the database when the server is running. The execution of the command shutdown, or disconnecting the users in the usual way is not a guarantee that the server is doing nothing with the database. If the sweep interval is not set to 0, garbage collection may be being executed. Generally the garbage collection is executed immediately after the last user disconnects from the database. Usually it takes several seconds, but if many DELETE or UPDATE operations were committed before it, the process may take longer.

5. Using unstable InterBase® server versions 5.1-5.5. The Borland Company officially admitted that there were several errors in these servers and these were removed in the stable upgrade 5.6 only after the release of certified InterBase® 6 was in free-running mode for all clients of servers 5.1-5.5 on its site.

6. Exceeding size restriction of a database file. At the time of writing this, for most existing UNIX platform servers the limit is 2 GB, for Windows NT/2000 - 4 GB, but it is recommended to assume 2 GB. When the database size is approaching its limit, an additional file must be created.

7. Exhaustion of free disk space when working with the database.

8. For Borland InterBase® servers using versions under 6.0.1.6 - exceeding the restriction of the maximum number of generators, according to Borland InterBase® R & D defined as follows (please refer to table 1 below).

Critical number of generators in early InterBase versions				
Version	**Page size=1024**	**Page size=2048**	**Page size=4096**	**Page size=8192**
Pre 6	248	504	1016	2040
6.0.x	124	257	508	1020

For all Borland InterBase® servers - exceeding the permissible number of transactions without executing a backup/restore. The number of transactions that have been made in the database since the last backup and restore can be determined by invoking the utility GSTAT with the key -h parameter NEXT TRANSACTION ID.

According to Ann W. Harrison, the critical number of transactions depends on the page size, and has the following values (please refer to table 2 below):

Critical number of transactions in Borland InterBase servers	
Database page size	**Critical number of transactions**
1024 byte	131 596 287
2048 byte	265 814 016
4096 byte	534 249 472
8192 byte	1 071 120 384

The constraints of Borland InterBase® servers enumerated above are not applicable to Firebird servers except for the earliest versions 0.x, the existence of which has already become history. If you use the final version Firebird 1.0 or above, or InterBase® 6.5-7.x, you should not worry about points 5, 6, 8 and 9 and should instead concentrate your efforts on other causes. Now we will consider the most frequent of these in detail.

Power supply failure

When shutting off the power on the server, all data processing activities are interrupted in the most unexpected and (according to Murphy's law) dangerous places. As a result the information in the database may be distorted or lost. The simplest case is when all uncommitted the data from a client's applications are lost as a result of an emergency server shutdown. After a power-cut restart the server. This analyzes data, makes a note of incomplete transactions related to none of the clients, and cancels all modifications made within the bounds of these «dead» transactions. Actually such behavior is normal and assumed from the start by InterBase® developers.

However power supply interruption is not always followed just by such insignificant losses. If the server was executing a database extension at the moment of power supply interruption, there is a large probability of orphan pages present in the

[Database structure | database file] (pages that are physically allocated and registered on the page inventory page (PIP), upon which it is however impossible to write data).

Only GFIX, the repair and modification tool (we will consider it further on), is able to combat orphan pages in the database file. Actually orphan pages lead to unnecessary use of disk space and, as such,

are not the cause of data loss or corruption. Power loss leads to more serious damages. For example, after shutting off the power and restarting, a great amount of data, including committed data, may be lost (after adding or modification of which the command «commit transaction» was executed). This happens because confirmed data is not written immediately to the database file on disk. The file cache of the operating system (OS) is used for this purpose. The server process gives the data write command to the OS. Then the OS assures the server that all the data has been saved to disk although in reality the data is initially stored in the file cache. The OS doesn't hurry to save this data to disk, because it assumes that there is a lot of main memory left, and therefore delays the slow operation of writing to disk until the main memory is full. Please refer to the next subject - Forced Writes - cuts both ways - for further information.

