



HQbird 2022 User Guide

1.09, by 08.01.2022

Table of Contents

Preface	6
About this Guide	6
About IBSurgeon	6
1. Overview of HQbird	7
1.1. What is HQbird	7
1.2. What's new in HQbird 2022	7
1.3. Feature matrix	8
2. Installation of HQbird	11
2.1. Installing HQbird Server on Windows	11
2.1.1. Silent installation on Windows	11
2.2. Installing HQbird Server for Windows using the installer	13
2.3. Installing HQbird Administrator on Windows	26
2.3.1. How to install community version of Firebird on Windows	28
2.4. Installing HQbird Server on Linux	29
2.4.1. Installation of HQbird with Firebird 2.5 on Linux	29
2.4.2. Installation of HQbird with Firebird 3.0 on Linux	30
2.4.3. Installation of HQbird with Firebird 4.0 on Linux	31
2.4.4. Installation of HQbird Standard on Linux	31
2.4.5. Firewall settings	32
2.5. Upgrade existing HQbird version	32
2.6. Registration of HQbird	34
2.6.1. How to activate HQbird	34
2.6.2. Offline Activation	37
2.6.3. Activation in web interface	38
3. Configuration of HQbird	39
3.1. Initial configuration of HQbird FBDataGuard (backups, monitoring, alerts,etc)	39
3.1.1. Launch web-console	39
3.1.2. Auto discovery feature of FBDataGuard	40
3.1.3. Firebird server registration	41
3.1.4. Firebird database registration	43
3.1.5. Email alerts in HQbird FBDataGuard	46
3.1.6. Next steps with FBDataGuard	49
3.2. Monitoring and maintenance configuration in FBDataGuard	49
3.2.1. Overview of web-console	49
3.2.2. Server: Active server	51
3.2.3. Server: Auto updates	52
3.2.4. Server: Replication Log	53
3.2.5. Server: Server log	54

3.2.6. Server: Temp files	55
3.2.7. Server: Firebird server folder	56
3.2.8. Server: HQbird Output Folder	58
3.2.9. Database: General configuration	58
3.2.10. Database: Transactions	59
3.2.11. Database: Lockprint	60
3.2.12. Database: Index statistics recalculation	64
3.2.13. Database: Verified Backup	65
3.2.14. Database: Incremental Backup	70
3.2.15. Database: Dump Backup	74
3.2.16. Database: RestoreDB	74
3.2.17. Database: Transfer Replication Segments	79
3.2.18. Database: Transfer Files	83
3.2.19. Database: Pump Files	87
3.2.20. Database: File Receiver	90
3.2.21. Database: Low-level metadata backup	93
3.2.22. Database: Validate DB	93
3.2.23. Database: Sweep Schedule	94
3.2.24. Database: Delta	95
3.2.25. Database: Disk space	96
3.2.26. Database: Database statistics	97
3.2.27. Database: Replica Check	97
3.3. FBDataGuard tips&tricks	98
3.3.1. Path to FBDataGuard configuration	98
3.3.2. Adjusting web-console port	99
3.3.3. How to change password for Admin user	99
3.3.4. Guest user for HQbird FBDataGuard	99
3.4. Appendix: CRON Expressions	100
3.4.1. CRON Format	100
3.4.2. Special characters	100
3.4.3. CRON Examples	101
3.4.4. Notes	102
3.5. Configuring firebird.conf for the best performance	102
4. Monitoring	104
4.1. Monitoring with HQbird FBDataGuard	104
4.1.1. Overview	104
4.1.2. Automatic monitoring with FBDataGuard (Trace API and MON\$)	105
4.1.3. What can we see in the performance report?	108
4.1.4. Automatic monitoring of long-running active transactions	110
4.1.5. How to select a tool for detailed monitoring	111
4.2. Monitoring with MON\$ tables: HQbird MonLogger	112

4.2.1. Aggregated performance statistics for users attachments	113
4.2.2. Aggregated performance statistics for statements	114
4.2.3. Attachments	116
4.2.4. Transactions	117
4.2.5. Statements	118
4.3. Advanced Monitor Viewer	118
4.3.1. FetchesReadsWritesMarks	120
4.3.2. Users	121
4.3.3. Traces	121
4.3.4. RAM and CPU Windows	122
4.3.5. RAM and LoadAvg Linux	122
4.3.6. Transactions	122
4.3.7. Lock Table Info	123
4.4. Monitoring with HQbird FBScanner	123
4.4.1. What is FBScanner?	123
4.4.2. Issues that FBScanner can help to resolve	124
4.4.3. Performance Impact	124
4.4.4. How to configure FBScanner for local computer?	124
4.4.5. How to setup FBScanner for remote computer?	125
4.4.6. How to setup logging?	128
4.4.7. How to analyze FBScanner log?	135
4.4.8. How to track 10054 errors, disconnects and failed login attempts?	140
4.4.9. Backup/restore and mass load operations	141
4.4.10. Real-Time Monitoring: FBScanner Viewer	142
4.4.11. FBScanner Feature Matrix	153
5. Database structure analysis	158
5.1. Overview of Firebird database structure	158
5.2. How to analyze database structure with HQbird Database Analyst (IBAnalyst)	159
5.2.1. How to get statistics from Firebird database in right way	160
5.2.2. Summary View	162
5.2.3. Tables view	166
5.2.4. Index view	169
6. HQbird Enterprise configuration: Native Firebird replication and PerformanceEnhancements	173
6.1. What is HQbird Enterprise?	173
6.1.1. Compatibility	173
6.1.2. How the replication works	173
6.2. Installation	173
6.3. Asynchronous replication for Firebird	173
6.3.1. Step 1: Configure HQbird for replication at the master	175
6.3.2. Step 2: Create a copy of master database	180
6.3.3. Step 3: Setup database for async replication at the replica(slave) server	181

6.4. Automatic initialization and re-initialization of replica	182
6.4.1. How re-initialization works	183
6.4.2. Troubleshooting asynchronous replication	184
6.5. Synchronous replication for Firebird	186
6.5.1. Steps to setup synchronous replication	187
6.5.2. Synchronous replication at master and replica	187
6.5.3. Replication parameters for testing synchronous replication	188
6.6. How to manually create replica of the database?	189
6.6.1. Creating copy online (with nbackup)	189
6.6.2. What is {DATABASEGUID}?	190
6.6.3. How to set replica database to the master mode	191
6.7. How to distinguish master database from replica	192
6.7.1. Using gstat -h	192
6.7.2. With SQL query to the context variable	192
6.8. Optional parameters for replication	193
7. Performance enhancements	196
7.1. Pool of external connections	196
7.2. Cached prepared statements	197
7.3. TempSpaceLogThreshold: monitoring of big sorting queries and BLOBs	198
7.4. SortDataStorageThreshold: REFETCH instead SORT for wide record sets	199
7.5. Multi-thread sweep, backup, restore	200
7.6. BLOB_APPEND function	202
7.7. Transform LEFT joins into INNER	206
8. Encryption support	207
8.1. OpenSSL files	207
8.1.1. How to encrypt and decrypt Firebird database	207
9. Authentication plugin for Execute Statement On External	215
9.1. Installation of authentication plugin for ESOE	215
9.1.1. Authentication plugin files	215
9.1.2. Configuration	215
9.1.3. How to test	218
10. RSA-UDR — security functions to sign documents and verify signatures	219
10.1. How to use RSA-UDR security and conversion functions	221
11. SPLIT-UDR — procedures to splitting lines by separator	223
12. OCR-UDR — function to recognizing text from images	226
12.1. Example of using OCR-UDR	226
13. LK-JSON-UDR — building and parsing JSON	228
13.1. Install UDR lkJSON	228
13.2. How it works?	228
13.3. Description of PSQL packages from UDR-lkJSON	229
13.3.1. JS\$BASE package	229

13.3.2. JS\$BOOL package.....	230
13.3.3. JS\$CUSTLIST package.....	231
13.3.4. JS\$FUNC package.....	233
13.3.5. JS\$LIST package.....	234
13.3.6. JS\$METH package.....	236
13.3.7. JS\$NULL package.....	237
13.3.8. JS\$NUM package.....	238
13.3.9. JS\$OBJ package.....	239
13.3.10. JS\$PTR package.....	242
13.3.11. JS\$STR package.....	243
13.4. Examples.....	245
13.4.1. Building JSON.....	245
13.4.2. Parse JSON.....	247
14. NANODBC-UDR — working with ODBC Data Sources.....	254
14.1. Install UDR nanodbc.....	254
14.2. How it works?.....	254
14.3. Description of PSQL packages from UDR-nanodbc.....	255
14.3.1. NANO\$UDR package.....	255
14.3.2. NANO\$CONN package.....	256
14.3.3. NANO\$TNX package.....	259
14.3.4. NANO\$STMT package.....	259
14.3.5. NANO\$RSLT package.....	270
14.3.6. NANO\$FUNC package.....	280
14.4. Examples.....	283
14.4.1. Fetching data from a Postgresql table.....	283
14.4.2. Inserting data into a Postgresql table.....	285
14.4.3. Batch insert into a Postgresql table.....	286
14.4.4. Using transaction.....	288
15. Firebird Streaming.....	290
15.1. Json plugin description.....	291
15.2. Sql plugin description.....	293
15.3. Rabbitmq plugin description.....	293
Appendix A: Support contacts.....	295

Preface

About this Guide

HQbird User Guide contains detailed description of functions and features of HQbird – advanced Firebird distribution, including with configuration examples and best practices recommendations.

About IBSurgeon

IBSurgeon (<https://www.ib-aid.com>) was founded in 2002 with the idea to provide InterBase and Firebird developers and administrator with services and tools focused on databases safety, performance and availability. In Russia, IBSurgeon is mostly known as iBase.ru, famous by its Russian InterBase and Firebird portal www.ibase.ru. IBSurgeon is a member of Firebird Foundation and, as a member of Technical Task Group, has strong relationship with Firebird Project, with direct representatives in Firebird-Admins and in Firebird Foundation Committee. Today, IBSurgeon serves thousands of companies worldwide with emergency, optimization and maintenance tools and various services. Our clients are medical institutions, financial organizations and ISVs in Germany, Brazil, Russia and other countries, and all who have applications based on Firebird and/or InterBase. The flagship project of IBSurgeon is HQbird, the advanced distribution of FirebirdSQL for big databases with enterprise features.

Chapter 1. Overview of HQbird

1.1. What is HQbird

HQbird is an advanced distribution of Firebird for enterprises, with the following list of features:

- Native one-to-many replication (2.5, 3.0, 4.0)
 - Replicates databases with 1500+ connections
 - Asynchronous replication with 10 seconds delay
 - Synchronous replication
 - No triggers or other changes in schema required, DDL support, online re-initialization
- multi-thread backup, restore, and sweep;
- support for encryption;
- pool of execute on external connections;
- cached prepared statements;
- optimized configurations;
- backups automation, including cloud backups;
- database health monitoring;
- automatic performance and transactions reports;
- advanced transactions and queries monitoring (Trace, MON\$ and FBScanner);
- database structure analysis;
- recovery toolset and database development GUI.

Also HQbird includes a performance self-test to measure Firebird performance on the specific hardware and OS configuration.

HQBird contains 2 parts: Server and Admin. The Server part has versions for Windows and Linux, and the administration part works on Windows only.

There are 3 editions of HQbird: Standard, Professional and Enterprise.

1.2. What's new in HQbird 2022

HQbird 2022 is a major new release that adds support for Firebird 4 and adds a number of significant features:

1. Performance improvements:
 - Optimizer improvements for automatic recognition of implicit INNER join (transform LEFT JOIN to INNER JOIN if queries are equivalent)
 - Temporary blob caching to speed up blob operations up to 15% and prevent abnormal database growth

- blob_append function for faster blob concatenation, blob operations up to 18x faster
- Multithreaded restore in Firebird 4 HQbird 2022

2. Streaming changes to the database:

- Each change (such as INSERT, UPDATE or DELETE) for each table can be sent outside the database by a commit event
- Ability to send changes as JSON via RabbitMQ and similar software

3. A set of additional UDRs:

- Text recognition (OCR) inside the database in queries, stored procedures and triggers, with support for dictionaries.
- UDR for working with JSON (construction and parsing)
- Split UDR to quickly split a string by delimiter
- UDR for working with external data sources via ODBC

4. The most complete performance monitoring, based on MON\$ and trace data, makes it easy to identify performance problems, from long transactions to slow or too frequent requests.

In addition, HQbird 2022 still offers advanced replication functionality, external connection pooling, prepared statements pool, and other features necessary when working with large databases under critical load.

1.3. Feature matrix

Below there is the feature matrix for HQbird editions.

Features	Editions		
	Standard	Professional	Enterprise
Optimized configurations	X	X	X
Backups	X	X	X
Automatic Performance Reports	X	X	X
SQL queries tracking	X	X	X
Transactions tracking	X	X	X
Database structure analysis	X	X	X
Performance self-test suite		X	X
SQL development, design&debugging		X	X
Recovery			X
Replication and high availability			X
Pool of Execute On External statements			X
Cached prepared statements			X
Multi-thread backup, sweep, restore			X

Encryption support			X
Enhanced authentication and security features			X

It is possible to single out components in the HQbird modules that are responsible for certain operations (such as backup, monitoring, database repair).

The table below shows how features are distributed among the HQbird modules:

Features	Modules	Description
Backup (automated verified and incremental backup)	FBDataGuard	FBDataGuard runs on the server side and performs all kinds of backup
Optimized Configurations (balanced, read-intensive and write-intensive)	FBDataGuard, a collection of optimized configurations	The optimized configuration file can be customized on the basis of recommendations from FBDataGuard
Performance Test Suite (hardware score)	Performance Test Suite	The test measures the hardware performance
Monitoring SQL Queries (MON\$, TraceAPI and FBScanner)	Performance Monitor, MON\$Logger, FBScanner	Three different monitoring methods are used in different scenarios
Health Monitoring (online validation, database health check, log analysis)	FBDataGuard	Everything is carried out on the server. FBDataGuard sends notifications by e-mail.
Transaction Tracking (dynamic analysis of transaction markers)	FBDataGuard, Transaction Monitor, MON\$Logger	FBDataGuard tracks problems with transactions, Transaction Monitor and MON\$Logger show the dynamics of changes and the current status of active transactions.
Database Structure Analysis (table and index sizes, fragmentation, versioning, etc.)	Database Analyst	Database Analyst analyses the database structure in detail and shows warnings and recommendations.

Features	Modules	Description
SQL Development & Debugging (a GUI tool for developing databases and queries)	SQL Studio	SQL Studio is a powerful tool for developing and debugging database objects and SQL queries.
Recovery (database recovery, backup recovery, record undeleting)	FirstAID, FBDataGuard, IBBackupSurgeon, IBUndelete	FirstAID repairs databases when they get corrupted, FBDataGuard stores important metadata thus increasing the chances of successful repairs, IBBackupSurgeon exports data from corrupted backup copies. IBUndelete can undo records deletion.
High Availability (replication)	The HQbird Enterprise edition includes replication and high availability tools.	
Performance improvements (pool of Execute On External and Cached prepared)	HQbird Enterprise includes performance improvements.	
Multi-thread backup, sweep, restore	HQbird Enterprise	
Encryption support	HQbird Enterprise	
Enhanced authentication and security features	HQbird Enterprise	

Chapter 2. Installation of HQbird

HQbird contains 2 parts: Server and Admin. Let's consider how to install them.

2.1. Installing HQbird Server on Windows

HQbird Server 2022 includes Firebird 2.5, 3.0 and 4.0 with replication, multi-thread support and other enhancements as part of its installer, so Firebird must be installed as part of HQbird. It is mandatory to install Firebird bundled with HQbird Server installer, if you plan to use replication (it also requires HQbird Enterprise license, full or trial) and other enhancements.

Optionally you can choose HQbird Standard, which will not install Firebird, so you can use previously installed Firebird – in this case, make sure that installed version is compatible (2.5.2, 2.5.5, 2.5.6, 2.5.7, 2.5.8, 2.5.9, 3.0.x, 4.0.x).

Please note, that only Firebird 2.5, 3.0 and 4.0 are fully supported in HQbird, the old Firebird versions are supported partially. We offer the comprehensive [Firebird migration service](#) with guaranteed and fast result to migrate Firebird to the latest version.

2.1.1. Silent installation on Windows

The fastest way to install HQbird is to use the silent installation command.

In the example below we will install HQbird Enterprise with Firebird 4.0 into *c:\HQbird*, configuration will be *c:\HQbirdData\config*, output in *c:\HQbirdData\output*.

```
HQBirdServer2022.exe /VERYSILENT /SP- /TYPE="hqbird40x64" /DIR="C:\HQbird2022"
/CONFIGDIR=C:\HQBirdData\config /OUTPUTDIR=C:\HQBirdData\output
```

The following parameters are mandatory to perform the silent installation:

- /VERYSILENT /SP - options to perform the silent installation
- /TYPE – what HQbird version should be installed. If you are doing silent upgrade, make sure the version is the same as it was installed previously.
 - "HQBird25x64" - "HQbird Enterprise (with Firebird 2.5 x64)";
 - "HQBird30x64" - "HQbird Enterprise (with Firebird 3.0 x64)";
 - "HQBird40x64" - "HQbird Enterprise (with Firebird 4.0 x64)".
- /DIR - where to install HQBird. If you are doing silent upgrade, make sure the version is the same as it was installed previously.
- /CONFIGDIR – where to store configuration data for HQbird.
- /OUTPUTDIR – where to store output data (default location for backups, performance reports, etc).

Optional parameters for the silent installation of HQbird:

- /fbport=3050 - port for Firebird to be installed with HQbird Enterprise

- /LOG=C:\temp\HQBirdServerSetup.log - where to store installation log
- DataGuard parameters:
 - /DGPORT=8082 – port for web interface of HQbird (FBDataGuard)
 - /DGLOGIN=admin – login for web interface of HQbird (FBDataGuard)
 - /DGPASSWORD=strong password – password for web interface of HQbird (FBDataGuard)
- Automatic registration parameters:
 - /REGEMAIL=youremail@company.com - email to perform the automatic registration of HQBird
 - /REGPASS=yourpassword – password from IBSurgeon Deploy Center account to register HQbird
 - /REGTYPE=S|E|T == Standard, Enterprise, Trial – license type, must be specified if you need to register HQbird during the installation
- Offline registration (incompatible with REG **)
 - /REGUIK=<uik filename>
 - /REGUNLOCK=<unlock filename>

Must be set in pairs, both are required!

```
/REGUIK="z:\HQBird\test\uik" /REGUNLOCK="z:\HQBird\test\unl"
```

- Email alerts parameters:
 - /EAHOST=smtp.company.com – SMTP server for email alerts
 - /EAPORT=25 – SMTP port for email alerts
 - /EALOGIN=support – SMTP login to send email alerts
 - /EAPASSWORD=psw – SMTP password to send email alerts
 - /EATO=support@email.to – where to send email alerts
 - /EAFROM=someemaildg@company.com – from address
 - /EAENABLED=true – enable or disable email alerts
 - /EADEFALT=true – send a copy of email alerts to IBSurgeon Control Center
- Built-in FTP server parameters:
 - /FTPENABLED=true – enable or disable FTP server
 - /FTPPORT=8721 - FTP port
 - /FTPLOGIN=admin2 - FTP login
 - FTPPASSWORD=strong password2 - FTP password

Please note, that in a case of error, for example, if you are trying to run silent installation to install HQbird to the location which is different from the current location, the error message window will popup and installation will be canceled.

2.2. Installing HQbird Server for Windows using the installer

Download HQbird from <https://ib-aid.com/en/download-hqbird>

The distribution package of HQbird server is the same for the 32-bit and 64-bit versions of the Firebird engine.

Make sure that HQbird installer is signed with valid IBSurgeon certificate («iBase LLC») and run it:

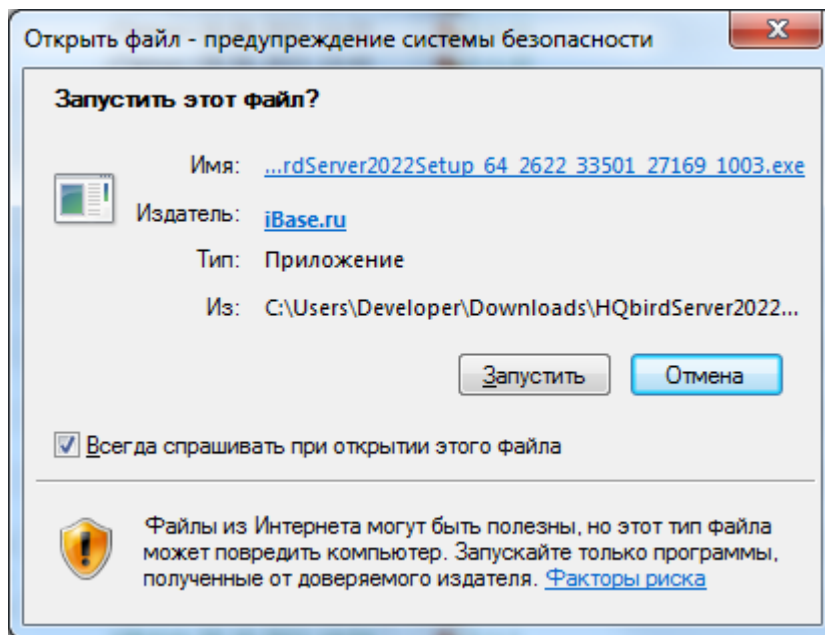


Figure 1. Installer is signed with IBSurgeon digital signature: iBase LLC

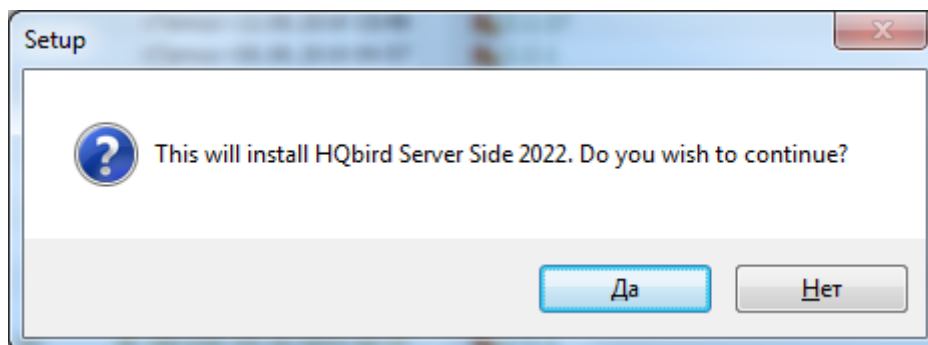


Figure 2. Confirmation to start the installation process

The HQbird Server Side installation wizard will be launched after that and it will take you through several steps, such as agreeing to the license agreement and selecting the installation folder.

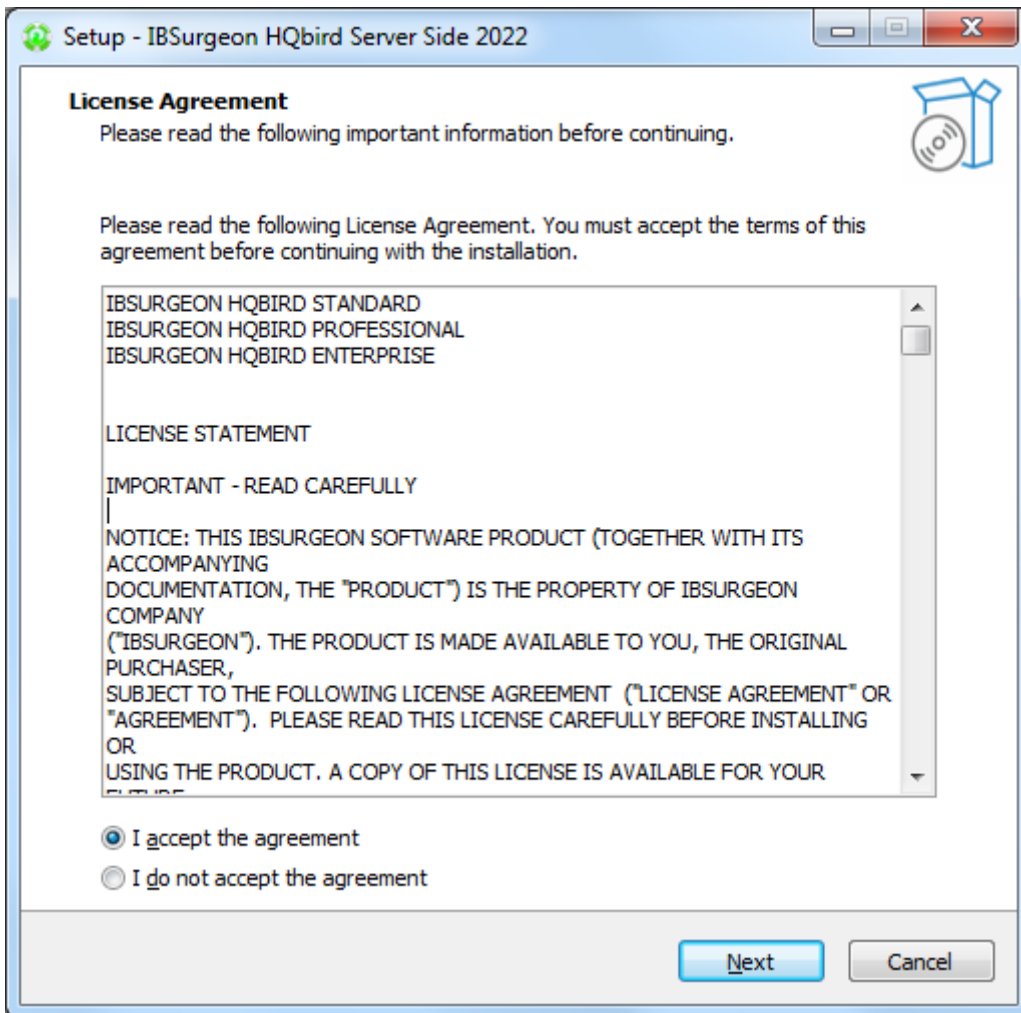


Figure 3. Licence agreement

At first, the installer will ask you where to install HQbird:

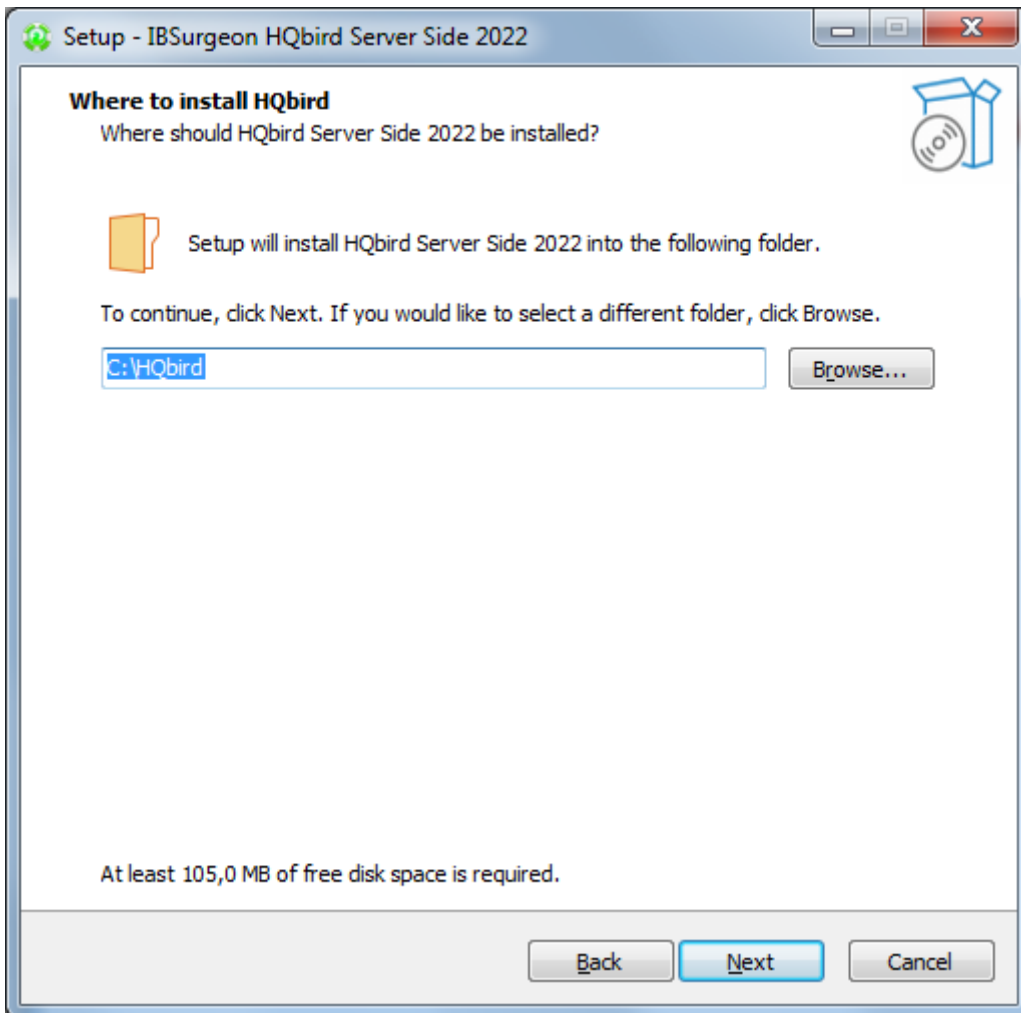


Figure 4. Where to install HQbird

We recommend to use the default location `c:\HQbird`, but you can use any suitable location.

After that, you should select folders for storing configuration files, backup copies of databases, statistics and HQbird log files:

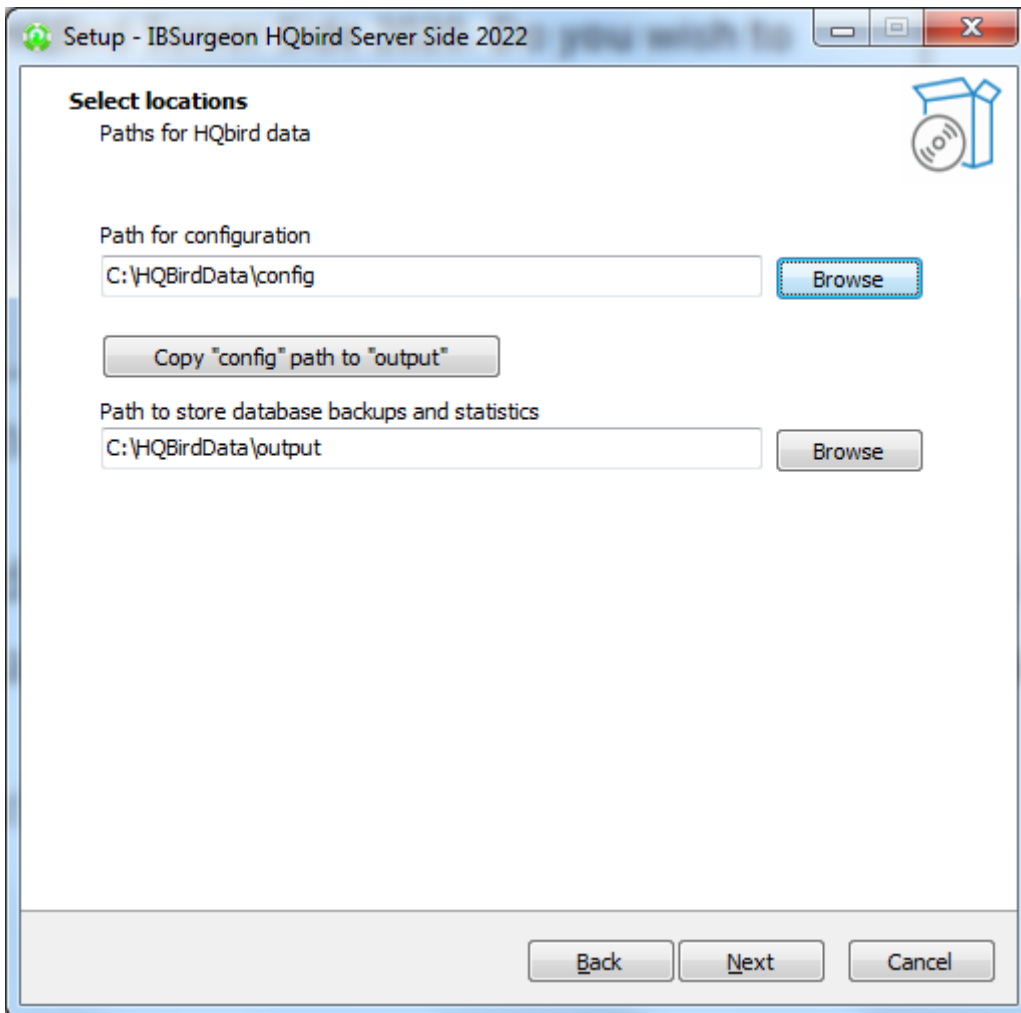


Figure 5. Select folders for HQbird configuration and log files

By default, the installation wizard offers to create folders for configuration and log files in `C:\HQbirdData`.



Usually, we recommend selecting a disk with a large amount of free space for this purpose, but you can configure it later.

If configuration files already exist in the selected location, the installation wizard will display the corresponding warning:

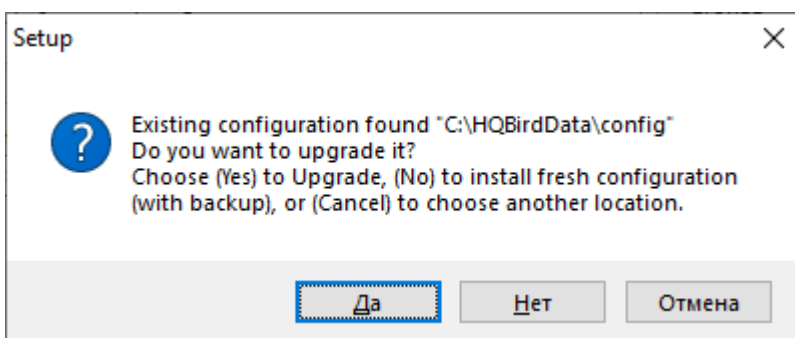


Figure 6. Warning about existing configuration files

We recommend the automatic upgrade, so default answer should be Yes.

However, you can choose to create fresh configuration of HQbird, and click No – in this case the

installer will warn you that existing configuration files will be moved:

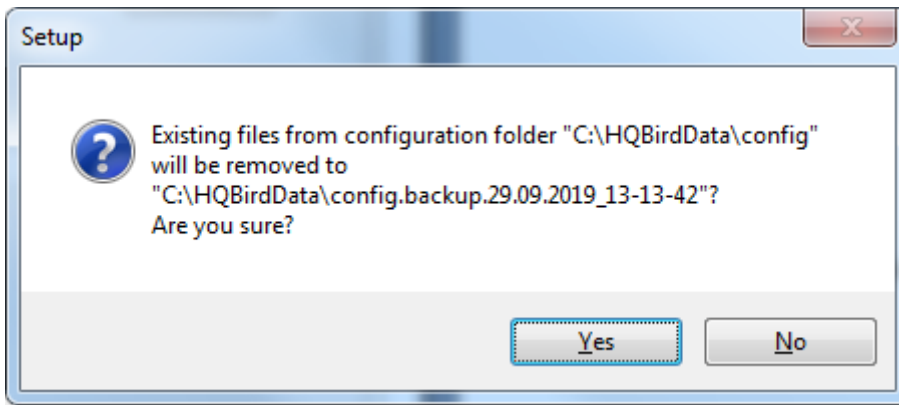


Figure 7. Confirmation of backup

In case of choosing Cancel, you need to specify the different location for the configuration and output/backup files.

After you confirm it, the folder with the existing configuration files will be renamed and the installation will continue.

After that, you will see the installation step where you can select components to be installed:

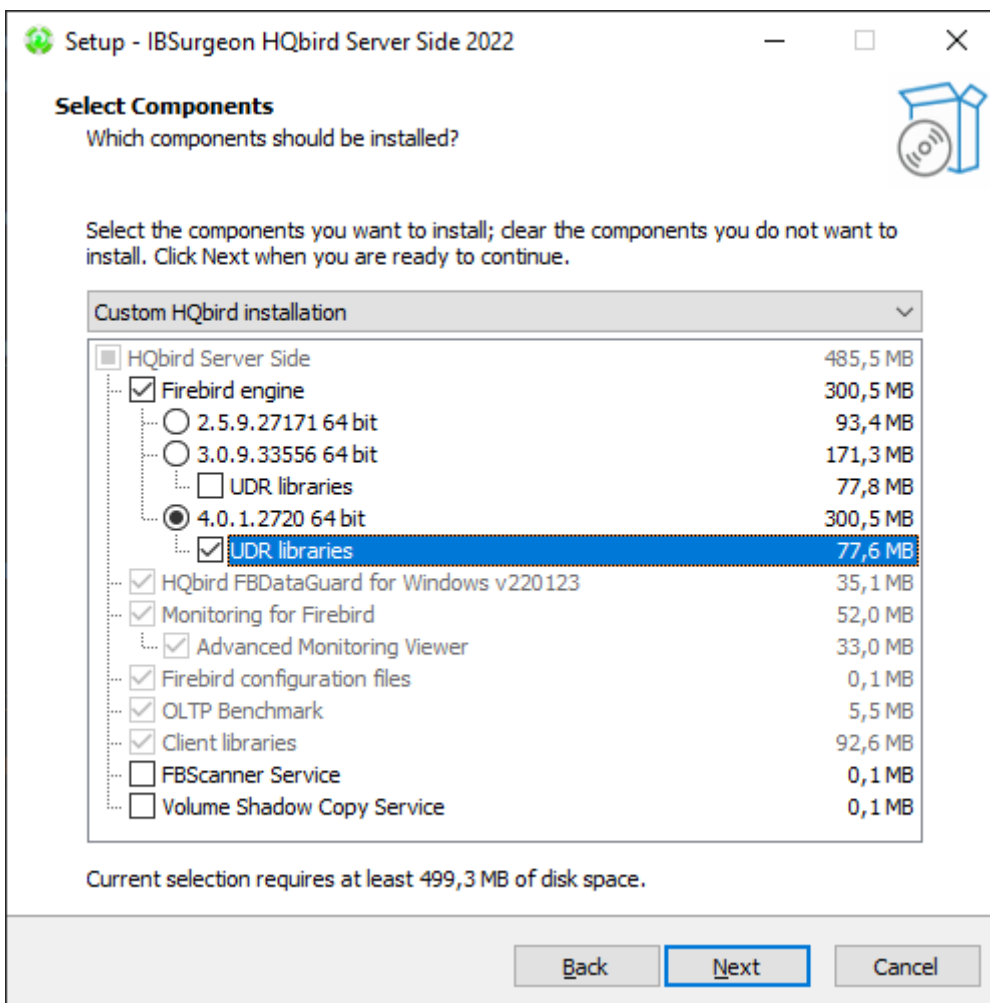


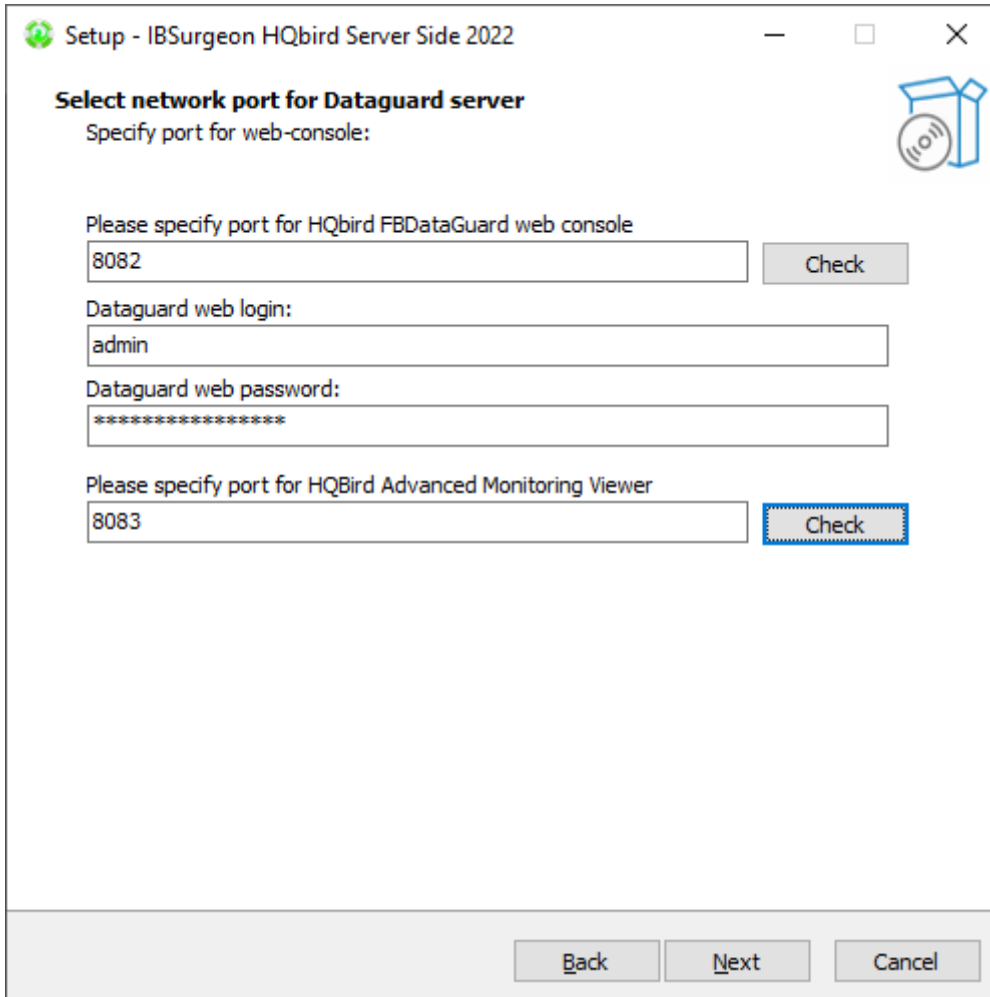
Figure 8. Select components from HQbird Server Side to be installed

We recommend that you install HQbird Enterprise, which contains all HQbird components and

Firebird, to avoid further configuration. All HQbird modules are installed in the inactive mode and do not affect the operation of the Firebird server until they are configured or used.

If you select to install HQbird Enterprise (i.e., with Firebird), it will install Firebird in the subfolder of HQbird installation, by default `C:\HQBird\Firebird25` or `C:\HQBird\Firebird30` or `C:\HQBird\Firebird40`.

Then, you will be asked to specify the port for HQbird FBDataGuard (web interface to manage HQbird):



Setup - IBSurgeon HQbird Server Side 2022

Select network port for Dataguard server
Specify port for web-console:

Please specify port for HQbird FBDataGuard web console
8082 Check

Dataguard web login:
admin

Dataguard web password:

Please specify port for HQBird Advanced Monitoring Viewer
8083 Check

Back Next Cancel

Figure 9. Specify port, login and password for HQbird FBDataGuard and HQBird Advanced Monitoring Viewer

We recommend to keep 8082, but sometimes this port can be occupied, so you can change it.

Default password: **strong password**

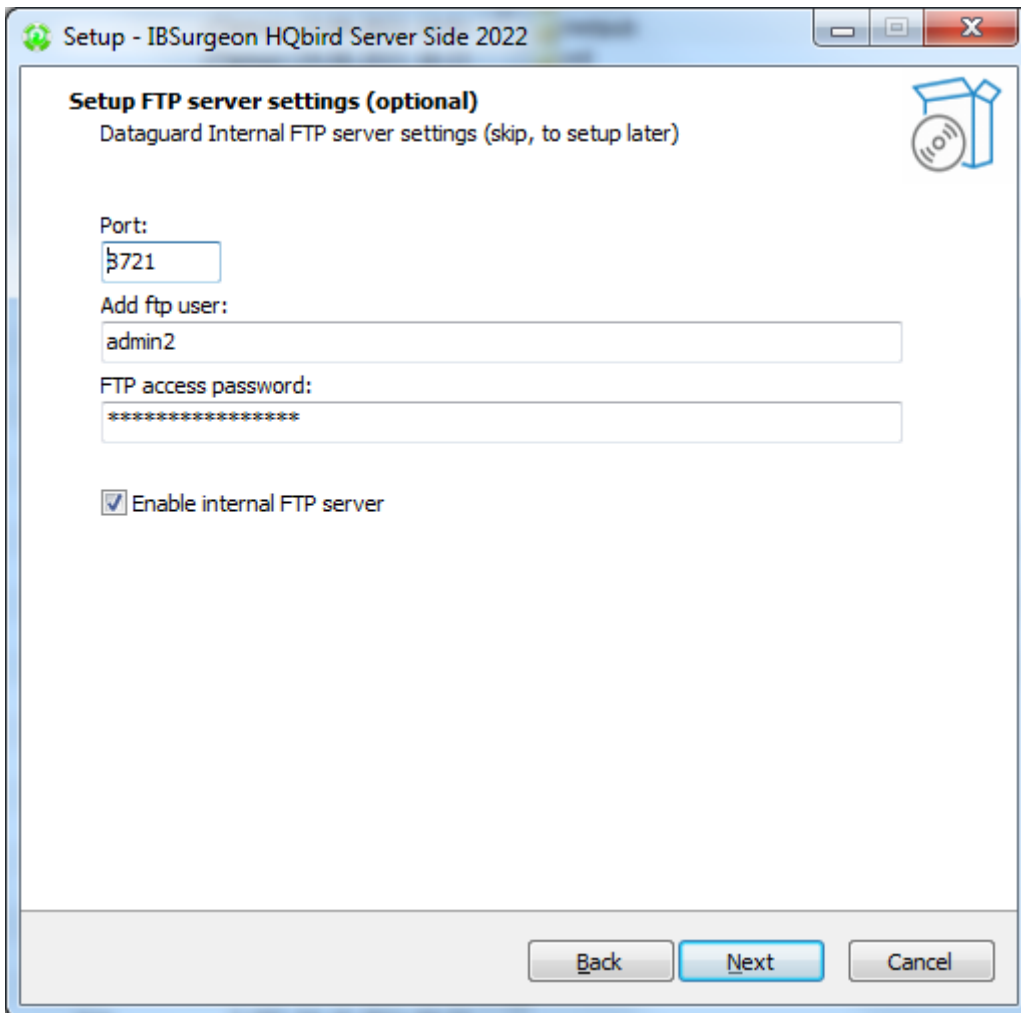
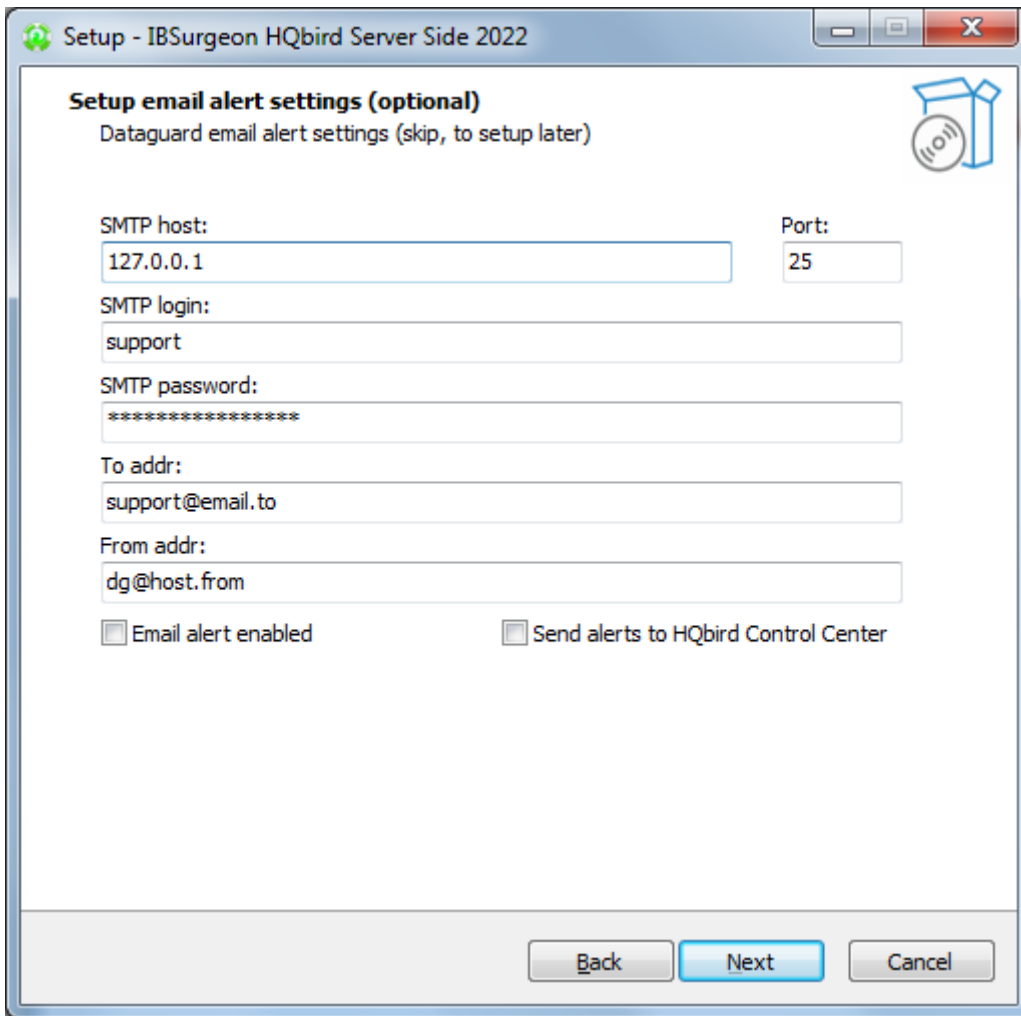


Figure 10. Setup FTP Server settings

After that, the installer will ask about email settings to be used to send email alerts:



The screenshot shows a Windows-style window titled "Setup - IBSurgeon HQbird Server Side 2022". The main heading is "Setup email alert settings (optional)" with a sub-heading "Dataguard email alert settings (skip, to setup later)". There is an information icon in the top right corner. The form contains the following fields and options:

- SMTP host: 127.0.0.1
- Port: 25
- SMTP login: support
- SMTP password: *****
- To addr: support@email.to
- From addr: dg@host.from
- Email alert enabled
- Send alerts to HQbird Control Center

At the bottom of the window are three buttons: "Back", "Next", and "Cancel".