Forced writes - cuts both ways

In order to influence this situation, tuning of the data write mode is provided in InterBase® 6 and Firebird. This parameter is called FORCED WRITES (FW) and has 2 modes - ON (synchronous) and OFF (asynchronous). FW modes define how Firebird/InterBase® communicates with the disk. If FW is turned on, the setting of synchronous writes to disk is switched on, and confirmed data is written to disk immediately following the COMMIT command, the server waits for writing completion and only then continues processing. If FW is switched off InterBase® doesn't hurry to write data to disk after a transaction is committed, and delegates this task to a parallel thread, while the main thread continues data processing, not waiting until all writes are written to disk.

Synchronous writes mode is one of the most careful options and it minimizes any possible data loss. However it may cause some loss of performance. Asynchronous writes mode increases the probability of loss of a great quantity of data. In order to achieve maximum performance FW OFF mode is usually set. But as a result of power interruption a much higher quantity of data is lost using the asynchronous writes mode than when using the synchronous mode. When setting the write mode you should decide whether a few percentage points of performance are more significant than a few hours of work should power be interrupted unexpectedly.

Very often users are careless with InterBase®. Small organizations save on any trifle, often on the computer server, where the DBMS server and different server programs (not only server) are installed and running as well. If they hang-up people don't think for long, and simply press *RESET* (it happens several times a day). Although InterBase® is very stable with regard to such activities compared with other DBMS, and allows work with the database to start immediately after an emergency reboot, such a procedure is not recommended. The number of orphan pages increases and data lose connections among themselves as a result of faulty reboots. It may still function and continue for a long time, but sooner or later it will come to an end. When damaged pages appear among PIP or generator pages, or if the database header page is corrupted, the database may never open again and become a big chunk of separate data from which it is impossible to extract a single byte of useful information.

Corruption of the hard disk

Hard disk corruptions lead to the loss of important database system pages and/or the corruption of links among the remaining pages. Such corruptions are one of the most difficult cases, because they almost always require low-level interference to restore the database.

Database design mistakes

It is necessary to learn of some mistakes made by database developers that can lead to an impossible database recovery from a backup copy (*.gbk files created by the GBAK program). First of all a careless use of constraints at database level. A typical example is the constraint NOT NULL. Let's suppose that we have a table filled with a number of records. Now using the ALTER TABLE command we'll add one more column to this table and specify that it mustn't contain the non-defined value NULL. Something like this:

```
ALTER TABLE sometable Field/INTEGER NOT NULL
```

In this case there will be no server error as should be expected. This metadata modification will be committed and we won't receive any error or warning message, which creates an illusion of normality.

However, if we backup the database and try to restore it from the backup copy, we'll receive an error message at the phase of restoring (because NULLs are inserted into the column that has NOT NULL constraint, and the process of restoring will be interrupted. (An important note provided by Craig Stuntz: with version InterBase® 7.1 constraints are ignored by default during a restore (this can be controlled by a command-line switch) and nearly any non-corrupt backup can be restored. It's always a good idea to do a test restore after performing a backup, but this problem should pretty much disappear in version 7.1.). This backup copy can't be restored. If the restore was directed to a file having the same name as the existing database (during restoration of the existing database the working file was being rewritten), we'll lose all information.

It has to do with the fact that NOT NULL constraints are implemented by system Triggers which check only incoming data. During restoration, data from the backup copy is inserted into the empty, newly created tables - here we can find inadmissible NULLs in the column with the constraint NOT NULL.

Some developers consider such InterBase® behavior to be incorrect, but others will be unable to add a field with NOT NULL restriction to the database table.

The question about required value by default and filling with this value at the moment of creation was widely discussed by Firebird architects, but it wasn't accepted because of the fact that the programmer is obviously going to fill it according to an algorithm, which is rather complicated and maybe iterative. But there is no guarantee, whether he'll be able to distinguish the records ignored by previous iteration from unfilled records or not.