Figure 11. Email alerts settings



You can skip this step: all email alerts can be set later in web interface.

Then, the installation wizard will ask to specify the port for Firebird engine installed with HQbird:

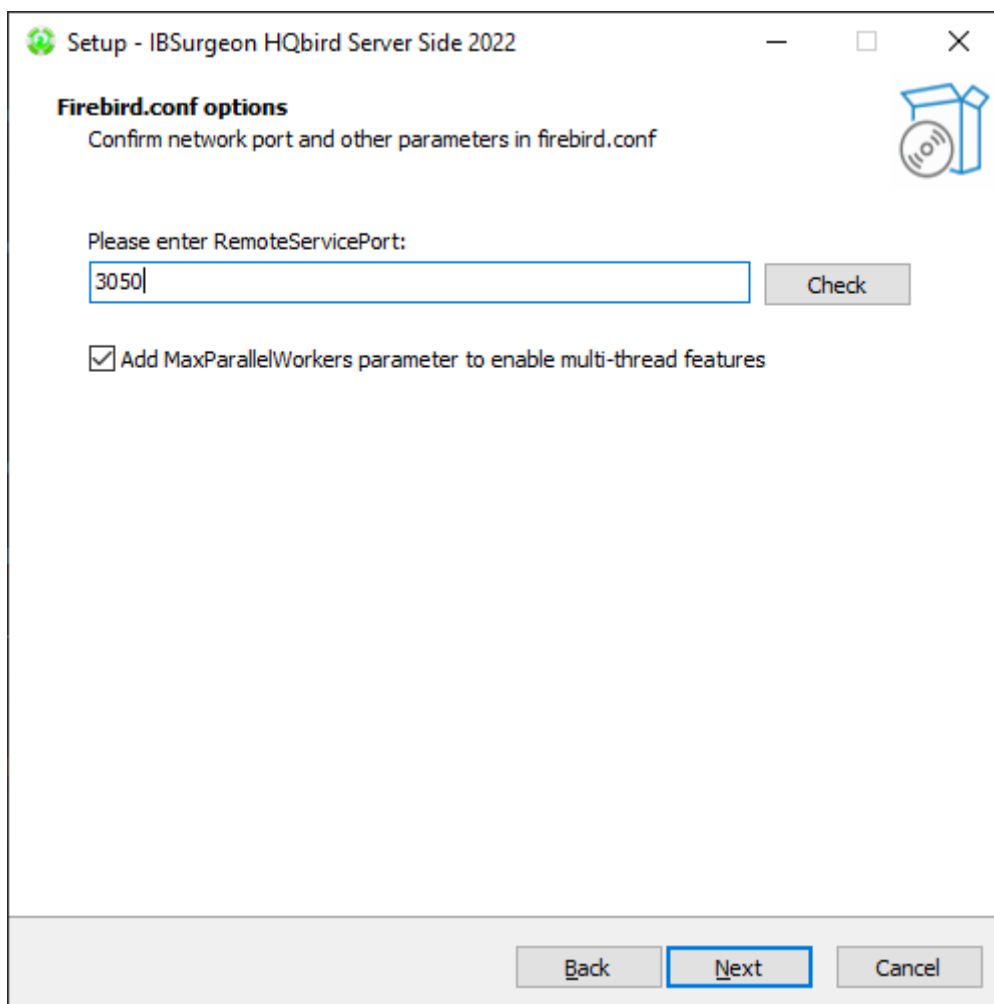


Figure 12. Specify port for Firebird server

By default, the port is 3050. If the port will be occupied by another running Firebird, the installation wizard will warn you and make to choose another port. Or, you can stop and uninstall another Firebird service.

The checkbox “Add MaxParallelWorkers parameter to enable multi-threaded features” enables multi-threading support for backup, restore and sweep. You can learn more in the “Performance enhancements” chapter of the "[Multi-thread sweep, backup, restore](#)" section.

Then, only if you are installing HQbird Standard (i.e., without bundled Firebird), the installation wizard will ask you to specify the folder where Firebird is installed:

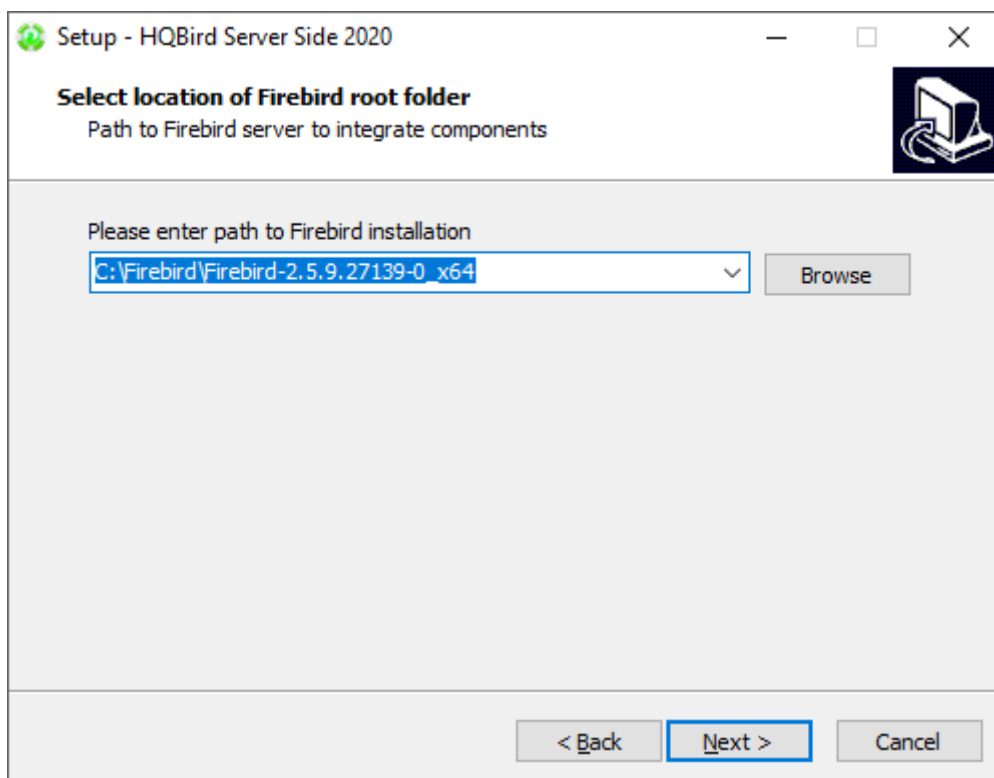


Figure 13. Confirm the location of the current Firebird instance (for HQbirdStandard).

Attention! During this step, the installation wizard checks the availability and compatibility of the installed Firebird version with HQbird. If the specified folder does not contain a correctly installed Firebird version, you will see the following warning:

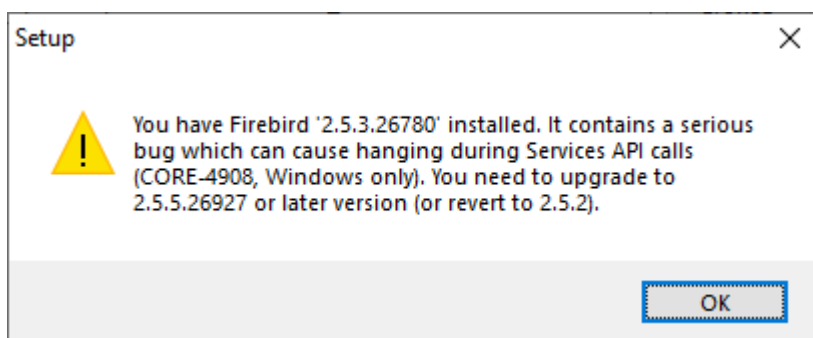


Figure 14. This warning from the installation wizard prompts you to select the correct Firebird folder.

You should use Firebird version 2.5.5 or higher for HQbird Standard to be installed (see [How to Update Firebird for Windows](#)), or choose HQbird Enterprise to install the newest Firebird.

Then, you can specify the folder name and location in Windows menu:

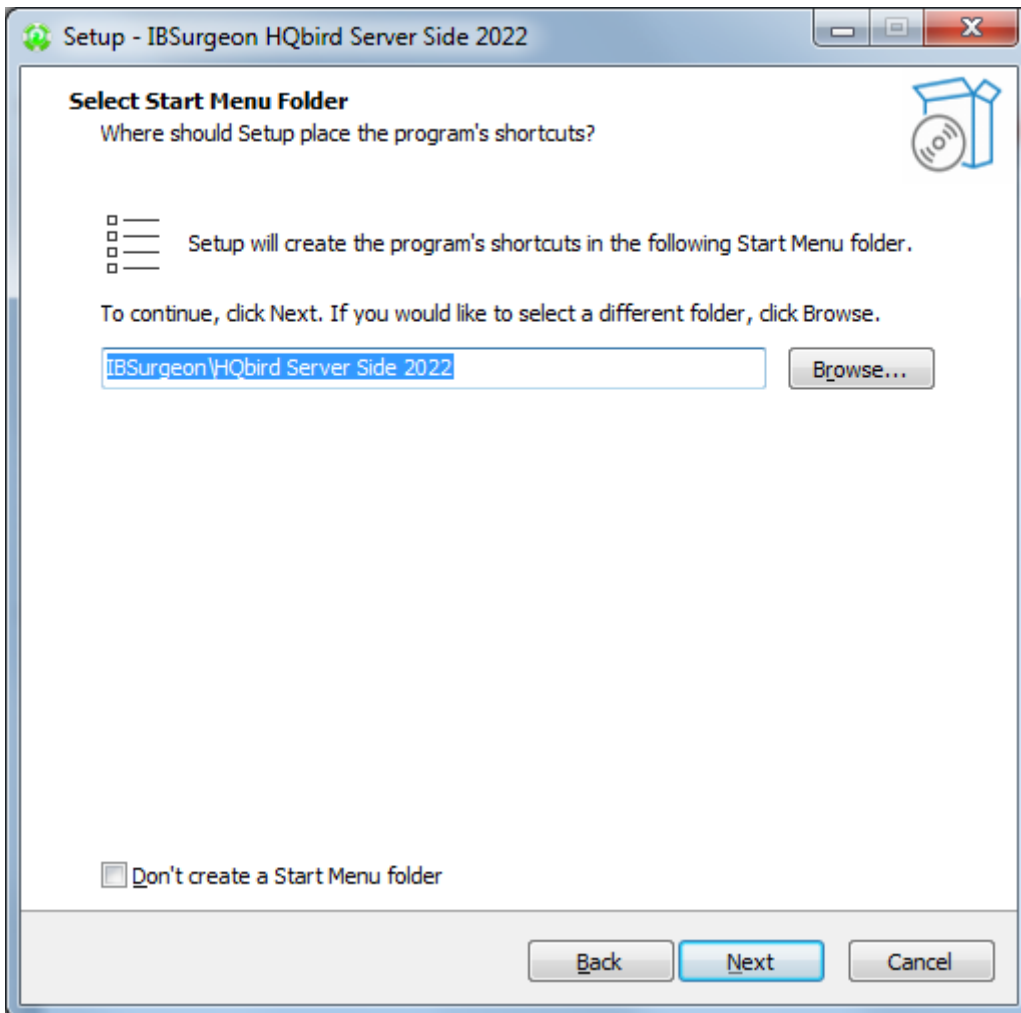


Figure 15. Choose Windows Start menu folder.

At the next step the installer will offer you to pre-configure HQbird to be used as master or replica server:

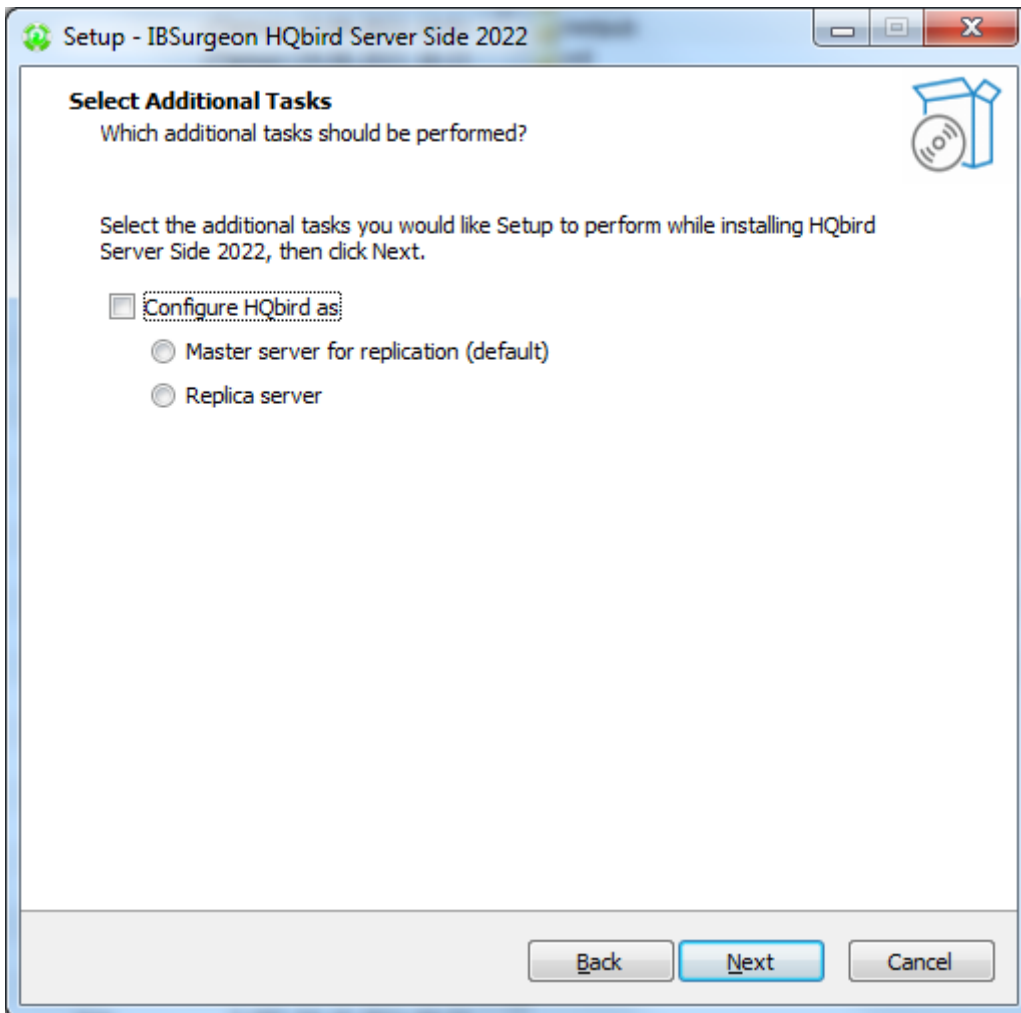


Figure 16. Pre-configuration for replication.

You can skip this step, this configuration can be done later.

The final step is a summary of components to be installed and their paths:

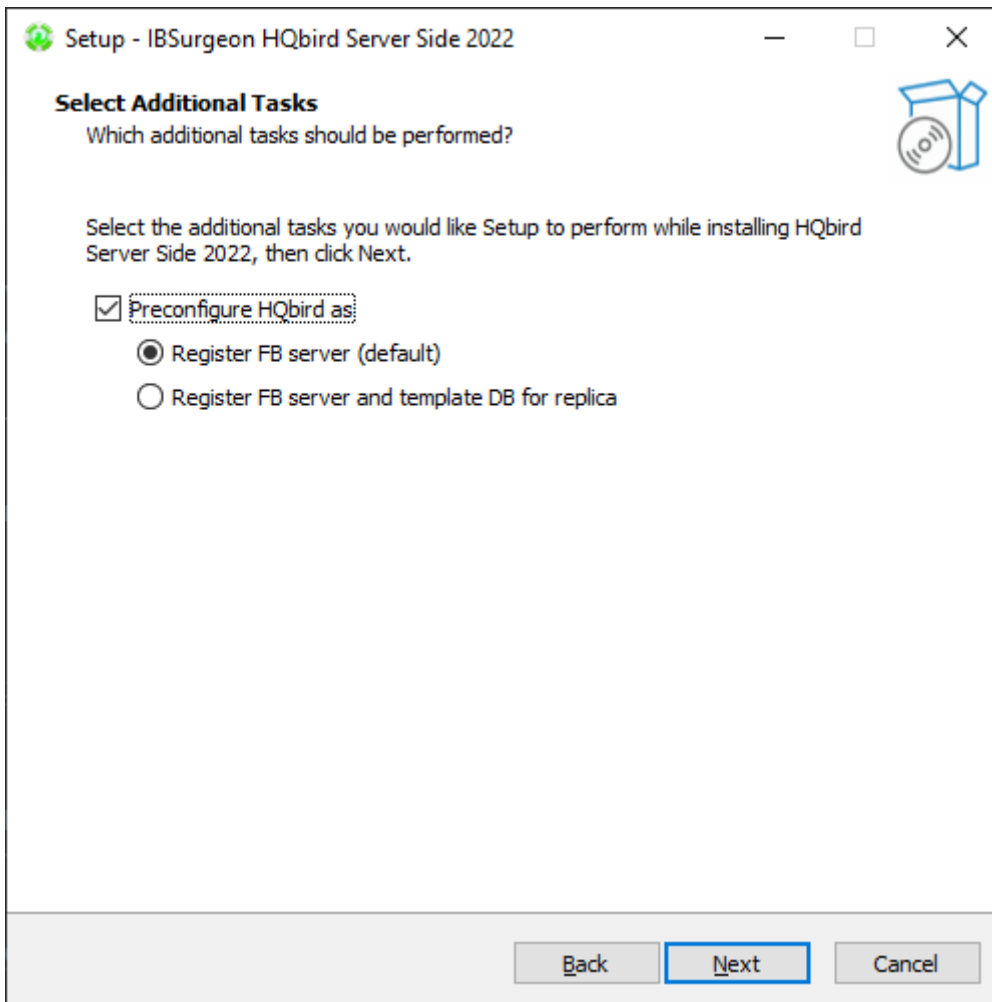


Figure 17. Click Install to complete the installation.

After that you have to activate HQbird (see [How to Activate HQbird](#)) and proceed to configure the HQbird components.

At the end of installation process, you will be asked about next steps:

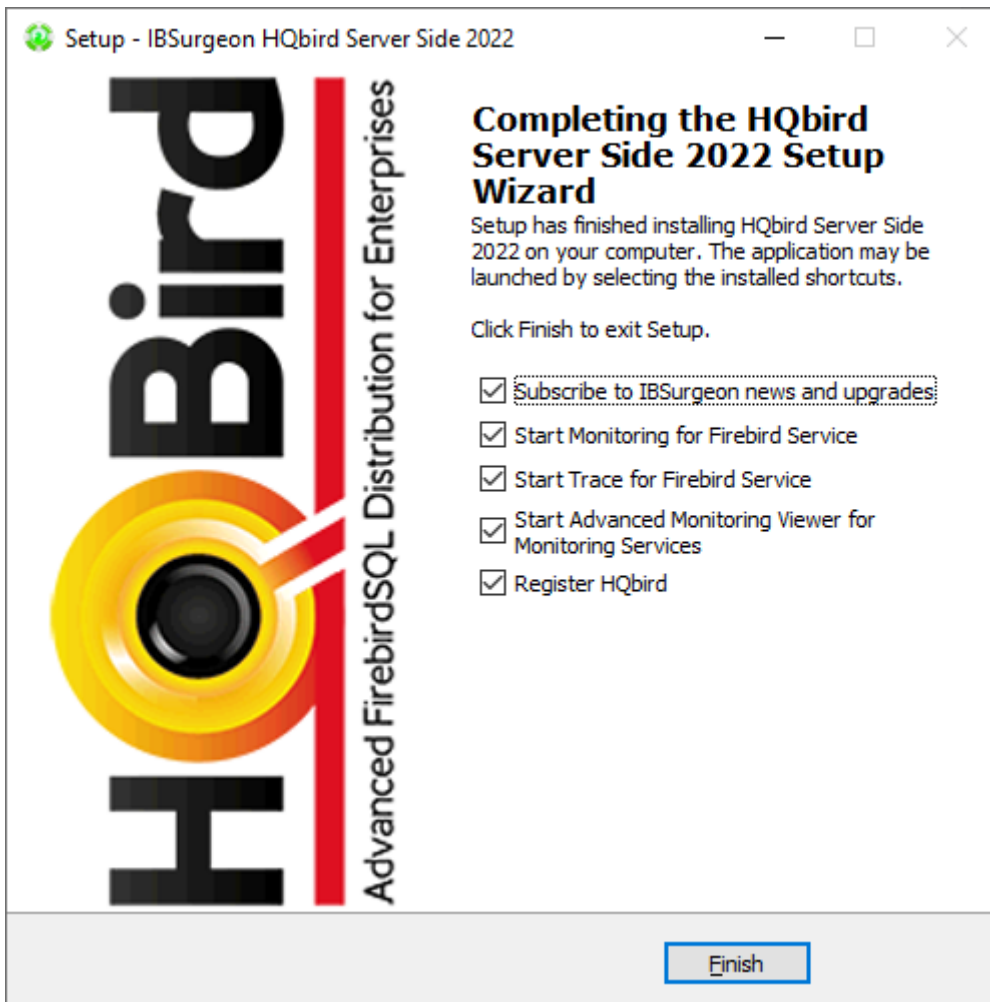


Figure 18. Post-installation steps.

2.3. Installing HQbird Administrator on Windows

To install HQBird Administrator, download the distribution package from <https://ib-aid.com/en/hqbird/>, or from your account at <http://deploy.ib-aid.com>.

The name of HQbird Administrator package is *HQbirdAdminNNNN.exe* (it is in the zip archive).

Run the installation wizard and follow the standard installation steps: digital signature check, license, then select the installation folder:

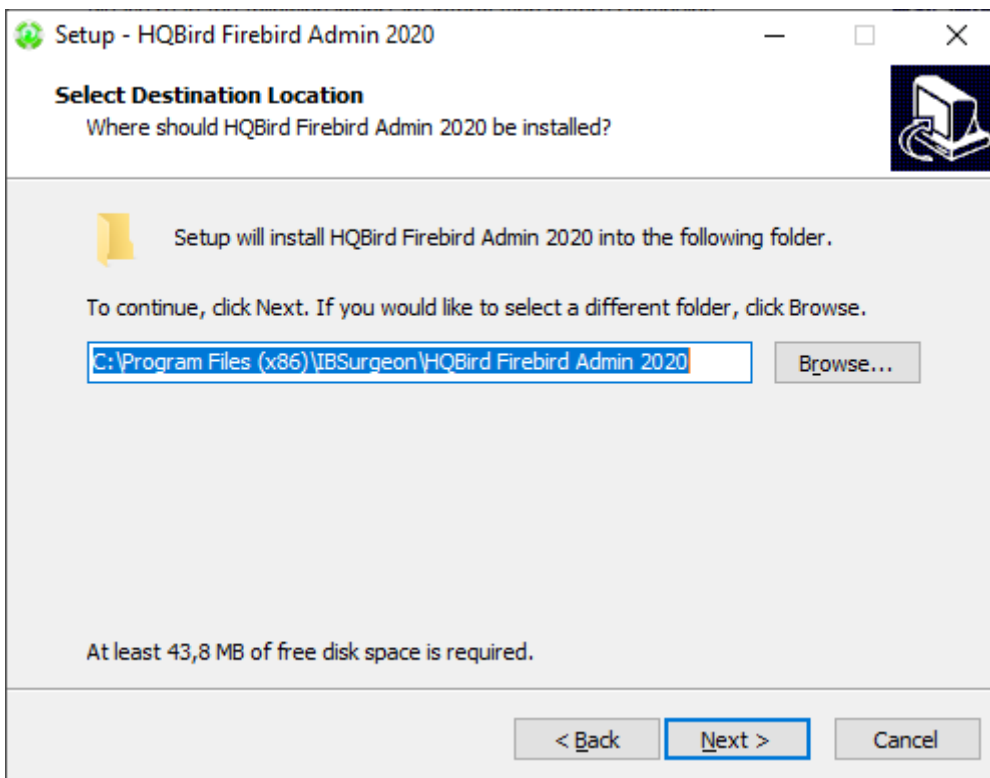


Figure 19. Select where to install HQbird Admin.

Select tools to install after that. We recommend that you install all tools.

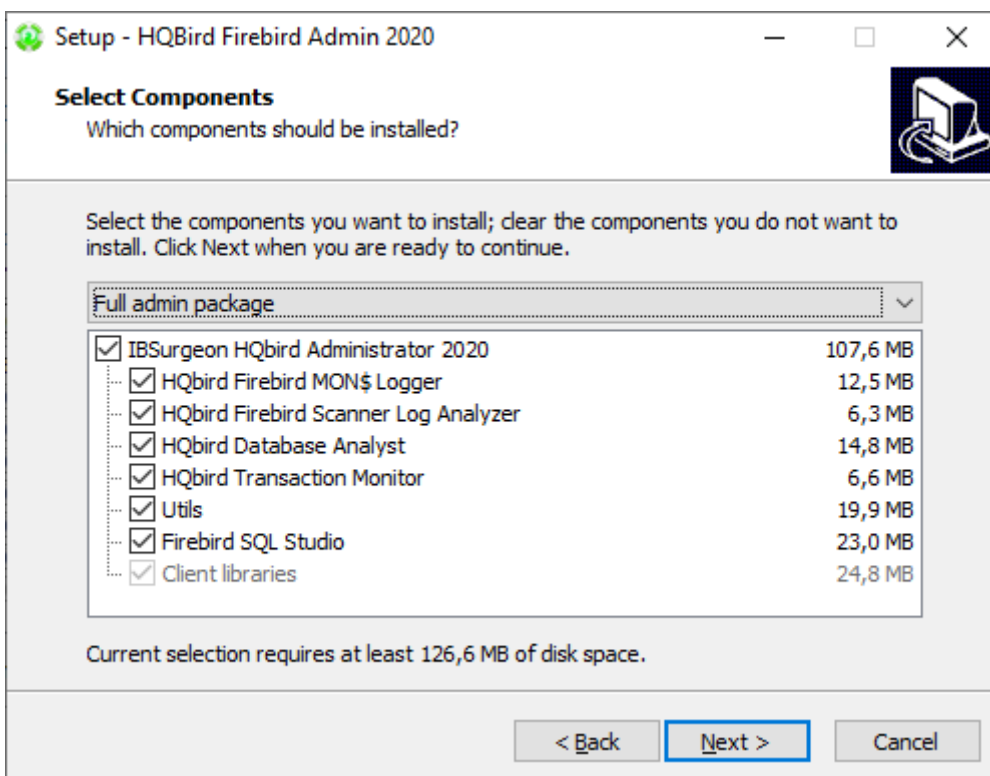


Figure 20. Select tools to install.

Follow the instructions after that. After the installation is over, you will be offered to launch the activation wizard. If you are installing HQbird Admin on the same computer where HQbird Server was already installed, the license will be automatically detected by HQbird Admin tools.

2.3.1. How to install community version of Firebird on Windows

The easiest way is to install Firebird bundled with HQbird – just choose the desired version during the installation. However, sometimes it is necessary to use HQbird with a community version of Firebird.



Please note – to enable replication and performance features in HQbird Enterprise you need to install Firebird bundled with HQbird ServerSide.

To install Firebird separately, download the Firebird zip archive from www.firebirdsql.org

Unpack the archive file to a suitable location (for instance, `C:\Firebird25`), after that copy the optimized configuration file `firebird.conf` (see [Optimized Configurations](#) below) to this folder.

Then, go to the Bin folder and then use the **Run As Administrator** option to run the batch file with the architecture you need.

- For Firebird 2.5 – run `install-superclassic.bat`.
- For Firebird 3.0 – set parameter `ServerMode=Super` and run `install_service.bat`.

Of course, you can choose the SuperServer for 2.5 or Classic architecture for 3.0 if you know what you need.

As a result of running the command file, Firebird of the selected architecture will be installed and run as a service.

You can make sure the Firebird service is installed and running in the **Services** snap-in (run `services.msc` in command prompt):

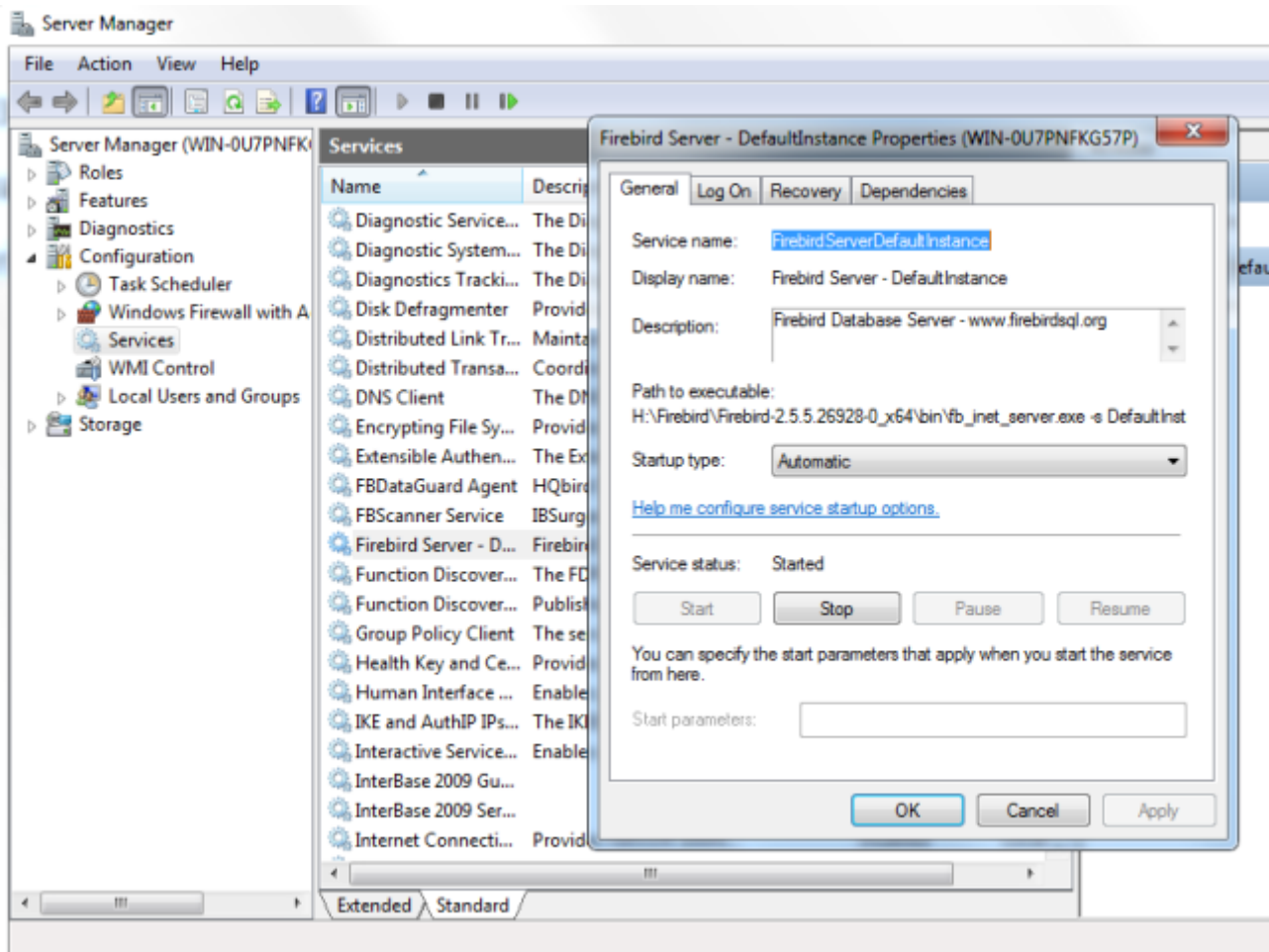


Figure 21. Firebird Service.

In this example, Firebird is installed in the folder `H:\Firebird\Firebird-2.5.5.26928-0_x64` and running as a service with the SuperClassic architecture.

2.4. Installing HQbird Server on Linux

To install HQbird Server Side on Linux, you need to download HQbird ServerSide for Linux with integrated Firebird from [this location](https://ib-aid.com/en/hqbird-installation/): <https://ib-aid.com/en/hqbird-installation/>

This archive contains 3 files:

- `install_fb25_hqbird2022.sh`
- `install_fb30_hqbird2022.sh`
- `install_fb40_hqbird2022.sh`

You must be root or sudoer to install HQbird on Linux!

General prerequisites: install **java version 1.8** before installing HQbird! We recommend OpenJDK, but Oracle's Java is also fine.

2.4.1. Installation of HQbird with Firebird 2.5 on Linux

1. Uninstall all previously installed Firebird versions before running this installer. Make sure you don't have Firebird installed from repositories!

2. Apply execution rights to the installation package:

```
chmod +x install_fb25_hqbird2022.sh
```

3. Run installation script *install_fb25_hqbird2022.sh*. It will install Firebird into */opt/firebird* and HQbird into */opt/hqbird*
4. By default, Firebird 2.5 is installed as Classic. We recommend to install it as SuperClassic – for this run script */opt/firebird/bin/changeMultiConnectMode.sh* and choose **thread**

Next steps:

1. Please note that Firebird 2.5 will be installed with SYSDBA/masterkey
2. You can stop/start Firebird 2.5 with command `service firebird stop` or `service firebird start`. Check is it running with command `ps aux | grep firebird`
3. You can stop/start HQbird with command `service hqbird stop` or `service hqbird start`. Check is it running with command `ps aux | grep dataguard`
4. Run browser, and log in to HQbird FBDataGuard <http://serverurl:8082>, with user/password = **admin/strong password**
5. Choose “I have HQbird Enterprise” and register HQbird with the email and password you have received from IBSurgeon Deploy Center.
6. If necessary, follow steps to setup — or see the appropriate chapter of this Guide.

2.4.2. Installation of HQbird with Firebird 3.0 on Linux

Prerequisites: make sure you have **libtommath**, **libncurses5-dev** and **ICU** installed (there will be an appropriate error message if they are not installed).

1. Uninstall all previously installed Firebird versions before running this installer
2. Apply execution rights to the installation package:

```
chmod +x install_fb30_hqbird2022.sh
```

3. Run installation script *install_fb30_hqbird2022.sh*. It will install Firebird into */opt/firebird* and HQbird into */opt/hqbird*
4. By default, Firebird 3.0 is installed as SuperServer. Keep it.
5. Firebird 3.0 will be installed with SYSDBA/masterkey

Next steps:

1. You can stop/start Firebird 3.0 with command `service firebird-superserver stop` or `service firebird-superserver start`. Check is it running with command `ps aux | grep firebird`
2. You can stop/start HQbird with command `service hqbird stop` or `service hqbird start`. Check is it running with command `ps aux | grep dataguard`

3. Run browser, and log in to HQbird FBDataGuard <http://serverurl:8082>, with user/password = **admin/strong password**
4. Choose “I have HQbird Enterprise” and register HQbird with the email and password you have received from IBSurgeon Deploy Center.
5. If necessary, follow steps to setup — or see the appropriate chapter of this Guide.

2.4.3. Installation of HQbird with Firebird 4.0 on Linux

Prerequisites: make sure you have **libtommath** and **ICU** installed (there will be an appropriate error message if they are not installed).

1. Uninstall all previously installed Firebird versions before running this installer
2. Apply execution rights to the installation package:

```
chmod +x install_fb40_hqbird2022.sh
```

3. Run installation script *install_fb40_hqbird2022.sh*. It will install Firebird into */opt/firebird* and HQbird into */opt/hqbird*
4. By default, Firebird 4.0 is installed as SuperServer. Keep it.
5. Firebird 4.0 will be installed with SYSDBA/masterkey

Next steps:

1. You can stop/start Firebird 4.0 with command `service firebird-superserver stop` or `service firebird-superserver start`. Check is it running with command `ps aux | grep firebird`
2. You can stop/start HQbird with command `service hqbird stop` or `service hqbird start`. Check is it running with command `ps aux | grep dataguard`
3. Run browser, and log in to HQbird FBDataGuard <http://serverurl:8082>, with user/password = **admin/strong password**
4. Choose “I have HQbird Enterprise” and register HQbird with the email and password you have received from IBSurgeon Deploy Center.
5. If necessary, follow steps to setup — or see the appropriate chapter of this Guide.

2.4.4. Installation of HQbird Standard on Linux

If you have a license of HQbird Standard, or if you don't want to change the existing Firebird installation, please run the following command:

```
install_fb4_hqbird2022.sh --nofirebird
```

It will install HQbird without Firebird binaries.



Please note, that advanced features (replication, multi-thread support, encryption, authentication) require HQbird Enterprise with Firebird binaries!

2.4.5. Firewall settings

Firebird is installed on port **3050**, HQbird web interface is listening on port **8082**, and licensing interface is listening on **8765**.

These ports can be changed in `/opt/firebird/firebird.conf` (RemoteServicePort), `/opt/hqbird/conf/network.properties` (server.port) and `/opt/hqbird/conf/license.properties` (serverlicense.port).

Make sure to allow these ports in your firewall configuration.

Attention!



After upgrade, make sure that there is only the one copy of HQbird is running! If there are 2 copies, stop them (`service hqbird stop` for the first and `kill <process-number>` for the second instances) and start it again.

2.5. Upgrade existing HQbird version

HQbird installer on Windows (from v 2018R2) and on Linux (from v 2018R3) supports automatic upgrade of the configuration of already installed HQbird version 2017R2 and later.

If HQbird installer will notice the previous version of HQbird, it will ask you to confirm the upgrade, and in case of the positive answer, it will stop Firebird, HQbird and upgrade their files.

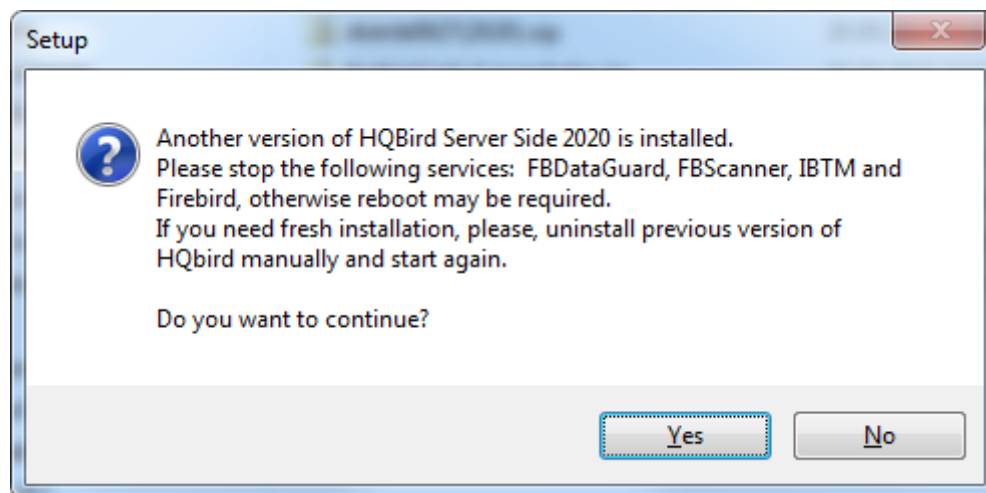


Figure 22. Warning about upgrade.

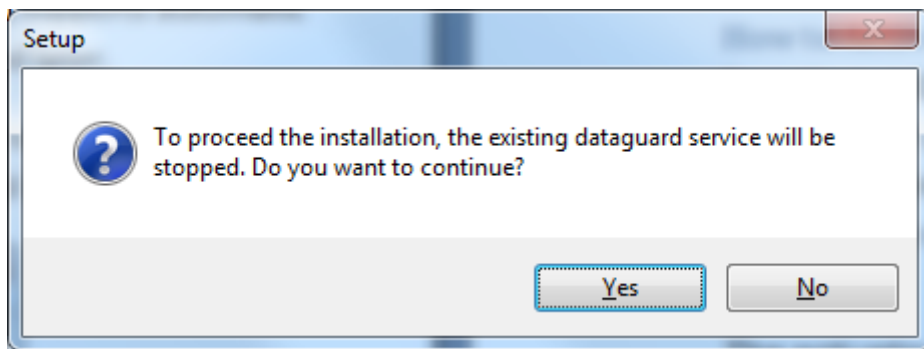


Figure 23. Warning about restart of currently running HQbird FBDDataGuard.

The configuration will be retained—it means that *firebird.conf*, *aliases.conf*, *securityX.fdb*, and HQbird configuration files will not be deleted (HQbird configuration files will be upgraded to the new configuration version).

The upgrade does not change the Windows service settings for Firebird and HQbird – it means that if you have changed “Run As” properties of the service, they will be retained.



After upgrade on Linux Firebird and HQbird must be started manually!

After upgrading HQbird, open the web-console and choose in the right upper corner: “Refresh HQbird web-console”. It is necessary to clean the cache of JavaScript part of the application.



Please note—if you are installing HQbird 2022 over the old version of HQbird on Windows, the dialog with installation options will be shown as disabled, because we cannot automatically upgrade from 2.5 to 3.0 or 4.0, and installer can only upgrade the same components. If you need a different installation, remove old version of HQbird from the computer prior installing 2022.

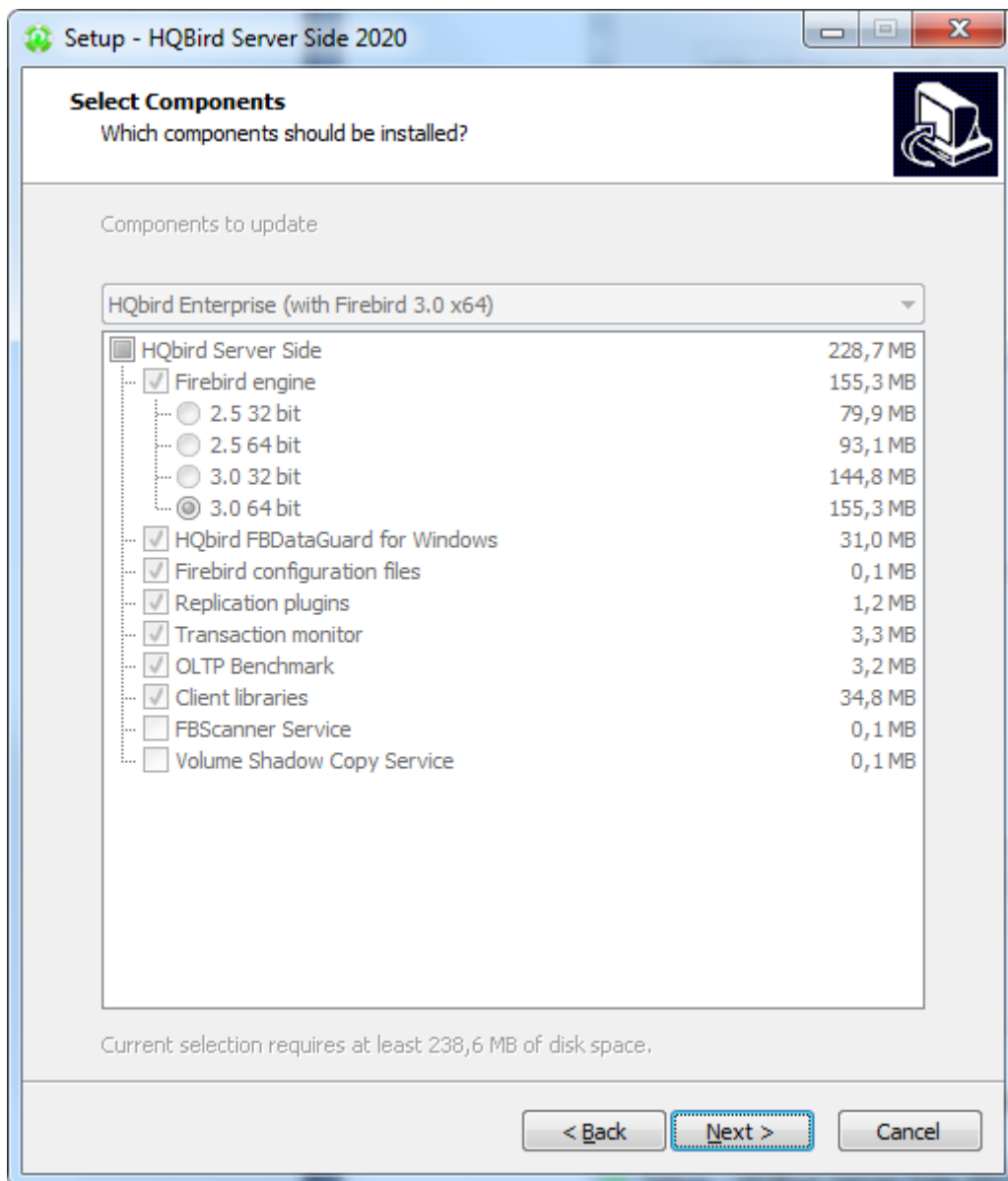


Figure 24. An example of the disabled components selection dialog in case of upgrade.

2.6. Registration of HQbird

2.6.1. How to activate HQbird

To activate HQbird, you can either use a separate utility included in the server and administrator packages for Windows, or use the registration mechanism embedded into the HQBird Firebird DataGuard web interface (for Windows and Linux), or run any tool from the administrator software and use the built-in activation wizard.

The activation wizard looks and works the same in the tools and in the activation tool. It is enough to perform activation once on any computer that can connect to the server where HQbird ServerSide is installed.

You can launch the registration utility from the **Start** menu (IBSurgeon\HQbird Firebird Admin\HQbird):

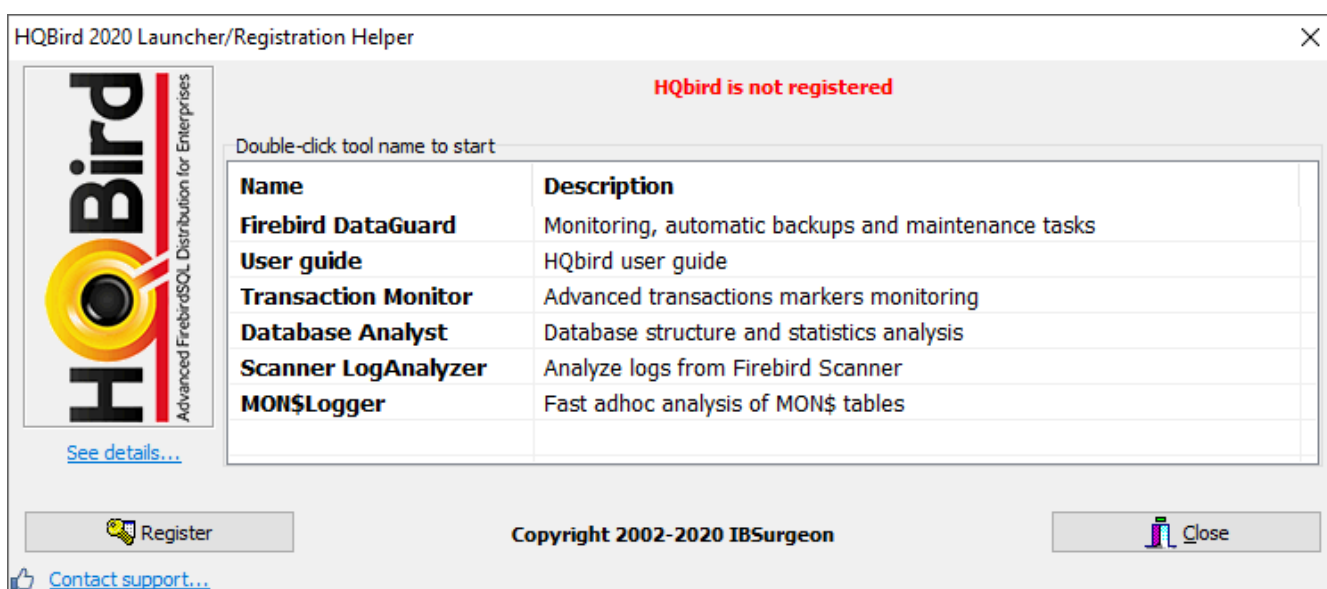


Figure 25. HQBird registration helper.

If you click the **Register** button (or Re-Register for repeated registration), you will see the activation wizard:

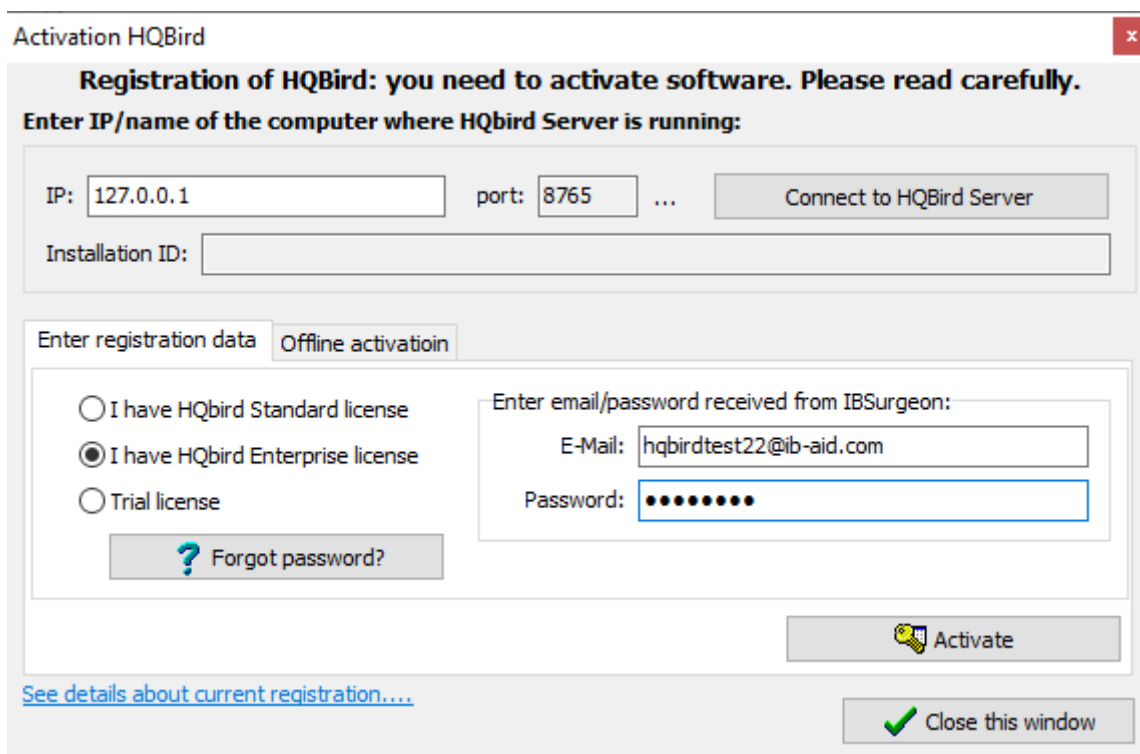


Figure 26. HQBird activation window.