A similar problem can be caused by a garbage collection fault, caused by the specification of an incorrect path to the database (the cause of corruption 3) at the time of connection, and file access to database files when the server is working with it (the cause of corruption 4), and records wholly filled with NULLs can appear in some tables. It's very difficult to detect these records, because they don't correspond to integrity control restrictions, and operator SELECT just doesn't see them, although they get into the backup copy. If it is impossible to restore for this reason, the GFIX utility should be used (see below), to find and delete these records using non-indexed fields as search conditions. After this try to make a backup copy again and restore the database from it. In conclusion we can say that there are a great number of causes of database corruption and you should always be prepared for the worst - that your database could become damaged for one reason or another. You should therefore be prepared at all times to restore and rescue valuable information.

Precautions and methods of repair

And now we shall consider precautions that guarantee Firebird/InterBase® database security, as well as methods of repairing damaged databases.

Regular backups

In order to prevent database corruption, backup copies should be created regularly (if you want to know more about backup then please refer to Backup and Restore for further information). It's the most trusted method to prevent and combat database corruption. Only a backup gives 100% guarantee of database security. As described above, it is possible get a useless copy as the result of restoring a backup file (i.e. a copy that can't be restored); that's why restoring a base from the copy should not be performed by writing over the script, and a backup must be carried out according to definite rules. Firstly, a backup should be executed as often as possible, secondly it must be serial and thirdly, backup copies must be checked for their restoring capability.

Usually, a backup means that it's necessary to make a backup copy rather often, for example, once every twenty-four hours. The shorter the period is between database backups, the less data will be lost as a result of a fault. The sequence of backups means that the number of backups should increase and should be stored for at least a week. If possible, backups should be written to special devices such as a streamer, but if this is not possible - copy them to another computer. The history of backup copies will help to discover hidden corruptions and cope with an error that perhaps arose some time ago but has only just showed up unexpectedly. It is necessary to check whether it is possible to restore the saved backup without errors or not. This can be checked in only one way - through the test restore process. It should be mentioned that the restore process takes 3 times longer than the backup, and it's difficult to execute restore validation every day for large databases, because it may interrupt the users' work for a few hours (a night break may not be enough).

It would be better if big organizations didn't save at the wrong end and assigned one computer just for these purposes.

In this case, if the server must work with a serious load 24 hours 7 days a week, we can use the SHADOW mechanism for taking snapshots of the database, and performing further backup operations from the immediate copy. When creating a backup copy and then restoring the database from this backup, all data in the database is recreated. This process (backup/restore or b/r) contributes to the correction of most non-fatal errors in the database connected with hard disk corruptions, detecting problems with integrity in the database, cleaning the database of garbage (old versions and fragments of records, incomplete transactions) which decreases the database size considerably.

Regular backup/restore is a guarantee of Firebird/InterBase® database security. If the database is working, then it is recommended to execute backup/restore on a weekly basis. To tell the truth, there are some examples of Firebird/InterBase® databases that are intensively used for some years without a single backup/restore.

Nevertheless, to be on the safe side it's desirable to perform this procedure regularly, especially as it can be easily automated (please refer to Backup and Restore).

If it's impossible to perform a regular backup/restore for certain reasons, then the GFIX tool can be used for checking and restoring the database. GFIX allows you to check and remove many errors without performing a backup/restore.

Using GFIX

The command-line utility GFIX is used for checking and restoring databases. Furthermore GFIX can also execute various database control activities: changing the database dialect, setting and canceling the mode “read-only”, setting cache size for a specific database and also some important functions.

GFIX is committed in command-line mode and has the following syntax:

```
Gfix [ options] db_name
```

Options is a set of options for executing GFIX, db_name is the name of the database for which the operations are to be performed, defined by a set of options. The following table displays the GFIX options related to database repair:

GFIX tool options for database restoration	
Option	**Description**
-**f**[ull]	This option is used in combination with -v and means it’s time to check all fragments of records
-**i**[gnore]	Option makes GFIX ignore checksum errors at the time of validation or database cleaning
-**m**[end]	Marks damaged records as not available, as a result of which they will be deleted during the following backup/restore. This option is used when preparing a corrupted database for backup/restore.
-**n**[o_update]	Option is used in combination with -v for read-only database validation without correcting corruptions
-**pas**[sword]	Option allows the password to be set when connecting to the database. (Note that there is an error in the InterBase documentation: -pa[ssword], the shortcut "-pa" will not work - you need to use "-pas")
-**user**	Option allows the user’s name to be set when connecting to the database
-**v**[alidate]	Option for presetting the database validation when errors are discovered
-**m**[ode]	Option for setting the write mode for the database - for read-only or read/write. This parameter can accept 2 values - read write or read only.
-**w**[rite] {**sync** | **async**}	Option that switches on and off the mode synchronous/ asynchronous forced writes to database. sync - to turn synchronous writes on (FW ON); async -to turn asynchronous writes on (FW OFF);

Here are some typical GFIX examples:

```
gfix -w sync -user SYSDBA -pass masterkey firstbase.gdb
```

In this example we set for our test database, firstbase.gdb, the synchronous writes mode (FW ON). (Of course, this is more useful before corruption occurs). And below is the first command that you should use to check the database after corruption has occurred:

```
gfix -v -full -user SYSDBA -pass masterkey firstbase.gdb
```

In this example we start checking our test database (option -v) and specify that fragments of records must be checked as well (option -full). Of course, it is more convenient to set various options for the checking and restoring process using IBExpert or another GUI interface, but we'll review the functions of database recovery using command-line tools. These tools are included in Firebird and InterBase® and you can be sure that their behavior will be the same on all OS running InterBase®. It is vital that they always be close to the server. Besides the existing tools, allowing you to execute database administration from a client's computer, you can use the Services API, which isn't supported by the InterBase® server Classic architecture. That means you need to use a third party product (such as IBExpert or other administration tool) with the Superserver architecture.

Repairing a corrupt database

Let's assume there are some errors in our database. Firstly, we have to check the existence of these errors; secondly, we have to try to correct these errors. We recommend the following procedure:

You should stop the InterBase® server if it's still working and make a copy of the file or the database files. All the restore activities should only be performed with a database copy, because it may lead to an unsatisfactory result, and you'll have to restart the restore procedure (from a starting point). After creating a copy we'll perform the complete database validation (checking fragments of records).

We should execute the following command for this (or use the IBExpert Services menu item Database Validation):

```
gfix -v -full corruptbase.gdb -user SYSDBA -password
```

In this case corruptbase.gdb - is a copy of the damaged database. This command will check the database for any structural corruption and produce a list of unsolved problems. If such errors are detected, we'll have to delete the damaged data and get ready for a backup/restore using the following command (or using the IBExpert Services menu item Backup Database):

```
gfix -mend -user SYSDBA -password your_masterkey corruptbase.gdb
```

After committing this command you should check if there are any errors left in the database. Run GFIX using the options -v -full, and when the process is over, perform a database backup:

```
gbak -b -v -ig -user SYSDBA -password corruptbase.gdb corruptbase.gbk
```

This command performs a database backup (option -b) and we'll get detailed information about the backup process execution (option -v). Errors with regard to checksums will be ignored (option -ig).

Please refer to GBAK and Backup Database for further information.

If some errors are found during the backup, you should start it in another configuration:

```
gbak -b -v -ig -g -user SYSDBA -password
corruptbase.gdb corruptbase.gbk
```

Where option -g will switch off garbage collection during the backup. This often helps to solve backup problems.

Also it may be possible to make a backup of a database if it is set in the read-only mode beforehand. This mode prevents writing any modifications to the database and sometimes helps to complete the backup of a damaged database. For setting a database to read-only mode, you should use the following command (or the IBExpert Services menu item Database Properties):

```
gfix -m read_only -user SYSDBA -password masterkey
Disk:Pathfile.gdb
```

Following this, you should try to perform the database backup again using the parameters given above (or the IBExpert Services menu item Backup Database).