After that, specify the **IP address** or the **computer name** of the server HQbird is installed on in the upper input field and click **Connect to HQbird Server**. If you started registration utility on the same computer with HQbird Server, it will be “localhost”, otherwise — some remote address.

Then enter your registration data. If you have a license, enter your e-mail address and password that you used to register with the IBSurgeon Deploy Center and click **Activate**.



If you have no license, choose Trial license, specify your e-mail address and click **Activate**. You will be automatically registered and the password will be sent to your e-mail address.

Right after you click **Activate**, the registration wizard will try to connect to the IBSurgeon Deploy Center () and obtain a license. If it succeeds, you will see the corresponding message. If there are any problems, you will see the error message.

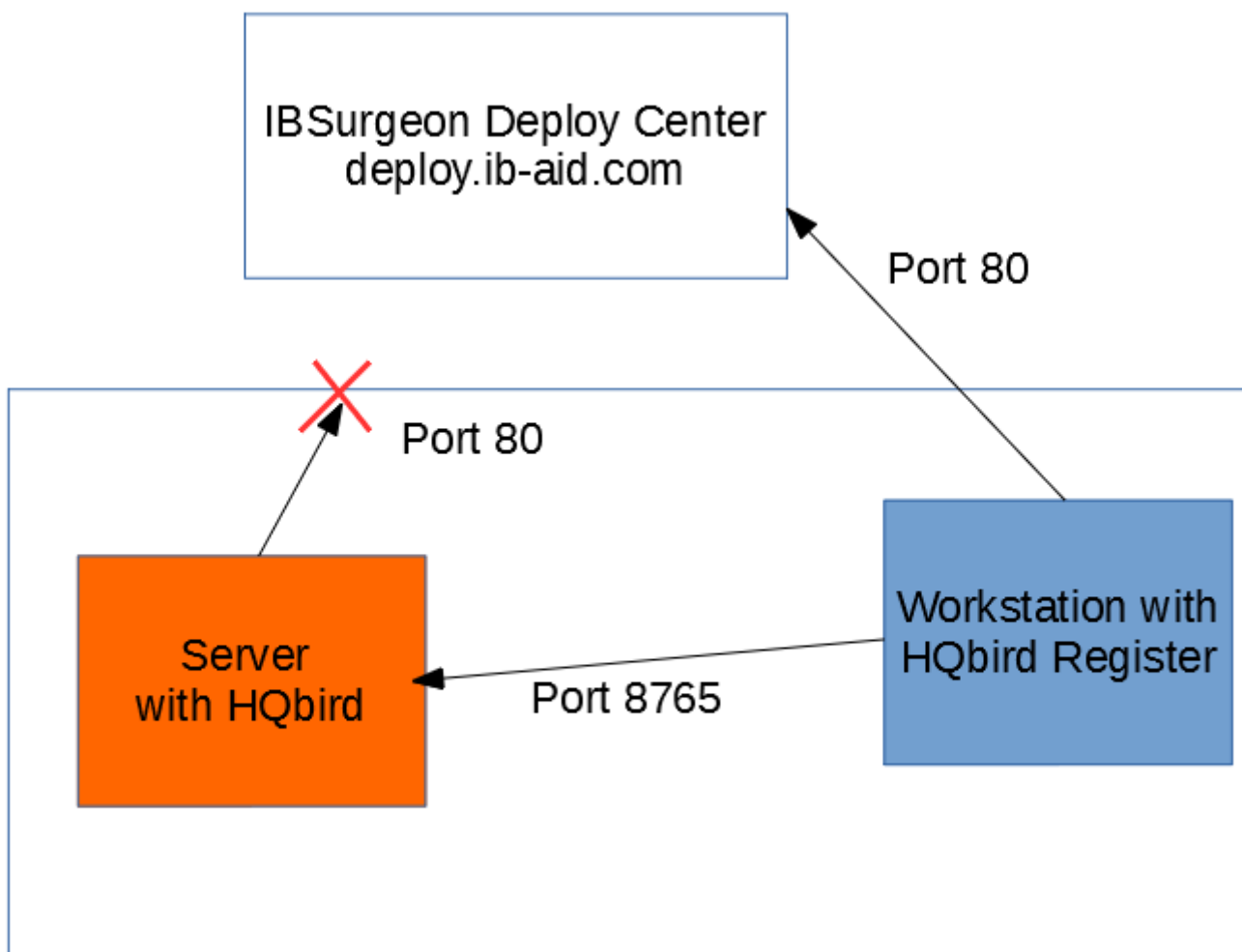
If you forget the password, click the **Forgot password...** button and it will open the browser with the password recovery form.

If you need to purchase a new or additional license or renew your subscription, click **Purchase**.

Click **Close this window** after the registration is over.

Internet Activation via a Client Computer

If the server with HQbird Server does not have access to the Internet, you can still activate it via the Internet: you can install HQbird Administrator on any client computer with Windows that has both access to the Internet and access to the HQbird Server and perform activation.



Run HQbird Register tool and enter there: IP address of your server (or, server name—for example, mylinuxserver), email and license, and click Activate:

Activation HQBird

Registration of HQBird: you need to activate software. Please read carefully.

Enter IP/name of the computer where HQbird Server is running:

IP: port: ...

Installation ID:

Enter registration data **Offline activation**

I have HQbird Standard license
 I have HQbird Enterprise license
 Trial license

Enter email/password received from IBSurgeon:

E-Mail:

Password:

[See details about current registration....](#)

Figure 27. HQBird activation window.

2.6.2. Offline Activation

If the server and all client computers have no access to the Internet, you should use offline activation. To do it, click Offline activation tab and follow instructions there. In case of any troubles please contact.

2.6.3. Activation in web interface

127.0.0.1:8082/static/dashboard.html

FBDataGuard 10.0.1003: please wait and refresh page later...

HQbird 10.0.1003 is not registered

✘ There are no valid unlock file found!

Get new/existing license (You can order full license at ib-aid.com)

Type:

I have HQbird Standard license

I have HQbird Enterprise license

I need trial license

Off-line registration

E-mail:

Password:

Show password

[I forgot my password.](#)

Activate

Figure 28. Activation in web interface.

Chapter 3. Configuration of HQbird

3.1. Initial configuration of HQbird FBDataGuard (backups, monitoring, alerts,etc)

Please follow these steps:

1. Make sure that you have Firebird 2.5.5 or later (Firebird 2.1 is also supported with some limitation), and it is working;
2. HQbird FBDataGuard service is installed and running
 - a. You can check it using Services applet in Control Panel (right-click on “My Computer”, choose “Manage”, then “Services and Applications”, “Services” and find in the list “FBDataGuard Agent”
 - b. At Linux you can check it with command `ps aux | grep dataguard`.
3. Make sure the FBDataGuard port is accessible (8082) and it is not blocked by firewall or any other antivirus tools. If necessary, adjust port in FBDataGuard configuration file (see [Web-console port](#)).

3.1.1. Launch web-console

To open web-console type in your browser or use IP address of your server with installed HQbird ServerSide.

Or you can choose in “Start” menu IBSurgeon\HQbird Server Side\Firebird DataGuard\“Launch the DataGuard web console for localhost”.

Supported browsers

FBDataGuard web interface works correctly with Firefox, Chrome, Safari and Internet Explorer 11.

Error message regarding webs-site certificate

If you have configured FBDataGuard to use https, the browser will indicate that this web-site () is not safe, and it will recommend leaving web-site. This message is caused by the default security certificate for FBDataGuard web-console.

Please ignore this message and click to open FBDataGuard web-console.

It will ask you for username and password (login dialog can be different for Firefox, Safari or Chrome).

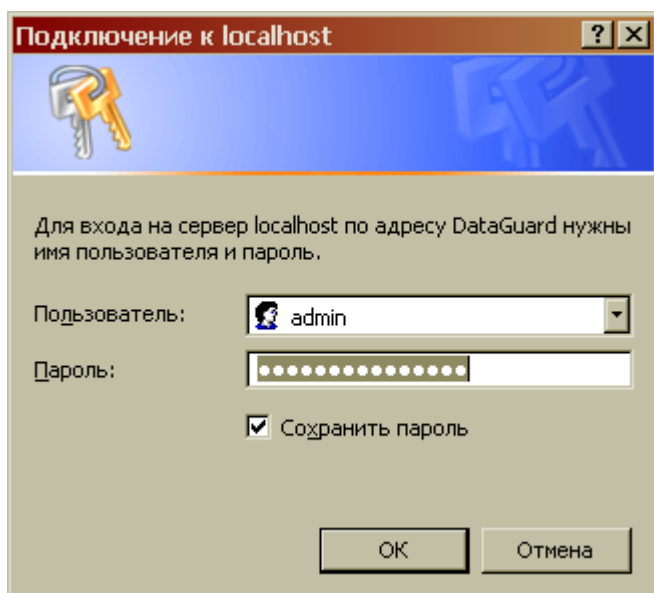


Figure 29. Enter username and password for FBDataGuard web-console.



Attention!

Default username/password for HQbird FBDataGuard is **"admin"/"strong password"** (without quotes).

3.1.2. Auto discovery feature of FBDataGuard

At the first launch FBDataGuard will check computer for installed Firebird servers. FBDataGuard for Windows search registry for Firebird records, FBDataGuard for Linux checks default locations of Firebird installations.

FBDataGuard will show the list of all Firebird copies installed, but only the one instance of Firebird can be monitored by FBDataGuard. Choose it by clicking **[Add Firebird engine to monitoring >>]**

If you don't see Firebird instance in auto discovery list, you can choose **[Add custom >>]** and configure instance parameters manually.

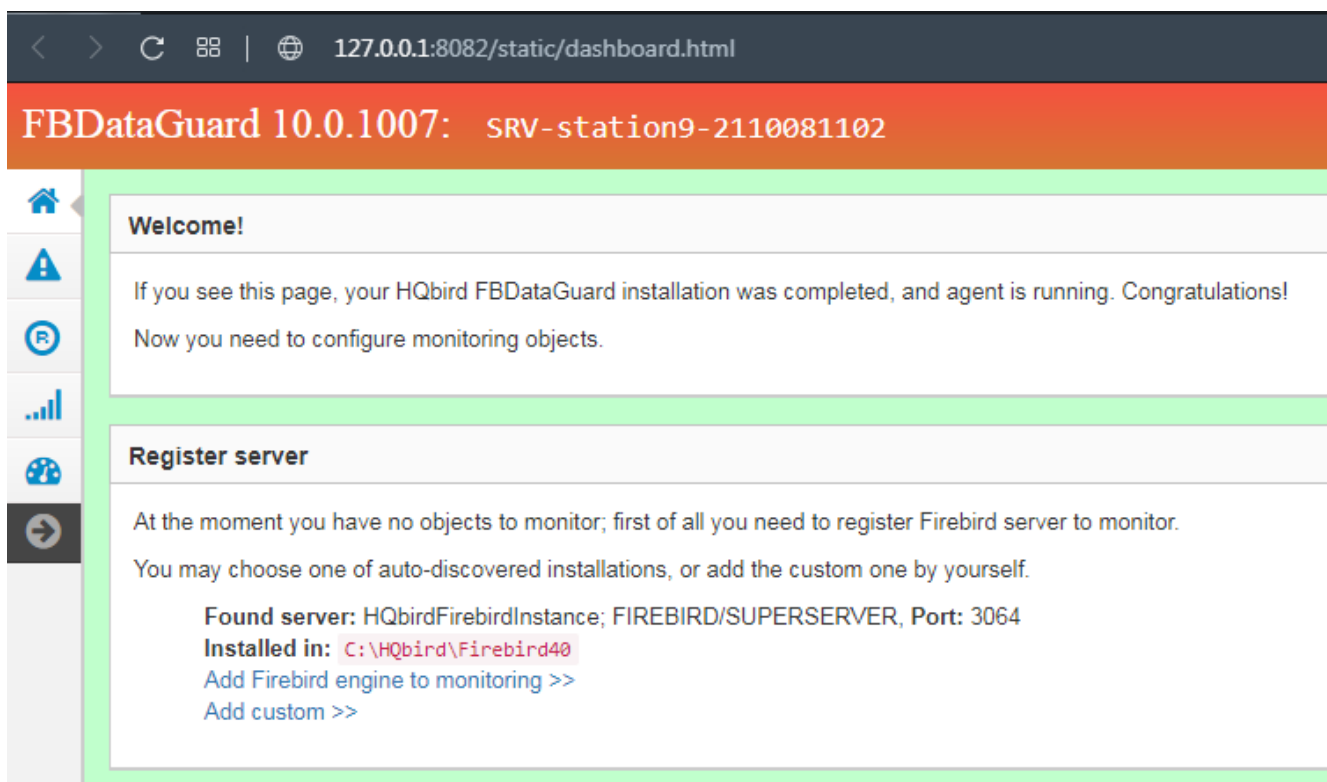


Figure 30. Auto discovery feature of HQbird.

3.1.3. Firebird server registration

To register auto-discovered server, you need to click at [**Add Firebird engine to monitoring>>**] and then adjust auto-discovered settings.



Note: to use Windows Trusted Authentication (by default it's off), you need to be sure that libraries *jaybird30.dll* and *fbclient.dll* (from appropriate Firebird version) are in searchable Windows paths.

When installing under Windows, if the option to automatically register the master/replica is selected, the server will be added automatically. In this case, you can skip this step. If the option to automatically register a replica is selected, then the database will be added in addition.

Let's consider what can you see in the Server dialog (and, normally, you don't need to change them):

- **Installed in:** Firebird installation folder
- **Binary folder:** Firebird bin folder (for Firebird 3 on Windows Binary folder is the same as the installation folder)
- **Log:** location of *firebird.log*
- **Configuration file:** location of *firebird.conf*
- **Aliases:** location of *aliases.conf* or, for Firebird 3, *databases.conf* (**please change it manually, if needed**)
- **Host:** name of the server, usually localhost
- **Port:** network port for Firebird, according *firebird.conf* settings

- **Use trusted auth:** use trusted authentication, by default it is off
- **SYSDBA login:** name of SYSDBA user, usually it is SYSDBA
- **SYSDBA password:** password for SYSDBA
- **Output directory:** Folder where backups, statistics and gathered logs will be stored

Register Firebird server

Installed in: C:\HQbird\Firebird30

Binary folder: C:\HQbird\Firebird30

Log: \$\{server.installation}\firebird.log

Configuration file: \$\{server.installation}\firebird.conf

Aliases: \$\{server.installation}\databases.conf

Host: localhost

Port: 3050

Use trusted auth:

SYSDBA login: SYSDBA

SYSDBA password:

Output folder: \$\{agent.default-directory}\\$\{server.id}

Cancel Save

Figure 31. Register server in HQbird FBDataGuard.

By default “Output directory” for Firebird server is $\{\text{agent.default-directory}\}/\{\text{server.id}\}$, it corresponds to $C:\text{HQbirdData}$ in case of a default installation.

It can be not very convenient, so we recommend pointing FBDataGuard output directory to more simple path, usually located at disk where backups are intended to be stored, for example $F:\text{myserverdata}$.

After clicking “Register” FBDataGuard will populate default configurations files and immediately start analysis of *firebird.log*. It can take a while (for example, 1 minute for 100Mb *firebird.log*). After that you will see initial web-console with registered Firebird server:

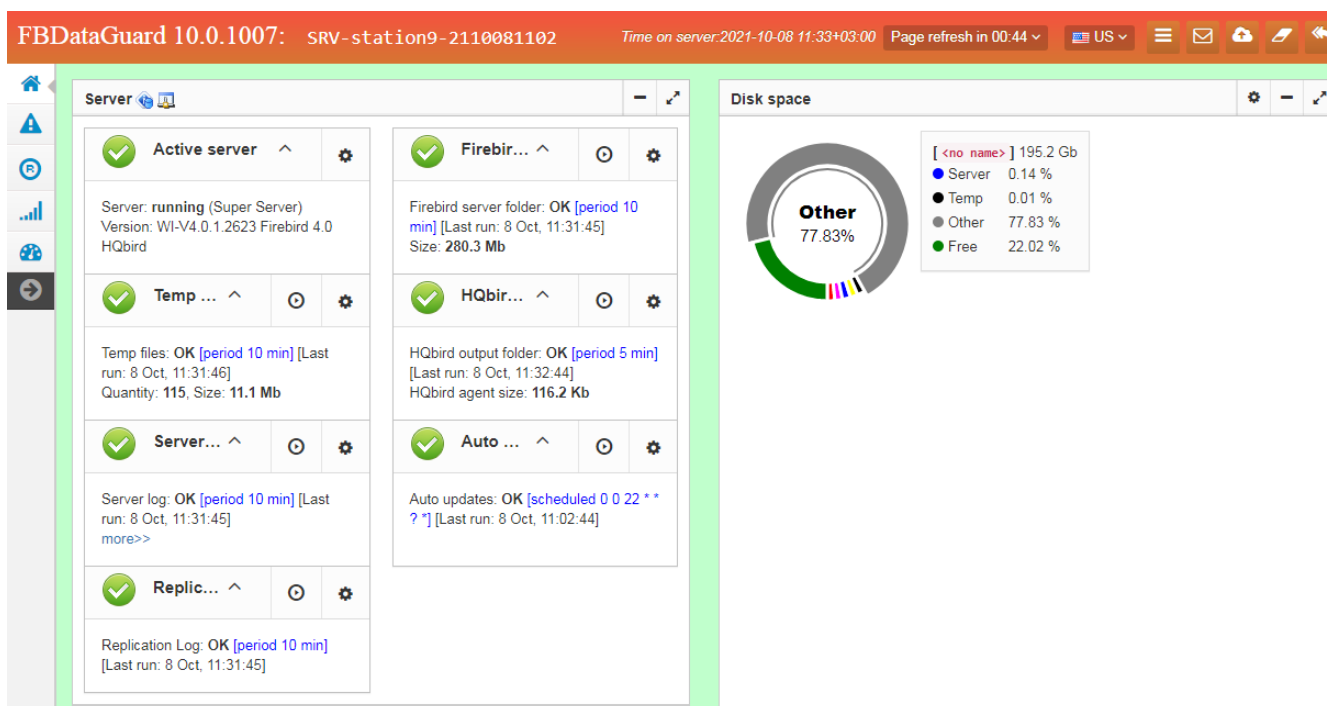


Figure 32. HQbird FBDaGuard with registered Firebird server.

FBDaGuard shows alerts and statuses of monitored objects: if everything is fine, it shows green signs, otherwise there will be yellow or red notifications.

Below we will consider in details each monitored objects and its settings.



Note: you cannot delete registered Firebird server in FBDaGuard web-console. The only way to unregister server is to delete its configuration files. In general, there is no reason for deleting registered server, until you want completely uninstall FBDaGuard.

Now we let's proceed with a database registration.

3.1.4. Firebird database registration

To register database in FBDaGuard, you need to click at the symbol “Settings” in the right corner of “Databases” (there will be a hint “Add database to monitoring”) and fill the following form:

Database monitoring configuration: "billing" ✕

Dataguard ID: 68a55275-dfc0-419f-90c1-87c5adb597b4

Database nick name:

DB alias:

Path to database:

User (optional):

Password (optional):

Output folder (backups and logs):

Enable advanced monitoring

[View database aliases](#) [View open databases](#)

Figure 33. Add database to monitoring.

- “**Database nick name**” is for your convenience, it is used to refer this database in alerts and email messages.
- “**DB alias**” is a database alias from *aliases.conf* or in *databases.conf*. If you specify both “DB Alias” and “Path to database”, “DB Alias” will be used.
- “**Path to database**” is the local path to database (remember that FBDataGuard should work at the same computer with Firebird). If you are putting database on external drive, it can raise error “File... has unknown partition”. To fix it you need to click on “Configure” at Server widget and click “Save” to make FBDataGuard re-read partitions.
- “**Crypt key name**” – if the database is encrypted, specify encryption key name.
- “**Crypt key value**” – if the database is encrypted, specify encryption key value.
- “**Output folder**” is the folder where FBDataGuard will store backups, logs and statistics for this database. If you do not select HQbirdData folder during the installation, and if you do not specify output folder for the server, it’s a good idea to specify “Output directory” to some explicit location like *F:\mydatabasedata*.
- “**Enable advanced monitoring**” - see [Advanced Monitor Viewer](#)



You can specify exact absolute locations for backups and statistics later in appropriate dialogs.

You can see the list of databases available for registration or their aliases by clicking on the link **View database aliases**.

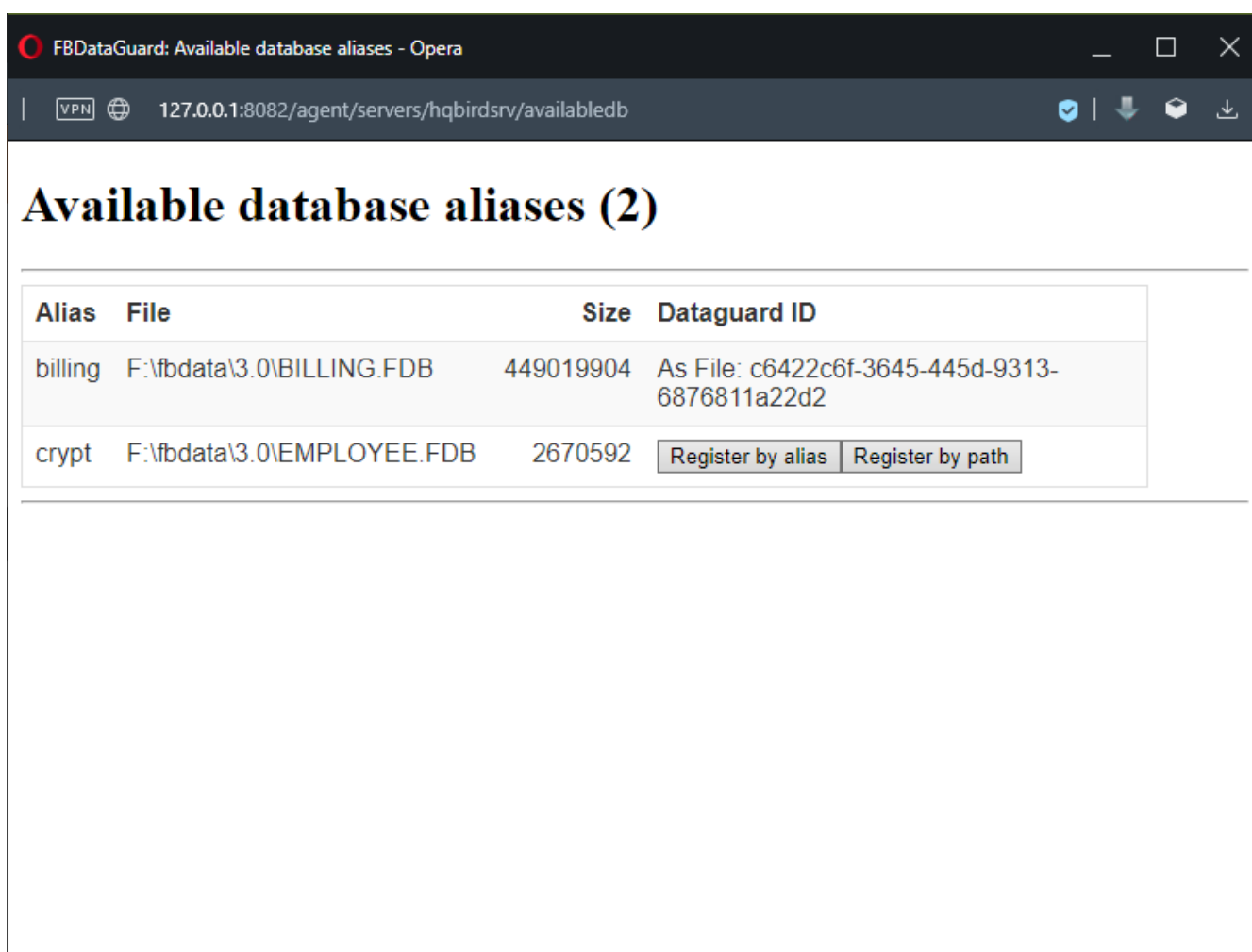


Figure 34. Available database aliases.

You can see the list of databases available for registration or their aliases by clicking on the link **View open databases**.

After registration, FBDataGuard will populate database configuration with default values and then show web-console with registered database:

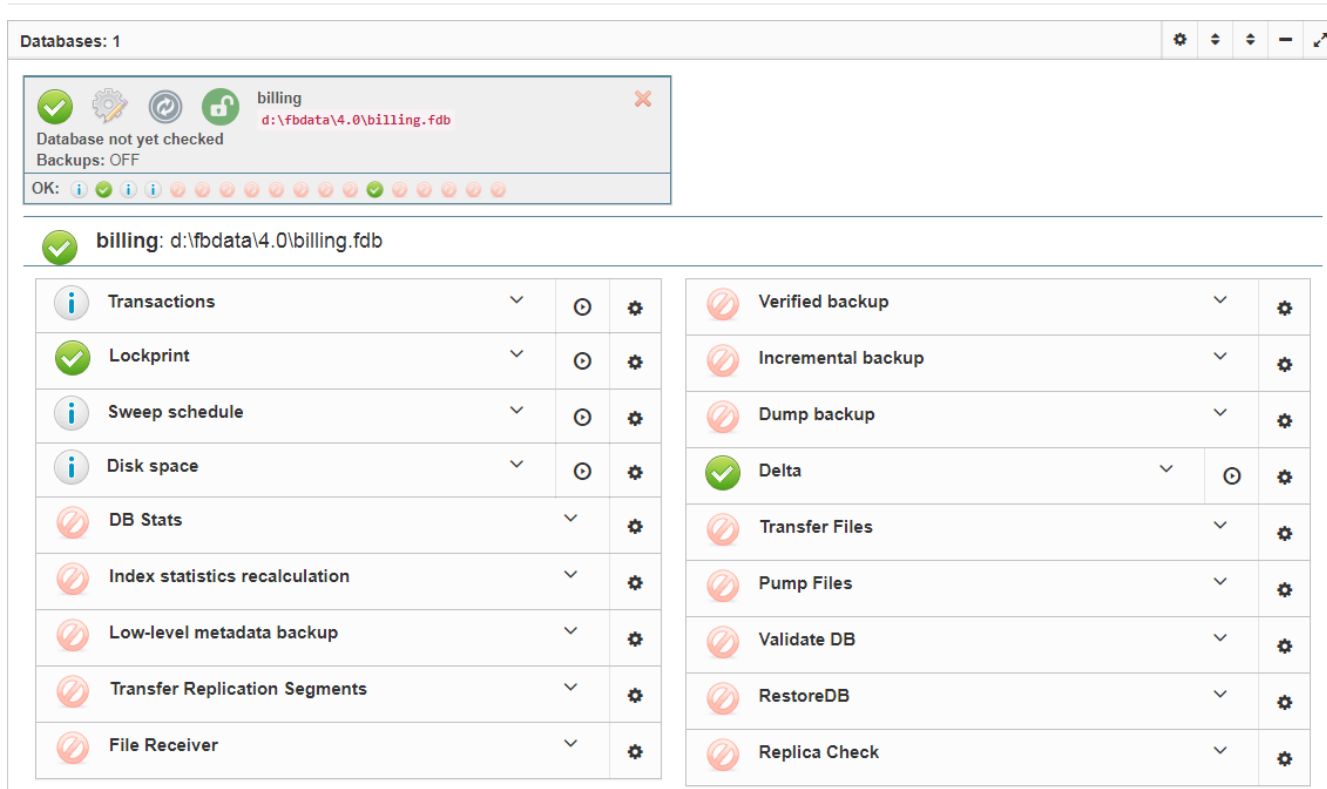


Figure 35. HQbird FBDataGuard web console after adding a database.

You can adjust database settings later; now let's proceed with alerts setup.

3.1.5. Email alerts in HQbird FBDataGuard

FBDataGuard can send alerts by email to administrator(s): such alerts contain information about successful backups and potential and real problems with databases.

General properties for notifications can be set by clicking on the server name (or computer name) at the top of the web-console:

FBDataGuard 10.0.1007: srv-station9-2110081102

After that you will see the configuration dialog for common alerts settings:

Dataguard Agent notify main properties
✕

Installation name:

✉

Installation GUID:

✉

Web console background color

✉

Keep maximum NN web notifications:

Hide duplicates for XX minutes:

Notify on alert reconfiguration or DataGuard startup

Generate daily report

Monitoring services install path

✉

Descriptions of some of the properties you can set here:

- **“Installation name”** is some readable name for your convenience; it will be referred in emails and alerts.
- **“Installation GUID”** is a service field; there is no need to change it.
- **“Web console background color”** – often it is useful to adjust the color of HQbird web interface to distinguish them easily.

It’s a good idea to enable setup email alerts. To do this you need to click on the envelope button in the top of the web-console:



After that you will see the configuration dialog for alerts:

Email alerts configuration

Send alerts by e-mail:

Send alerts to: support@email.to

'From' field: dg@host.from

SMTP server address: 127.0.0.1

SMTP server port: 25

SMTP server login: support

SMTP server password:

Send Test Message

Append version info to email subject

Add HQbird ID

Add detailed source name

Group notifications in emails

Limit max group size: 100

Accumulate messages YY minutes: 10

Cancel Save

Figure 36. Email alerts configuration dialog in FBDDataGuard.

First of all, you need to enable alerts sending by enabling checkbox “Send alerts by e-mail”.

- **“Send alerts by email”** - enable email alerts and configure email settings below.
- **“Send alerts to”** specify where to send emails.
- **“From field”** is what will be set as sender in the email.
- **“SMTP server address”, “SMTP server port”, “SMTP server login” and “SMTP server password”** are data which will be used to send emails.

Before saving the settings, you can click the "Send Test Message" button, if the settings are correct, you should receive a letter to the specified address.

In order to limit the number of letters, you can collect messages into groups and send them in

batches. To do this, set "Group notifications in emails" checkbox. It will also help bypass some of the anti-spam systems that can blacklist you due to too frequent send emails.

Click "Save" to save email alerts settings.

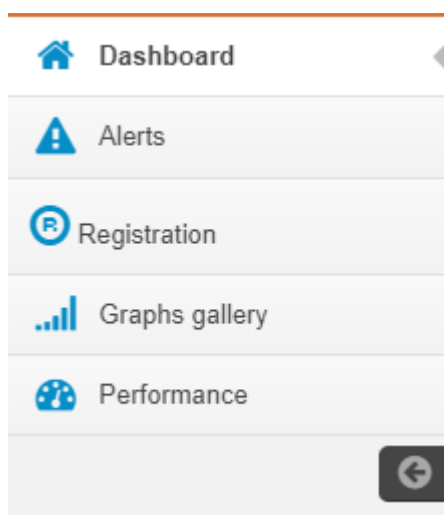
3.1.6. Next steps with FBDataGuard

After you have setup HQbird FBDataGuard and added there server and database(s) to be monitored, you need to adjust settings for the most important maintenance activities: backups, sweep and performance monitoring.

3.2. Monitoring and maintenance configuration in FBDataGuard

3.2.1. Overview of web-console.

Parts of web-console.



FBDataGuard Web-console contains 5 tabs (in the left side of the screen, usually they are collapsed):

- *Dashboard* – it is the main tab where administrator can configure HQbird FBDataGuard and see server and databases statuses.
- *Alerts* – contains the full list of alerts, generated by FBDataGuard.
- *Registration* – license and registration/activation information.
- *Graphs gallery* – performance graphs.
- *Performance* – performance monitoring settings and performance reports.

Jobs

Web-console is intended for easy configuration of activities (called “**jobs**”) which are fulfilled by HQbird FBDataGuard.

Almost all FBDataGuard jobs have 2 purposes: the first is to monitor for some values and to raise alerts if necessary, and the second is to store historical values into logs, so later it's possible to see the dynamics of important parameters of Firebird server and database.

In this section we will consider general configuration of jobs parameters, but not an analysis of gathered logs.

Jobs widgets

General approach is the following: each activity is represented by a “widget”, which has the following parts:

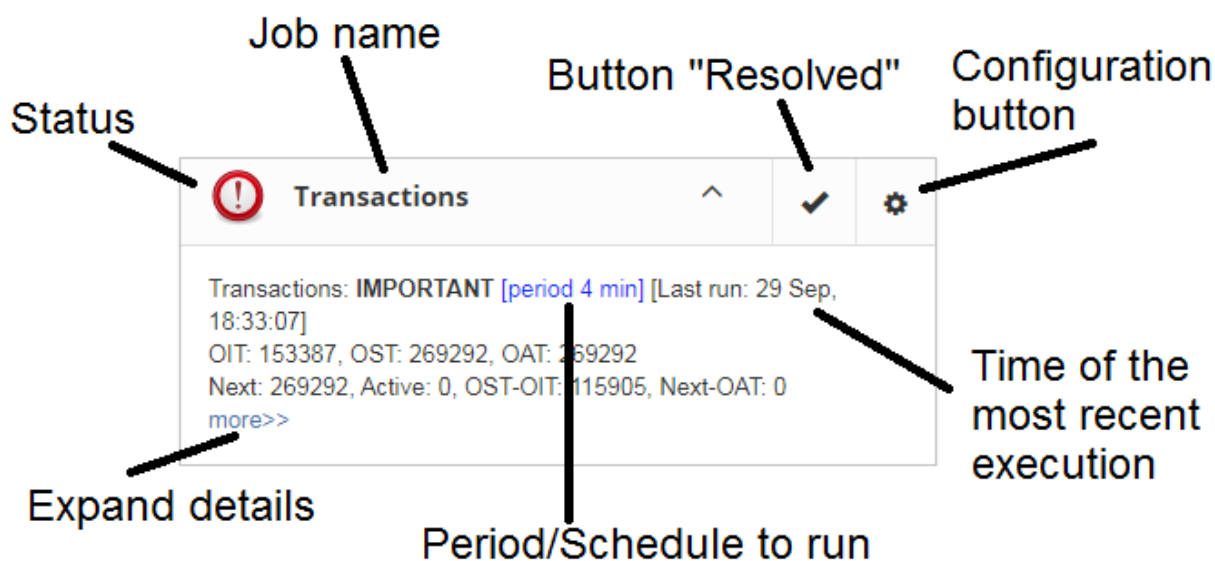


Figure 37. Elements of FBDDataGuard web-console widget.



Status—it is indicated with color icon and name. Status of database is a summary of all included server-level jobs and databases' statuses, and, respectively, status of database is a summary of all database-level jobs.

Status types

CRITICAL means problems, **OK** means “Everything is fine”, **WARNING** means some issues which require attention, **MAJOR** means major issue, **MINOR** – minor issue, **MALFUNCTION** means that jobs was not succeeded (something prevents its execution), **NOT_AVAILABLE** means that job is not available in this server or database version.

OFF means that job is not active, **UNKNOWN** means that job is active but was not started yet, so actual results are unknown.

Job name indicates the name of activity.

-  Configuration button opens configuration dialog, which is individual for each job.
-  Resolved is the link to flush the status to UNKNOWN and forgot errors which were

discovered previously. The status will be updated according the current situation after the next execution of the job.

Last run shows the time after the last run of this job.

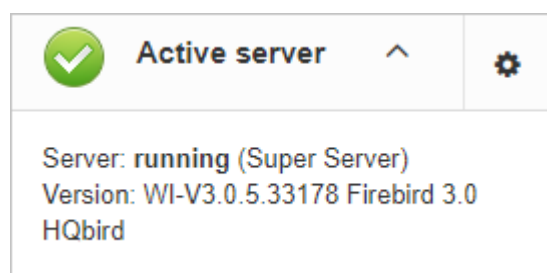
Period/Schedule to run shows how often or when the job will be started.

More>> is the link which opens the widget and shows more details and suggested action for administrator to resolve the situation.

All jobs in FBDataGuard have default settings, which are very close to recommended values for the 80% of Firebird installations, so after initial configuration server and database will be protected at pretty good level comparing with default installation, however, we recommend additional configuration and tuning for every job. In the next sections we will consider each job and its configuration.

3.2.2. Server: Active server

Server: Active server widget shows summarized status of all server-level jobs and statuses of monitored databases.



Server: Active server also indicates Firebird currently running or not and shows detailed version of Firebird and HQbird.

If you click on **configure** link, you will see the same dialog that we have used to register Firebird instance in FBDataGuard, and now it can be used for changing Firebird instance properties:

Server monitoring configuration: hqbirdsrv ✕

Installed in:	C:\HQbird\Firebird30	🗑
Binary folder:	C:\HQbird\Firebird30	🗑
Log:	\${server.installation}/firebird.log	🗑
Configuration file:	\${server.installation}/firebird.conf	🗑
Aliases:	\${server.installation}/databases.conf	🗑
Host:	localhost	🗑
Port:	3050	
	<input type="checkbox"/> Use trusted auth:	
SYSDBA login:	SYSDBA	🗑
SYSDBA password:	🗑
Output folder:	\${agent.default-directory}/\${server.id}	🗑

Cancel
Save

In general, there is no need to edit Firebird server details after the registration, until you are not reinstalling Firebird – but in this case we recommend reinstalling HQBird too.

3.2.3. Server: Auto updates

✓

Auto updates

^

⚙

Auto updates: OK [scheduled 0 0 22 * * ? *] [Last run: 26 Oct, 22:00:03]

Auto update is an important job which notifies you that newer version of HQbird FBDataGuard is available and suggests to update the software.

It provides an alert regarding available updates and appropriate download link. Default time to run this job is 22-00 everyday (for information).

In the configuration dialog of auto-updates you can disable auto check or set another time for it.

Agent updates monitoring
✕

Enabled

Schedule:

URL to get new version info:

URL to download new version:

URL Login:

URL Password:

Use passive mode for ftp

Limit download file size:

Folder for backup of dataguard files:

Download new version after check

In fact there is a small confusion here: auto update does not perform automatic update of HQbird, it just checks for the new versions periodically, and optionally download it.

The upgrade is initiated by the administrator.

3.2.4. Server: Replication Log

Server log
^
⚙️

Server log: OK [period 10 min] [Last run: 27 Oct, 14:23:25]

If you are using HQbird Enterprise, FBDataGuard can check *replication.log* for errors. In case of error it sends an appropriate alert (by email) to the administrator.

To enable this job please check “Enabled”.

Replication log monitoring
✕

Enabled

Check period, minutes:

Size to roll, bytes:

Date pattern for rolling:

Pack renamed files

- Check period – how often to check *replication.log* for changes
- Size to roll, bytes — if *replication.log* will exceed will value, it will be renamed according date-time pattern
- Date pattern for rolling – how to rename *replication.log*
- Keep NN error messages: how many errors will be stored in the list of the recent errors.

3.2.5. Server: Server log

✓

Server log

^

⚙️

Server log: OK [period 10 min] [Last run: 27 Oct, 14:43:25]

“Server log” job periodically checks *firebird.log* and if it detects that file was changed, log analysis starts. The embedded analytic engine checks each entry in *firebird.log* and categorizes them into several categories with different levels of a severity. According the severity of messages status of job is assigned and appropriate alerts are generated.

Once administrator has reviewed errors and alerts (and performed necessary actions to solve the reason of error), he need to click on **“Resolved”** link and FBDataGuard will forget old error messages in *firebird.log*.

In the configuration dialog of “Server log” you can enable/disable this job and set the check period (in minutes).

Server log monitoring
✕

Check period, minutes:

Size to roll, bytes:

Date pattern for rolling:

Enabled

Notify on critical message

Notify on important message

Notify on minor message

Send summary

Pack renamed files

Cancel
Save

Also this job watches for the size of *firebird.log* and if its size exceeds “Size to roll”, FBDataGuard will split *firebird.log* and rename it according to the date-time pattern.

3.2.6. Server: Temp files

✓
Temp files
^
⚙

Temp files: OK [period 10 min] [Last run: 27 Oct, 15:23:34]

Quantity: 56, Size: 22.1 Mb

“Server: Temp files” job is useful to catch and solve performance problems with Firebird database.

While performing SQL queries Firebird stores intermediate results for sorting and merging data flows in temporary files, which are allocated in specified TEMP locations. FBDataGuard shows at “Server: Temp files” widget information about quantity and size of temporary files.

FBDataGuard recognizes locations of TEMP folders and monitors quantity and size of temporary files. Lack of space can lead to the performance problem or more serious errors, too many (or too large) temporary files can indicate problems with SQL queries quality.

Temporary files monitoring

Enabled

Check period, minutes:

Maximum size, bytes:

Maximum number:

Cancel Save

Using configuration dialog you can enable/disable this job, set period and thresholds to the maximum size of temporary files (size of all files) and quantity.

If you see that size of temp files is too high and if there is enough RAM on the server, increase `TempCacheLimit` parameter in `firebird.conf` to fit all temporary tables into RAM.

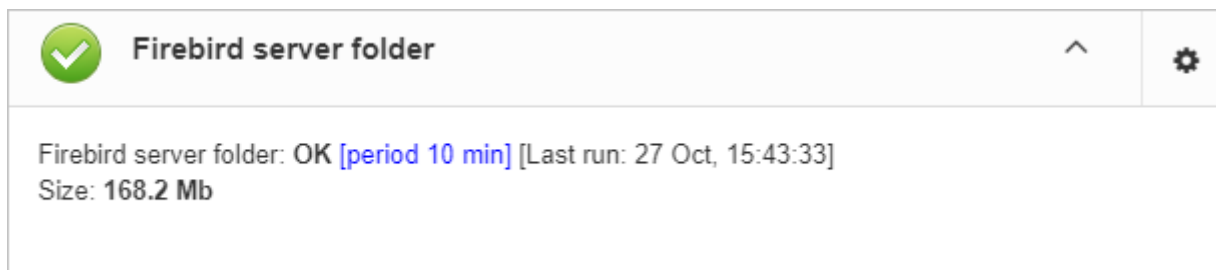
Also, HQbird checks other temp files used by Firebird — if you see extreme values (several Gb) for trace or monitor, the good idea will be check the `FIREBIRD_TMP` folder for outdated files (with old modification timestamps). Please note — the screenshot below is not a real alert (i.e., values are Ok), it was created to demonstrate the output in case of large temporary files.

Temp files

Temp files: **IMPORTANT** [Last run: 26 Nov, 12:45:59]
 Quantity: **121**, Size: **23.3 Mb**
 <<less
 Total size of all temporary files 24.4 MB is more than recommended (88.0 kB). Files count 121:
 fb_table_: 33 (11.8 MB)
 fb_repl: 1 (65.5 kB)
 fb_lock: 2 (10.5 MB)
 fb_monitor: 1 (1.0 MB)
 fb_trace: 83 (36.4 kB)
 fb_user_mapping: 1 (1.0 MB)
 33 files in "C:\Windows\Temp"; size: 11.8 MB;
 88 files in "C:\ProgramData\Firebird"; size: 12.7 MB;
 Firebird creates temporary files for some SQL queries (PLAN SORT). Too big size of temporary files can indicate performance problems. This is not a strictly defined number, so this threshold depends on particular database and application.

3.2.7. Server: Firebird server folder

“Firebird server folder” jobs monitors size, occupied by Firebird installation. It’s enabled by default.



There are several threats prevented by this job: maladministration issues when database volumes or external tables are being created in `%Firebird%\Bin` folder, very big `firebird.log` which can exhaust all places at drive with Firebird installation, and some other problems.

Also this job monitors and analyses information, gathered by all space-related jobs (including database-level jobs). At the picture below you can see quick representation of space analysis for all drives where Firebird, databases and backups are stored.

Using configuration dialog you can enable/disable this job, set period of checking and thresholds for server folder size.

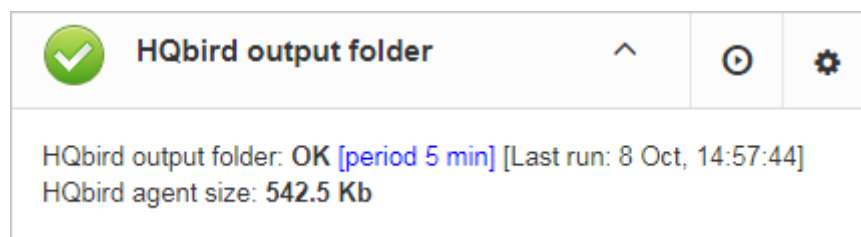
By default we use 200 Mb is a standard setting for Firebird installation.

If the size of your Firebird is larger, please consider clean-up of old logs and other unwanted artifacts, or increase parameter **Max occupied** (in bytes) to prevent false alerts.

Note for Linux users: if you see red warning regarding the inconsistent space information, add locations with database and backups to Disk Space widget:

You can get idea where is your database and backup is actually located with command `df -h`.

3.2.8. Server: HQbird Output Folder



“HQbird output folder” monitoring is intended to watch space occupied by reports, logs, stats, metadata repository and other data, gathered and generated by HQbird – this folder by default is *C:\HQbirdData\output*.

For databases unattended for a long time (1-2 years) it is possible that FBDataGuard logs will occupy too much space and lack of space can lead to database outage. To prevent it for sure, “HQbird output folder” is watching for occupied space.

By default “HQbird output folder” job is enabled.

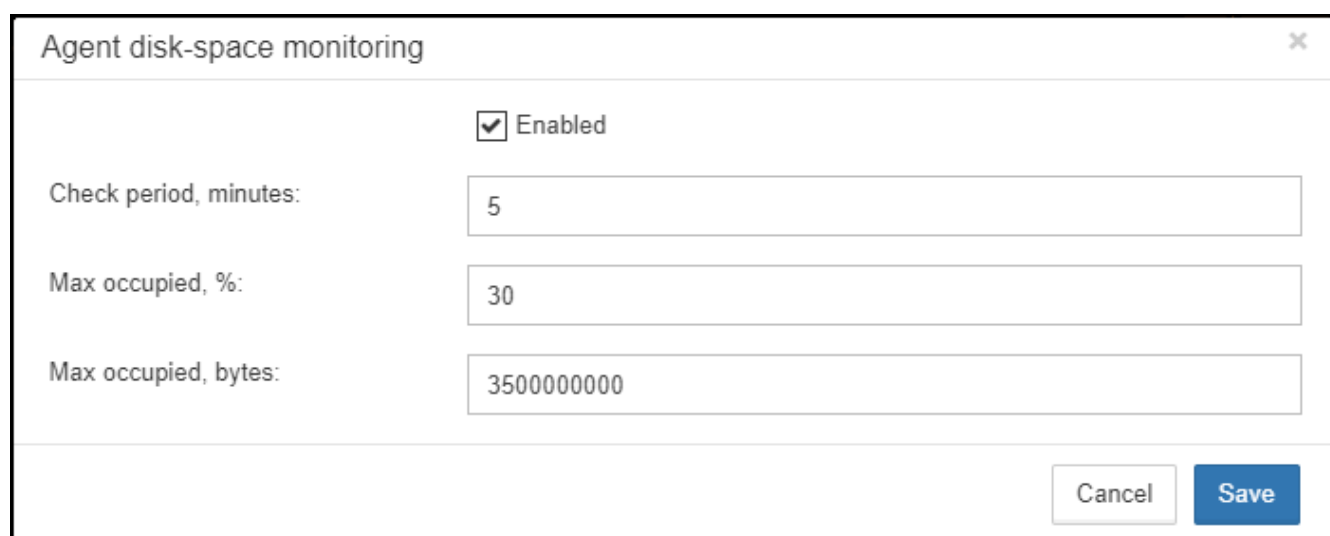
Also, if someone has ignored recommendations to put backups’ folders to the explicit locations, it is possible that database backup will be created inside Agent folder. In this case you’ll see CRITICAL status immediately — FBDataGuard will recognize and warn you regarding wrong configuration.

And, this job is useful for bundles of FBDataGuard and third-party applications.

In the configuration dialog you can enable/disable this job, set check period (by default it is 10 minutes), and set thresholds for alerts.

Thresholds can be set in % of max size occupied by log or using the explicit size in bytes.

FBDataGuard checks both values and raises alert for the first threshold. If you wish to set % only, you need to set -1 as value to “Max occupied”.



3.2.9. Database: General configuration

FBDataGuard can monitor several databases at the single server (up to 80 databases). For each database the separate widget is created. At the top widget database status is shown, database

nickname (it's specified during database adding and can be changed). Also database widget shows the full path to the database, its size, status of backups and the number of currently connected users.



Using configuration dialog you can set database nickname, path to database and output folder for the database (to store logs and jobs results).

 A screenshot of the "Database monitoring configuration: 'billing'" dialog box. The title bar includes a close button. Below the title, the "Dataguard ID" is "9b6e6443-de7e-428c-af75-a7b9b520f533". The form contains several fields:

- "Database nick name:" with the value "billing".
- "DB alias:" with an empty field.
- "Path to database:" with the value "d:\fbdata\4.0\billing.fdb".
- "User (optional):" with the value "\${server.sysdba-login}".
- "Password (optional):" with the value "\${server.sysdba-password}".
- "Output folder (backups and logs):" with the value "\${server.default-directory}/\${db.id}".

 At the bottom left, there are two links: "View database aliases" and "View open databases". At the bottom right, there are "Cancel" and "Save" buttons.

FBDataGuard checks the validity of path to database and it does not allow specifying the wrong path.

Also, for HQbird Enterprise, in the database widget you can see status of the replication and configure replication by clicking on the icon. Please read more details in the replication configuration section.

Since HQbird 2020, the database widget in HQbird also shows the encryption status of the database.

3.2.10. Database: Transactions

"Database: Transactions" job is intended to log transactions activity. It monitors 2 important intervals: difference between Oldest Active Transaction and Next transaction and gap between Oldest Snapshot and Oldest Interesting.

If these intervals are out of the frames of the specified threshold, it means problem with

transactions management.