If the backup was completed successfully, you should restore the database from the backup copy, using the following command (or the IBExpert Services menu item Restore Database):

```
gbak -c -user SYSDBA -password masterkey Disk:Pathbackup.gbk
Disk:Pathnewbase.gdb
```

When you are restoring the database, you may come across some problems, especially when creating the indices.

In this case the -inactive and -one_at_a_time options should be added to the restore command. These options deactivate indices when creating from the database backup and commit data confirmation for each table. Alternatively use the IBExpert Services menu item Backup Database.

Extract data from a corrupt database

It is unfortunately possible that even the operations previously mentioned in this section do not lead to a successful database recovery.

It means that the database is seriously damaged or it cannot be restored as a single entity, or a huge effort must be made to recover it. For example, it is possible to execute a modification of system metadata, use non-documented functions and so on. It is very hard, time-consuming and ungrateful work with doubtful chances of success. If at all possible, try to evade it and use other methods. If a damaged database opens and allows you to perform reading and modification operations with some data, you should take advantage of this possibility and save the data by copying it to a new database, and say good-bye to the old one for good.

So, before transferring the data from the old database, it's necessary to create a new destination database. If the database hasn't been altered for a long time, you can use the old backup, from which metadata can be extracted for creating the new database. Based on these metadata it is necessary to create a data destination and start copying the data. The main task is to extract the data from the damaged database. Then we'll have to allocate the data in a new base, but that's not very difficult, even if we have to restore the database structure from memory.

When extracting data from tables, you should use the following algorithm of operations:

1. At first you should try to execute SELECT * from table N. If it ran normally you could save the data you've got in the external source. It's better to store data in a script (using the IBExpert Tools menu item Extract Metadata for example), as long as the table doesn't contain blob fields. If there are blob fields in the table, then this data should be saved to another database by a

client program that will play the role of mediator.
2. If you failed to retrieve all data, you should delete all the indices and try again. In fact, indices can be deleted from all the tables from the beginning of the restore, because they won't be needed any more.
3. Of course, if you don't have a metadata structure which is the same as that of the corrupted database, it's necessary to input a protocol of all operations that you are doing with the damaged database source.

If you cannot read all the data from the table after deleting the indices, try to execute a range query by primary key, i.e. select a definite range of data. For example:

```
SELECT * FROM table N WHERE field_PK >=0 and field_PK <=10000
```

Field_PK here is a primary key.

InterBase® has page data organization and that's why a range query of values may be rather effective.

Nevertheless it works because we can expel data from the query from damaged pages and fortunately read the other ones. You may recall our thesis that there is no defined order of storing records in SQL.

Really, nobody can guarantee that an unordered query will, during restarts, return the records in the same order, but nevertheless the physical records are stored within the database in a defined internal order. It's obvious that the server will not mix the records purely to abide to SQL standards. Try to use this internal order when extracting data from a damaged database. Vitaliy Barmin, an experienced Russian InterBase® developer reported that in this way he managed to restore up to 98% of information from an unrecoverable database (there were a great number of damaged pages). Thus, data from a damaged database must be moved to a new database or into external sources such SQL scripts. When you copy the data, pay attention to [Generator | generator]] values in the damaged database (they must be saved for restarting proper work in the new database. If you don't have a complete copy of the metadata, you should extract the texts of stored procedures, triggers, constraints and the definition of indices.

Restoring hopeless databases

In general, restoring a database can be very troublesome and difficult and that's why it's better to make a backup copy of the database and then restore the damaged data and whatever has happened, you shouldn't despair because a solution can be found even in the most difficult situations. And now we'll consider two cases.