Transactions monitoring / test
✕

Enabled

Check period, minutes:

Max transactions:

Alert when OST-OIT more than:

Alert when Next-OAT more than:

Use services API

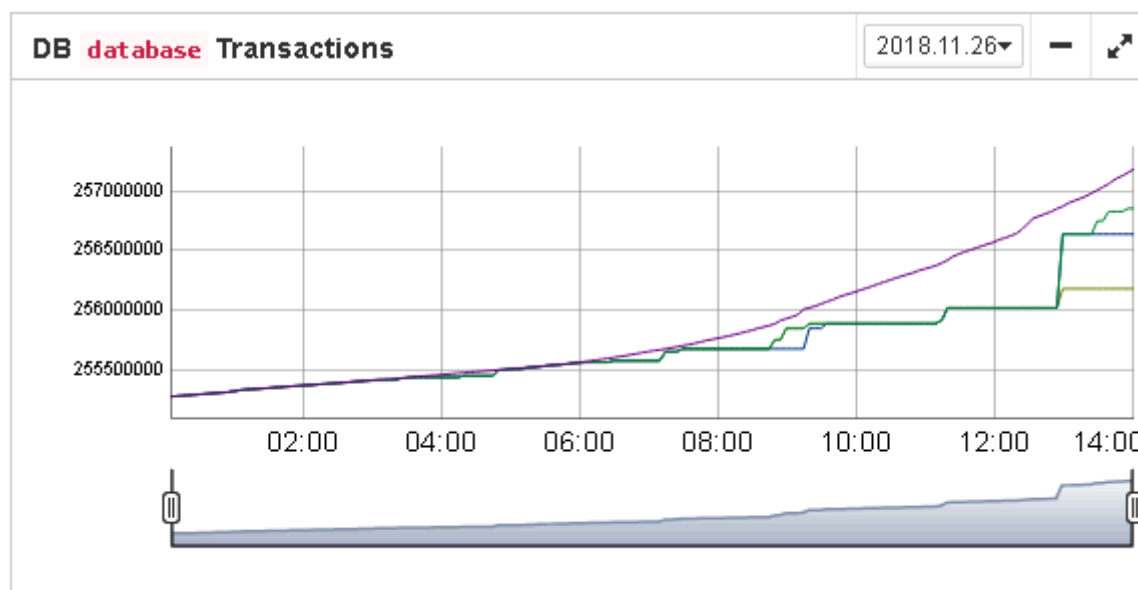
Cancel

Save

These logs can be analyzed to get helpful insight regarding database performance and application quality (see more information here <http://ib-aid.com/en/articles/ibanalyst-what-you-can-see-at-summary-view/>).

This job also monitors the implementation limit in Firebird: maximum transactions number in Firebird versions before 3.0 should be less than $2^{31}-1$.

Near this number database should be backup and restored. It will throw an alert if transaction number will be close to the restrictions. Also, the transaction dynamics is shown on the tab “Graphs gallery”:



3.2.11. Database: Lockprint

“Lockprint” job monitors the information from the lock table of Firebird. It is very important for

architectures Classic/SuperClassic and useful for SuperServer.

The lock table is an internal Firebird mechanism to organize access to the objects inside Firebird engine. HQbird monitors the important parameters of the lock table:

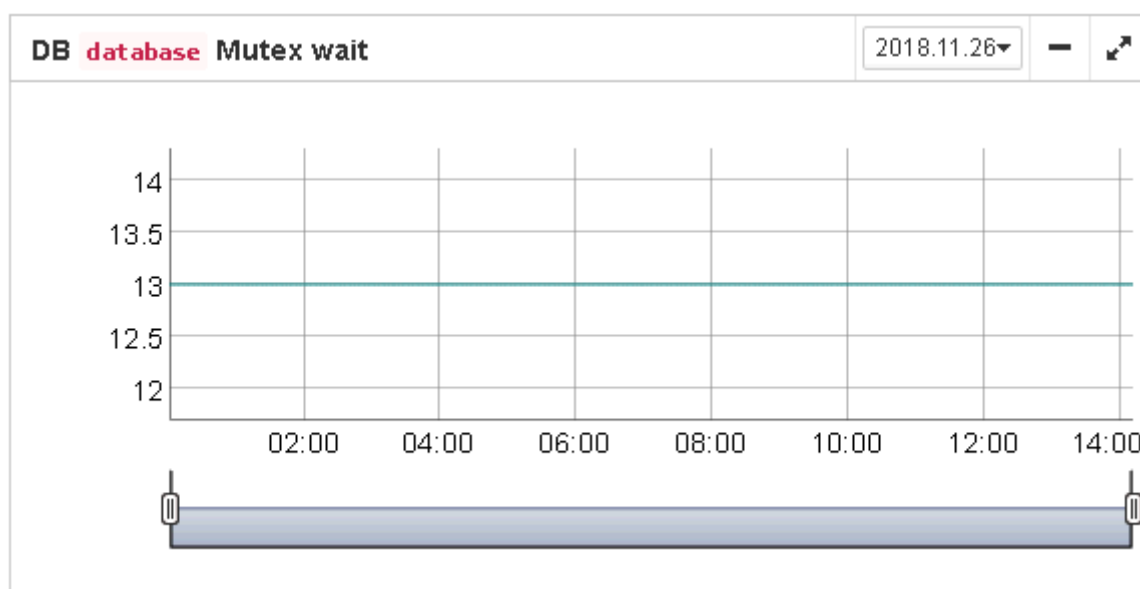
Lock Table Analysis / billing
✕

	<input checked="" type="checkbox"/> Enabled
Check period, minutes	<input style="width: 80%;" type="text" value="3"/>
	<input checked="" type="checkbox"/> Send alert if number of Deadlock scans more than
Deadlock Scans threshold	<input style="width: 80%;" type="text" value="12345"/>
	<input checked="" type="checkbox"/> Send alert if number of Deadlocks exceeds:
Deadlock threshold	<input style="width: 80%;" type="text" value="0"/>
	<input checked="" type="checkbox"/> Send alert if Mutex Wait more than
Mutex Wait threshold	<input style="width: 80%;" type="text" value="18"/>
	<input checked="" type="checkbox"/> Hash slots alert
Hash Slots is less than	<input style="width: 80%;" type="text" value="2047"/>
Min length is more than	<input style="width: 80%;" type="text" value="1"/>
Average length is more than	<input style="width: 80%;" type="text" value="6"/>
	<input checked="" type="checkbox"/> Send alert if number of database connections is more than
Owners limit	<input style="width: 80%;" type="text" value="700"/>
Free owners limit	<input style="width: 80%;" type="text" value="1000"/>
	<input checked="" type="checkbox"/> Send alert if size of the lock table is more than
Lock table size	<input style="width: 80%;" type="text" value="40000000"/>
Warn if page buffers in database header more than [NNN]	<input style="width: 80%;" type="text" value="10000000"/>

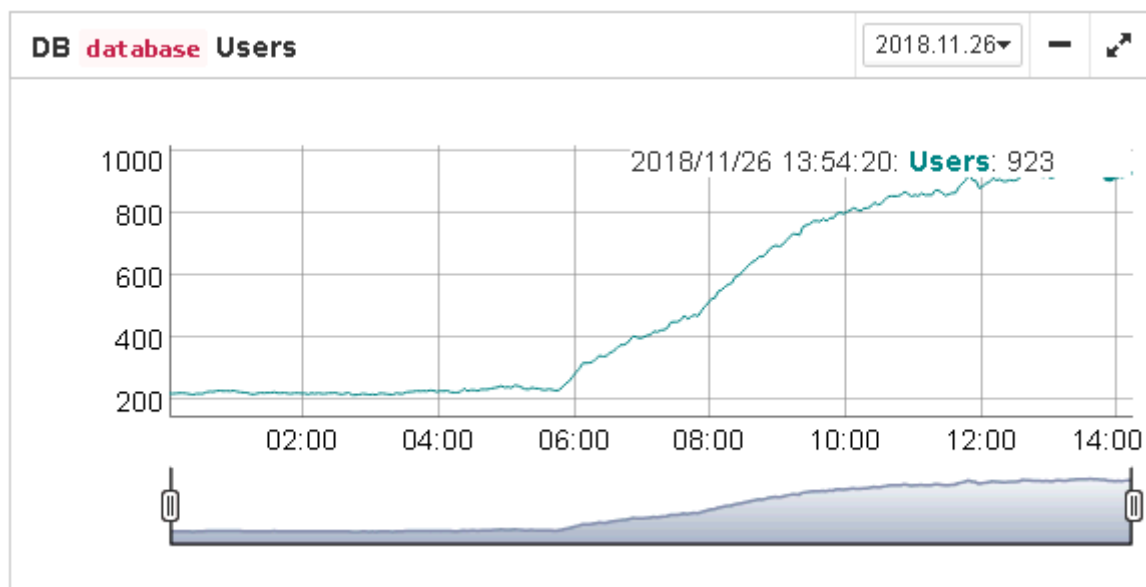
- **Check period**, minutes—how often HQbird analyses lock table. 3 minutes is an optimal

interval.

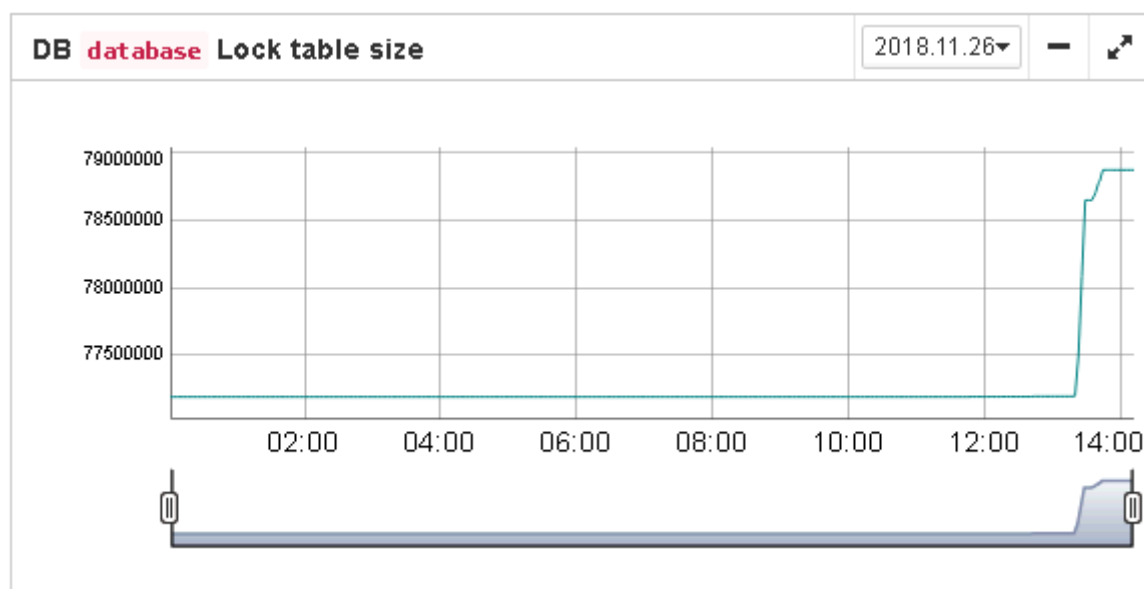
- **Deadlock Scans threshold** — deadlock scan is a process, started by Firebird engine, in case of a long response delay from the one of the threads. If a number of deadlock scans is high, it means that Firebird is heavily loaded. The value is accumulated since the Firebird engine start. The default value is pretty big – 12345, so if it is exceeded, it means that database performance is poor.
- **Deadlock threshold** — if Firebird engine finds the true deadlock during the deadlock scans, it increases this value. Please note: true deadlocks are very seldom. Don't confuse them with transactions' conflicts (“deadlock. Lock conflict on nowait transaction” etc).
- **Mutex wait threshold** — Mutex Wait is a parameter of lock table which implicitly indicates the conflicts for the resources. The higher mutex wait, the more competition exists inside the engine for the resources. By default, the mutex wait threshold is set to 18%, but this value is not universal for all databases. The good approach is to watch for the mutex values during 1-2 weeks and then set the highest value seen during this period. Mutex wait graph is available in Mutex Wait gallery.



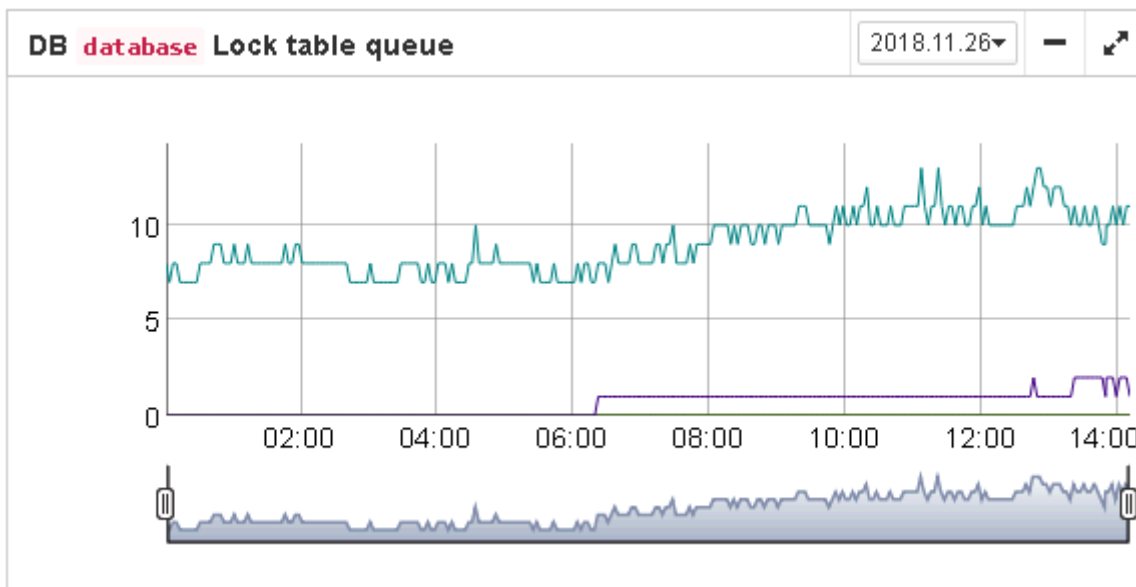
- **Hash slots alerts.** Lock table header has a parameter “Hash lengths (min/avg/max): 0/0/4”, it shows the lengths in the lock table. It is important to keep these values as low as possible, so HQbird monitors them and suggest, how to improve the situation, if hash length is more than specified in this job.
- **Owners limit.** “Owners” is a number of connections established to the specified database. In fact, this is the fastest way to get the actual number of connections to the database with the minimum load to the database — other ways like request to MON\$ATTACHMENTS or isc_tpb_database have various disadvantages. The limit here should be set according the actual peak number of connections. For example, if you are sure that peak number of the connections to your database is 500, set 550 as Owners limit, and if at some moment the load will increase, you will not miss that moment.



- **Free owners.** “Free owners” is the value between the peak number of owners and current number of owners. If you see Free owners = 0, it means that number of connections grows steadily since the Firebird start. If you see high number of Free owners, it can be sign that many connections were disconnected recently.
- **Lock table size.** The lock table size is an implicit indicator of the load to the system. Normally, lock table size should be stable. Also, it is recommended to set the initial lock table size to the value it has after some active work period — though the lock table is enlarged on demand, the re-allocation process is a heavy operation and can lead to micro-freezes in database responses. Lock table graph is useful to determine the proper initial value.



- **Lock table queue.** Lock table queue does not have the explicit threshold in Lockprint job, but its values are collected and shown in “Graphs gallery”. Lock table queue is an indicator of a general load.



3.2.12. Database: Index statistics recalculation

“Database: Index statistics recalculation” is an important job which helps to keep performance of indices at optimal level, and performs additional checking of a database health.

“Database: Index statistics recalculation” allows to run re-computing of indices selectivity values. During this procedure Firebird quickly walks through leaf pages of indices, and renews statistics about selectivity. By visiting these pages Firebird also verifies their integrity and if index is corrupted, the warning will be thrown.

Also, this job verifies that all indices are active in database. Inactive or non-activated indices usually indicate corruption and lead to performance degradation.

By default this job is disabled, but we recommend enabling it after careful selecting of indices for the recalculation.

There are three modes in this job: AUTO, ALL, SELECTED.

ALL is the mode where all indices will be checked.

AUTO is the default mode. It is very similar to ALL, but it also checks the size of database and do not touch indices if database is bigger than 3.6Gb.

Update index statistics configuration / billing

Enabled

Schedule:

Update mode:

Included indices names:

Excluded indices names:

DB size to switch, bytes:

Check index activity

Cancel Save

SELECTED is the recommended mode. It allows choosing of indices which should be recomputed or those which should be avoided.

To include indices into the list of recomputed, you need to specify indices names (divided by comma), and to exclude – perform the same in the appropriate field.

As you can see at configuration dialog screenshot, there are fields to enable/disable job, to set update mode, and to include or exclude indices. “DB size to switch, bytes” is to set limit where AUTO mode is working. “Check index activity” switch should be always on, until you are not performing special manipulations with inactive indices.

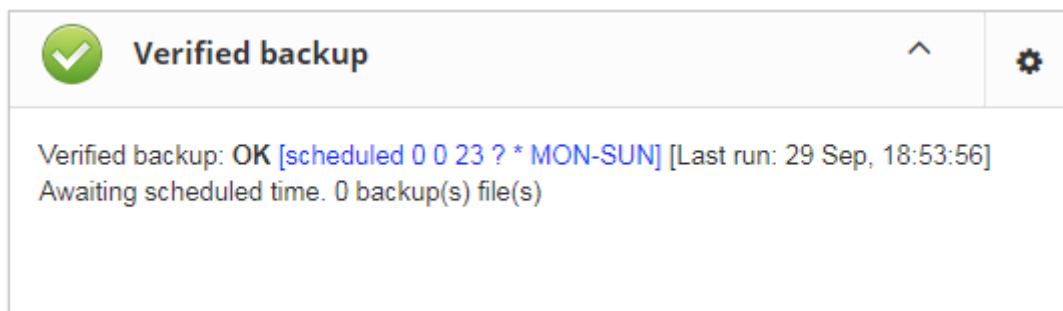
3.2.13. Database: Verified Backup

“Database: Verified Backup” is one of the key jobs to guarantee the safety of data stored in the protected database. During the development of HQbird we had in mind certain recovery scenario, and this scenario implies that the key goal of database protection is to minimize potential losses of data. If we have healthy backup, recovery can be concentrated on saving the most recent data (just entered into the database), and it greatly decreases the time of overall outage.

As you will see below, “Database: Verified Backup” is not just a wrapper for standard gbk functionality and scheduler, this is a smart job which has many built-in rules to prevent problems with backups and provide suitable interface for backups management.



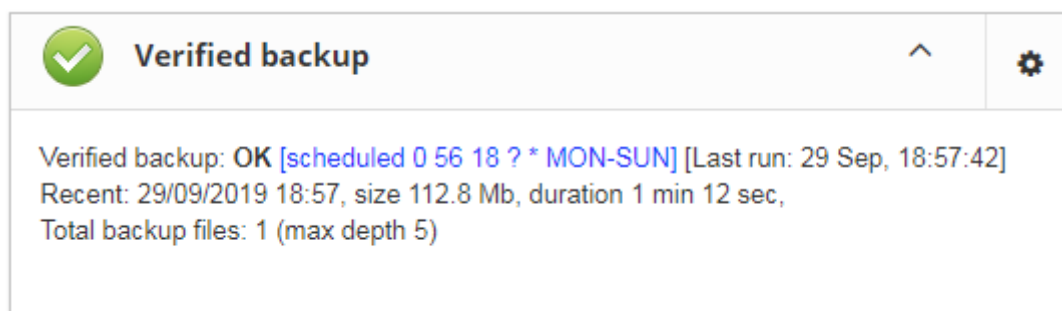
“Database: Verified Backup” is disabled **by default**, but we strongly recommend reviewing of its settings immediately after HQbird setup.



Initially “Database: Verified Backup” job is shown as Ok, though backup was not tried. In this case OK means that backup at least scheduled.

Also this job recognizes files according the name pattern (see below information regarding configuration), and shows the totals number of backups.

After the backup will be done, the widget information will be changed: creation time of last successful backup will be shown, and also the time took to actually perform the backup (only 1 minute 12 seconds at the screenshot with example).



Also, the detailed alert will be send to your email and/or HQbird Control Center:

Name	Desc
Regular backup was done successfully.	<p>Backup 'C:\HQBirdData\output\output\hqbirdsrv\4e6c4186-1c24-4f97-b3fd-6b74db7f6518\backup\backup_20190929_18-56.zbk' 1.3 GB was created at 2019-09-29 18:56:00.001+03:00 took total 01m:42s.691 to complete. Test restore is omitted.</p> <p>Backup: [OK] Backup [C:\HQBirdData\output\output\hqbirdsrv\4e6c4186-1c24-4f97-b3fd-6b74db7f6518\backup\backup_20190929_18-56.fbk] 1.3 GB done successfully at 2019-09-29 18:57:12.228+03:00, taking 01m:12s.213.</p> <p>Other: [OK] Archive [C:\HQBirdData\output\output\hqbirdsrv\4e6c4186-1c24-4f97-b3fd-6b74db7f6518\backup\backup_20190929_18-56.zbk] 118.3 MB done successfully at 2019-09-29 18:57:42.690+03:00, taking 00m:30s.448.</p>

“Database: Verified Backup” checks the free space at the drive with backup destination, and if it detects that there is not enough free disk space, CRITICAL alert will be sent, and current backup will be canceled (if necessary).



Be careful – by default backup time is set to **23-00 Monday-Sunday**.

By default, database backups will be stored into the output folder that you have specified during installation step! By default, it is `C:\HQbirdData\output...`

It is very important to carefully review database backups settings and adjust them according the local configuration!

Let's consider the configuration dialog for backup in more details:

- **“Enabled”** is obvious – it enables or disables scheduled backups
- In the **“Schedule”** field you can set the time when backup should be run. Scheduler uses CRON expression and this is a right place to apply all the power of CRON (see [CRON Expressions](#)).
- **“Backups folder”** specifies the folder to store backups. This folder should be at the same computer where database is. By default, it is situated inside database default directory. Usually it's a good idea to set the explicit path to the folders with backups.
- **“Maximum number of backup files in folder”** specifies how many previous backups should be stored. FBDataGuard stores backups in revolver order: when the maximum number will be reached (i.e., 5 backups will be created), FBDataGuard will delete the oldest backup and create the new backup. In combination with CRON expressions it gives a powerful ability to create necessary history of backups.
- **“Backup name pattern”** specifies how backup files will be named. Also this name pattern allows FBDataGuard to recognize old backups with the same name pattern.
- **“Backup extension”** is `fbk` by default.
- **“Compress backups”** specifies should FBDataGuard archive backups after regular Firebird backup. By default, this option is on, but you need to know that FBDataGuard will zip backups' files which are less than **100 Gb** in size. After that size, the backup compression will be automatically switched off. We recommend to turn this feature on for small databases only.
- **“Check restore”** is an important option. If it is on (by default), FBDataGuard will perform test restore of fresh backup, in order to test its validity. It guarantees the quality of created backup and notifies administrator in case of any problems with test restore.
- **“Remove restored”** specifies should FBDataGuard delete restored database. By default it is OFF, so you might want to turn it ON, but you need carefully consider – do you really need to keep the copy of test restored database. With each test restore this copy will be overwritten.
- **“Use multiple cores to backup and test restore”** - this feature is for HQbird Enterprise only, it allows to backup database and restore test database using multiple CPU cores, so backup can be made 3-5 times faster. We recommend to allocate $\frac{1}{2}$ of CPU cores,
- **“Send "Ok" report”** – by default it is off, but it's strongly recommended to turn it ON and start to receive notifications about correct backups. This feature will use email settings from alerts system

Backups configuration / test
✕

Enabled:

Schedule:

0 0 23 ? * MON-SUN

Backup folder:

\${db.default-directory}/\${job.id}

Maximum numbers of backup files in folder:

5

Backup name pattern:

backup_{0,date,yyyyMMdd_HH-mm}

Backup extension:

.fbk

Compress backups:

.zbk

Check restore:

\${backup-directory}/restore.\${db.id}.fdb

Remove restored

Use NN CPU cores to backup and test restore:

N/A

Send 'OK' report

more>>

Cancel

Save

If we will click on button **[More>>]**, the advanced backup options will appear:

Backup (gbak) timeout, minutes:	<input type="text" value="240"/>
Restore (gbak) timeout, minutes:	<input type="text" value="480"/>
Final destination folder for backups	<input type="text" value="{backup-directory}"/>
<input type="checkbox"/> Copy backup:	<input type="text" value="/mnt/backups"/>
<input type="checkbox"/> Execute shell command:	<input type="text"/>
Optional path to gbak executable:	<input type="text"/>
Backup options for gbak:	<input type="text" value="-ST TDRW"/>
Restore options for gbak:	<input type="text"/>

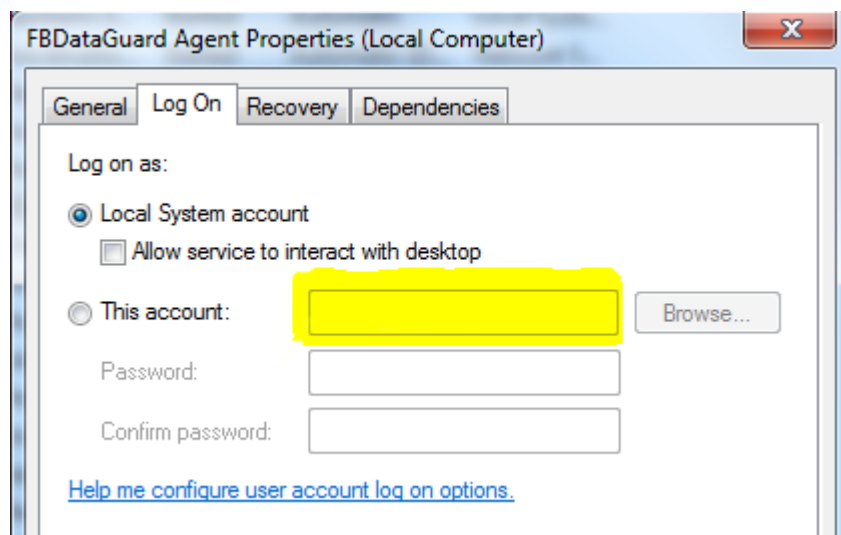
- **“Backup (gbak) timeout, minutes”** - maximum time to complete only backup (gbak -b) operation, otherwise alert will be generated.
- **“Restore (gbak) timeout, minutes”** – maximum time to complete test restore operation.
- **“Final destination folder for backups”** - if you need to make backups into the one folder, and then move created backup to another folder (for long-term storage, for example), you can change the value of this parameter from *{backup-directory}* to the folder where you will keep them. Backup files in both locations are watched by HQbird FBDataGuard, and included into the count of backup copies shown in the widget.
- **“Copy backup”** switch and **“Copy backup to”** path. If you have network location or plugged USB drive to store database where you want to store copy of backup (in addition to usual backups), FBDataGuard can copy the latest backup there: just turn on “Copy backup” switch and specify **“Copy backup to”** path. The copied files are not monitored and not included into the number of backup files shown in the widget.
- **“Execute shell command”** switch and **“Shell command”** path. It is possible to specify custom script or executable after the general backup procedure will be complete. Shell command gets as the path to the fresh database backup as a parameter.
- **“Optional path to gbak executable”** - it is possible to specify other gbak tool than standard *gbak*.
- **“Backups option for gbak”** - if you need to add some specific options, add them here.
- **“Restore options for gbak”** - if you need to add specific options for test restore, add them here.