The first case (a classic problem): A backup that can't be restored because of having NULL values in a column with NOT NULL constraints (the restore process was run over the working file). The working file was erased and the restore process was interrupted because of an error. And as a result of thoughtless actions the result was a great amount of useless data (that can't be restored) instead of a backup copy. But a solution was found. The programmer managed to recollect which table and which column contained the constraint NOT NULL. The backup file was loaded to a hexadecimal editor. And a combination of bytes, corresponding to the definition of this column, was found by searching. After innumerous experiments it turned out that the constraint NOT NULL adds 1 somewhere near the column name. In the HEX-editor this 1 was corrected to 0 and the backup copy was restored. Following this, the programmer memorized once and for all how to execute the backup process and

restore successfully!

The second case: The situation was catastrophic. The database corrupted on the extension phase because of lack of disk space. When increasing the database size, the server creates a series of critically important pages (for example, Transaction Inventory Page and Page Inventory Page, additional pages for RDB$Pages relations) and writes them down at the end of database.

As a result, the database could not be opened, neither by administration facilities nor using the utility GBAK. And when we tried to connect to the database, an error message (Unexpected end of file) appeared.

When we ran the utility GFIX strange things happened: The program was working in an endless cycle. When GFIX was working, the server was writing errors to log (file InterBase® log) at high speed (around 100 Kb per second). As a result, the log file filled all the free disk space very quickly. We even had to write a program that erased this log by timer. This process lasted for a long time - GFIX was working for more than 16 hours without any results.

The log was full of the following errors: Page XXX doubly allocated. When starting InterBase® sources (in file val.c) there is a short description of this error. It says that this error appears when the same data page is used twice. It's obvious that this error is a result of corruption of critically important pages.

As a result, after several days of unsuccessful experiments, all attempts to restore the data in the standard way were abandoned. Which is why we had to use a low-level analysis of the data stored in the damaged database.

Alexander Kozelskiy, head of Information Technologies at East View Publications Inc, had the idea of how to extract information from similar unrecoverable databases. The method of restoring, arrived at as a result of our research, was based on the fact that a database has page organization and data from every table is collected by data pages. Each data page contains an identifier of the table for which it stores data. It was especially important to restore data from several critical tables. There was data from similar tables, received from an old backup copy that worked perfectly and could be used as a model. This database sample was loaded into an editor of hexadecimal sources and then we searched for the patterns of the data that interested us. This data was copied into a buffer in hexadecimal format and then the remains of the damaged database were loaded into the editor. A sequence of bytes corresponding to the sample was found in the damaged database, and the page was analyzed (on which this sequence was found).

At first we needed to define the start page, which wasn't difficult because the size of the database file is divisible by the data page size. The number of current bytes divided by page size - 8192 bytes, approximates the result to integer (and we obtained the number of the current page). Then the number of current page was multiplied by page size and we got the number of bytes corresponding to the beginning of the current page. Having analyzed the header, we defined the type of page (for pages with data the type is 5 - please refer to the file ods.h from the set of InterBase® sources as well as the identifier of the necessary table.

Then a program was written, that analyzed the whole database, collected all the pages for the necessary table into one single piece and moved it to file. Thus, once we had the data we initially needed, we began analyzing the contents of the selected pages.

InterBase® uses data compression widely in order to save space. For example, a string such as VARCHAR containing an ABC string, stores a sequence of following values: string length (2 bytes), in

our case it is 0003, and then the symbols themselves followed by a checksum. We had to write an analyzer of the string as well as other database types that converted data from hexadecimal format into an ordinary view. We managed to extract up to 80% of the information from several critical tables using a “manual” method of analyzing the database contents. Later, on the basis of this experience, Oleg Kulkov and Alexey Kovyazin, one of the authors of this book, developed the utility InterBase® Surgeon which performs direct access to the database, bypassing the InterBase® engine and enables you to read directly and interpret the data within the InterBase® database in a proper way.

Using InterBase® Surgeon, we have managed to detect the causes of corruption and restore up to 90% of absolutely unrecoverable databases, which can't be opened by InterBase® and restored by standard methods. This program can be downloaded from the official site https://www.ib-aid.com.