If you are monitoring more than one database, it is highly recommended splitting the runtime of the restores.

Important Note: Backup to the network locations

Please be aware that for creating and copying backup to the network locations Firebird and FBDataGuard services must be started under the account with sufficient rights. By default, Firebird and FBDataGuard are started under LocalSystem account, which does not have rights to access network location.



So, to store Firebird backups to the network location on Windows, run Services applet (*services.msc*) and on the tab Log On change “Log on as” to the appropriate account (Domain Admin should be fine).

For Linux – add necessary rights for “firebird” user.

3.2.14. Database: Incremental Backup

Incremental backup is a job to schedule and manage incremental backups in Firebird.

Please note that we recommend to use incremental backups only in combination with verified backups, since incremental backup performs copying of database pages changed since the last backup (in case of multilevel incremental backup).

HQbird FBDataGuard implements 2 types of multilevel incremental backup: Simple and Advanced incremental backups, and also Dump backup (see [Database: Dump backup](#)).

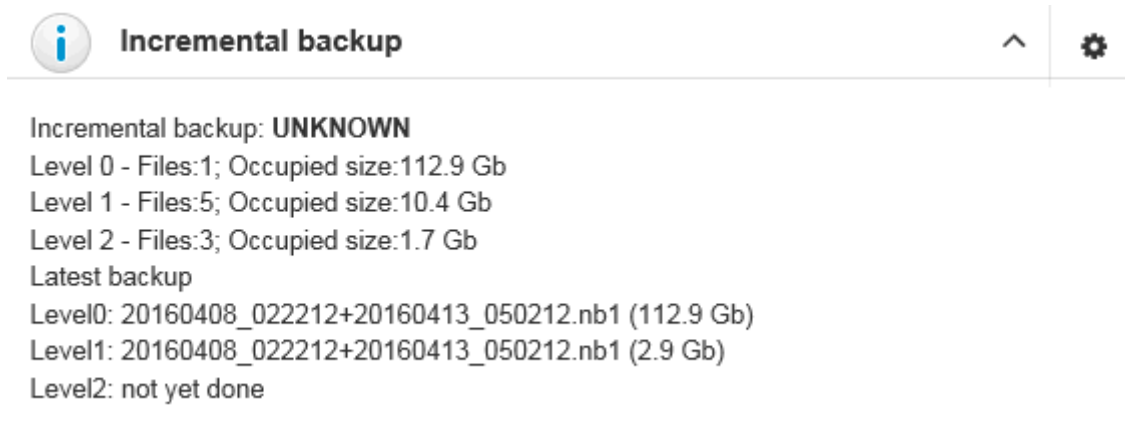
Multilevel backup in Firebird must follow the following steps:

1. Create initial backup (level 0) which essentially is the copy of the database at the moment of backup start and mark it with backup GUID.
2. Since Firebird marks each data page with a certain identifier at every change, it is possible to find data pages, changed from the moment of previous backup and copy only them to form backup of level 1.
3. It is possible to create several level of the backups – for example, the initial backup (full copy,

level 0) is being created every week, every day we create level 1 (differences from the level 0), and at every hour we create level 2 backups (differences from daily level 1).

Incremental backup with simple schedule allows planning 3 levels of backups: weekly, daily and hourly.

You can see summary information for such incremental backup configuration at the following screenshot of its widget:



Incremental backup

Incremental backup: **UNKNOWN**

Level 0 - Files:1; Occupied size:112.9 Gb

Level 1 - Files:5; Occupied size:10.4 Gb

Level 2 - Files:3; Occupied size:1.7 Gb

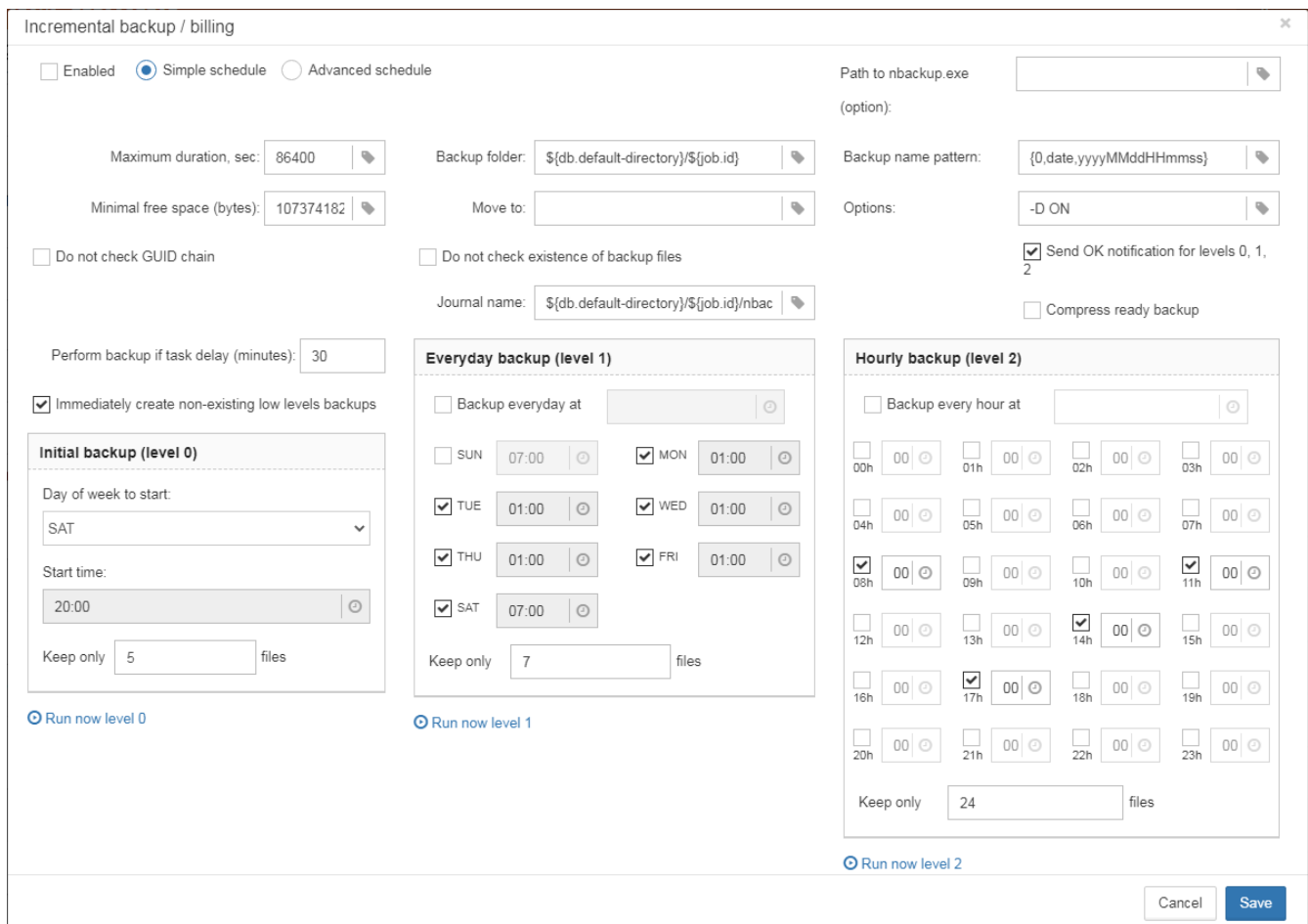
Latest backup

Level0: 20160408_022212+20160413_050212.nb1 (112.9 Gb)

Level1: 20160408_022212+20160413_050212.nb1 (2.9 Gb)

Level2: not yet done

In order to setup Simple incremental backup, click on Settings “gear” of the widget and select “Simple schedule” (selected by default). The following dialog will appear:



Incremental backup / billing

Enabled Simple schedule Advanced schedule

Path to nbackup.exe

(option):

Maximum duration, sec:

Backup folder:

Backup name pattern:

Minimal free space (bytes):

Move to:

Options:

Do not check GUID chain Do not check existence of backup files

Journal name:

Send OK notification for levels 0, 1, 2

Compress ready backup

Perform backup if task delay (minutes):

Immediately create non-existing low levels backups

Initial backup (level 0)

Day of week to start:
SAT

Start time:
20:00

Keep only files

Run now level 0

Everyday backup (level 1)

Backup everyday at

SUN 07:00 MON 01:00

TUE 01:00 WED 01:00

THU 01:00 FRI 01:00

SAT 07:00

Keep only files

Run now level 1

Hourly backup (level 2)

Backup every hour at

00h 01h 02h 03h

04h 05h 06h 07h

08h 09h 10h 11h

12h 13h 14h 15h

16h 17h 18h 19h

20h 21h 22h 23h

Keep only files

Run now level 2

Cancel Save

There are 4 main areas in this dialog, let's cover them one by one.

The top area is devoted for general settings of the incremental backup – they are the same for

Simple and Advanced schedules:

Incremental backup / billing

Enabled Simple schedule Advanced schedule

Path to nbackup.exe

(option):

Max duration, sec – it limits the maximum duration of backup process, by default is 1 day (86400 seconds).

Minimum free disk space (bytes) – minimal size of free disk space to prevent backup to start, by default ~9Mb

Backup folder – where incremental backup for the selected database will be stored. It is necessary to store incremental backups for each database separately from backups of other databases: i.e., the separate folder for each database.

It is necessary to specify backup folder with enough free disk space to store all level of backups!

Journal name–file name details information about incremental backups files, for internal use only.

Path to nBackup – it is possible to specify other nbackup tool than standard *nbackup* (not recommended).

Backup name pattern – pattern for files of incremental backup (no need to change it).

Options – additional options for nbackup command line tool (no need to change it).

Do not check existence of backup files – this option should be checked if you plan to delete or more incremental backups to another location.

Do not check GUID chain – this option should be checked if you want to skip existence check of previous levels of incremental backups.

Immediately create non-existing low-level backups – by default this option is *On*. It means that if you have scheduled the initial start moment of level 1 backup earlier than the initial start moment of level 0 backup, DataGuard will automatically fix it and create level 0 backup right before level 1. The following backups of level 0 will be fulfilled according the regular schedule.

Send OK email for levels 0, 1, 2 – enable this option to receive notifications about incremental backups (*highly recommended!*)

After setting main set of parameters the schedule itself should be set. As you can see on the screenshot below, you need to specify day of the week and time to do level 0 (weekly) backup, days of week and time to start level 1 (daily) backups and hours and minutes of level 3 (hourly backups).

For each backup level you can specify how many files to keep in history.

Incremental backup / dbmaster

Enabled Simple schedule Advanced schedule

Path to nbackup.exe

(option):

Maximum duration, sec: Backup folder: Backup name pattern:

Minimal free space (bytes): Move to: Options:

Do not check GUID chain Do not check existence of backup files Send OK email for levels 0, 1, 2

Journal name: Compress ready backup

Perform backup if task delay (minutes):

Immediately create non-existing low levels backups

Initial backup (level 0)

Day of week to start:

Start time:

Keep only files

Everyday backup (level 1)

Backup everyday at

SUN 07:00 MON 01:00

TUE 01:00 WED 01:00

THU 01:00 FRI 01:00

SAT 07:00

Keep only files

Hourly backup (level 2)

Backup every hour at

00h 00 01h 00 02h 00 03h 00

04h 00 05h 00 06h 00 07h 00

08h 00 09h 00 10h 00 11h 00

12h 00 13h 00 14h 00 15h 00

16h 00 17h 00 18h 00 19h 00

20h 00 21h 00 22h 00 23h 00

Keep only files

Cancel Save

By default it is set to keep 5 weekly backups, 7 daily and 24 hourly backups.

However, sometimes more flexible schedule is required, for this purpose Incremental Backup widget has Advanced schedule:

Incremental backup / billing

Enabled Simple schedule Advanced schedule

Path to nbackup.exe

(option):

Maximum duration, sec: Backup folder: Backup name pattern:

Minimal free space (bytes): Move to: Options:

Do not check GUID chain Do not check existence of backup files Send OK notification for levels 0, 1, 2

Journal name: Compress ready backup

Level 0

Enabled

Start at CRON:

Keep only files

[Run now level 0](#)

Level 1

Enabled

Start at CRON:

Keep only files

[Run now level 1](#)

Level 2

Enabled

Start at CRON:

Keep only files

[Run now level 2](#)

Level 3

Enabled

Start at CRON:

Keep only files

[Run now level 3](#)

Level 4

Enabled

Start at CRON:

Keep only files

[Run now level 4](#)

Cancel Save

As you can see, the upper part of the configuration screen is the same as in Simple schedule, and the difference is in the way how backup levels are scheduled.

Advanced schedule allows to setup up to 5 levels of backup, and plan them with flexible [CRON](#)

expressions.

For example, you can setup it to create full copy (level 0) backup every 3 months, level 1 copy every month, level 2 – every week, level 3 every day and level 4 – every hour.



If you are monitoring more than one database, it is highly recommended splitting the runtime of the backups.

3.2.15. Database: Dump Backup

This job also utilizes nbackup functionality in Firebird, but unlike multilevel backups, it always performs a full copy (level 0) of the database. Such job is useful to quickly create a copy of working database.

The configuration of Database: Dump backup is trivial:

Dump backup / test
✕

Enabled

0 0 23 ? * MON-SUN

Minimal free space (bytes):

10000000

Backup folder:

\${db.default-directory}/\${job.id}

Backup name:

nbackup_copy_{0,date,yyyyMMdd_HH-mm}

Maximum numbers of *.nbackupcopy files in folder to keep:

1

Copy type:

Lock with SQL command

Send OK email

Cancel




Save

You just need to setup when and where DataGuard should copy a full copy (level 0 incremental backup), and how many copies it should keep.

3.2.16. Database: RestoreDB

One of the often tasks of the database administrators is restoring database from the backup. There could be many reason to do restore, the most popular reasons are regular check of the stored backups and necessity to have fresh restored copy for quick rollback. HQbird FBDataGuard can automate restoring of backups (which were created with gbak or Database: Verified backup) with **Database: RestoreDB** job. Let's consider the options and parameters of this job.

74

 **RestoreDB**  

RestoreDB: OK [Last run: 21 min 59 sec ago]
Recent restored database: H:\TEMP\ba...strestore.fdb(8.5 Gb)
Restored from backup: H:\TEMP\ba...418_14-25.fbk(18/04/2016
14:25)
Restore time: 8 min 51 sec

By default, restore is disabled – and, since restoring can be long and resource-consuming job, please plan when to restore carefully.

The database can be restored from different types of backups. To specify which types of backups are used during recovery, use the **Restore Source** switch.

Below you can see the configuration dialog for **Database: RestoreDB** in **nbackup** mode:

Restore Database / billing
✕

Enabled

Scheduled:

Get backup from folder:

Take backup not older than, hours:

Restore source: nbackup gbak

Datetime pattern for nbackup:

Keep files from recent days (NN):

Restore to directory:

Restore with file name:

When existing database found: Replace existing file Rename existing file

Append suffix to database name:

Execute command after restore:

Restore timeout, minutes:

Check available space before restore. Minimum value (bytes):

Notify on successful restore

In **gbak** mode, the configuration dialog for **Database: RestoreDB** looks like this:

Restore Database / billing
✕

Enabled

Scheduled

Get backup from folder

Take backup not older than, hours

Restore source: nbackup gbak

Use NN CPU cores to restore:

Restore options

Template for gbak backup file name

Backup gbak file extension

Take date of gbak backup from File name File date

Keep files from recent days (NN)

Keep only recent files (NN)

Restore to directory

Restore with file name

When existing database found Replace existing file Rename existing file

Append suffix to database name

Execute command after restore

Restore timeout, minutes:

Check available space before restore. Minimum value (bytes)

Notify on successful restore

- “**Scheduled**” field contains [CRON expression](#) which defines when to run restore.
- “**Get backup from folder**” - specify the location of backup file(s) to be restored. If you are restoring backups at the same computer where they have been created, specify the same folder

as it is in Database: Verified backup job. If you are restoring backups from the another computer, specify the folder where those backups are located.

- **“Take backup not older than, hours”** - this parameter specifies the maximum age of backup to be restored. If the latest backup file will be older than specified number of hours, RestoreDB job will send the alert with warning that backup is too old. This is useful for automatic checking of backups created on the remote computer.
- **“Restore source”** specifies what types of backups will be used to restore the database .
- **“Datatime pattern for nbackup”** contains the template for backup names made with nbackup. It should be the same as **Backup name pattern** see [Database: Incremental Backup](#).
- **“Template for gbak backup file name”** contains the template for backup names. It should be the same as **Backup name pattern** see [Verified backup](#).
- **“Backup gbak file extension”** - by default it is fbk
- **“Use NN CPU cores to restore”** - only available in gbak mode.
- **“Restore options”** - only available in gbak mode.
- **“Restore to directory”** - folder where FBDataGuard will restore backups.
- **“Restore with filename”** - template for the restored database file. By default it contains the following parts
 - `${db.id}_{0,date, yyyyMMdd_HH-mm}_testrestore.fdb`
 - Db.id – internal identifier of the database (GUID)
 - 0,date, yyyyMMdd_HH-mm – timestamp
 - testrestore.fdb – description (You can set there any filename you need).
- **“When existing database found”** - if FBDataGuard will encounter a file with the same name as restored database in the destination folder, by default it will rename the existing file. If you want to replace old restored file with new one, choose “Replace existing file”.
- **“Append suffix to filename when rename”** - if you have chosen “Rename existing file”, this suffix will be used to rename it. If you have chosen “Replace existing file”, this suffix also will be used to rename, but after that the old file will be deleted.
- **“Execute command after restore”** - in this field you can specify an optional path to the command file or another utility to be started after the restore. There will be 2 parameters passed: the first is the path to the backup which was just restored, and the second is the path to the restored file.
- **“Restore timeout, minutes”** - here you can set the time limit for restore operation. If this limit will be exceeded, the warning will be sent, saying that restore takes too long.
- **“Check available space before restore (bytes)”** - here you can set the limit for the minimal free space in the restore destination – if there is less free space than specified, restore will not start, and associated warning will be sent.
- **“Notify on successful restore”** - send email about successful restore (by default it is off, only alerts about problems will be sent).

3.2.17. Database: Transfer Replication Segments

The purpose of "Transfer Replication Segments" job is to send replication segments produced by async replication from master to replica server. In the case of distributed environment of the asynchronous replication, when the network connection between master and replica server is unstable, or with high latency, or when servers are in the different geographical regions, the best way to transfer replication segments will be through FTP or FTP over SSH.

Below we will consider how to setup Cloud Backup for this task.

First, the asynchronous replication master should be configured to save replication segments into the some local folder – by default, it will be `${db.path}.LogArch` – as it is shown in the example below:

Database replication configuration: "billing" ✕

Dataguard ID: 9b6e6443-de7e-428c-af75-a7b9b520f533

Replication role	<input type="radio"/> Off <input checked="" type="radio"/> Master <input type="radio"/> Replica
Working mode	<input type="radio"/> Synchronous <input checked="" type="radio"/> Asynchronous
Write committed data every NN seconds	<input type="text" value="90"/>
Log directory	<input style="border: 1px solid #ccc;" type="text" value="\${db.path}.ReplLog"/> 🗑
Log archive directory	<input style="border: 1px solid #ccc;" type="text" value="\${db.path}.LogArch"/> 🗑
Override log archive command	<input style="border: 1px solid #ccc;" type="text"/> 🗑
Optional parameters	<input style="border: 1px solid #ccc; width: 100%; height: 40px;" type="text" value="exclude_without_pk=true
journal_segment_count=64"/> <div style="margin-top: 5px;"> <p style="background-color: #2c5e8c; color: white; padding: 2px 5px; display: inline-block;">Find and exclude tables without Primary or Unique keys</p> <p style="background-color: #2c5e8c; color: white; padding: 2px 5px; display: inline-block; margin-top: 5px;"><<less</p> <p style="background-color: #2c5e8c; color: white; padding: 2px 5px; display: inline-block; margin-top: 5px;">Reinitialize replica database</p> <p style="background-color: #2c5e8c; color: white; padding: 2px 5px; display: inline-block; margin-top: 5px;">See initialization status</p> <p style="background-color: #2c5e8c; color: white; padding: 2px 5px; display: inline-block; margin-top: 5px;">Enable publications (and grant it for all tables)</p> </div>

Transfer Replication Segments / billing
✕

Enable/Disable

Check period, seconds

Monitor this folder

🗑

Filename template

🗑

Compress segments

🗑

Where to upload

#	Type	Server:Port	User	Path		
1	Disabled				🔴	⚙
2	Disabled				🔴	⚙
3	Disabled				🔴	⚙
4	Disabled				🔴	⚙
5	Disabled				🔴	⚙

Failed connection attempts to disable FTP (nodes 2-5)

🗑

Delete local prepared file copy

How many old (sent) files to keep

🗑

Send Ok report

Name prefix to rename uploaded reini files

🗑

Figure 38. Transfer Replication Segments configuration

Then we can setup **Transfer Replication Segments** job to monitor this folder for the new replication segments and upload them to the remote FTP server.

As you can see at the screenshot above, Cloud backup job checks folder, specified in “**Monitor this folder**” with an interval, specified in “**Check period, seconds**”. Please note – Cloud Backup sends files in the order of their names, not dates.

To check that transferred files are valid replication segments, and to support automatic re-

80

initialization of the replica databases, the checkmark “**Enable/disable replication cloud backup job**” must be enabled.

By default, Cloud Backup compresses and encrypts replication segments before send them. The default password is “**zipmasterkey**” (without quotes), which can be specified in the field “**Compress with optional password**”. FBDataGuard creates the compressed and encrypted copy of the replication segment and upload it to the specified target server.

To disable packing and encryption, uncheck the “**Compress with optional password**” checkmark.

FTP/FTPS/FTPS over SSH

There are several types of target servers: FTP, FTP over SSL/TLS, FTP over SSH. When you select the necessary type, dialog shows mandatory fields to be completed.

You can select up to 5 simultaneous remote servers to upload backups. Below you can see the configuration dialog for FTP.

Transfer Replication Segments / billing / FTP 1

Upload to FTP

Upload to FTP FTP FTP over SSL/TLS FTP over SSH

FTP Server

FTP Port

FTP User

FTP Password

Use passive mode

Upload to folder

Existing files will be overwritten!

Cancel Save



If you don't have FTP installed on the target server with Windows, install Filezilla – it is very popular fast and lightweight FTP-server for Windows.



Replication segments will be uploaded to the subdirectory specified in the “Upload to folder”. By default, this is `/dababase0/${db.id}`, where `db.id` is the identifier of the database inside the DataGuard. The replica about this `db.id` does not know anything, so you need to register it manually in “Unpack to directory” (see [File Receiver](#)).

FTP over SSL/TLS

Transfer Replication Segments / billing / FTP 1
✕

Upload to FTP

Upload to FTP FTP FTP over SSL/TLS FTP over SSH

Key store file

Key store password

Implicit

FTP Server

FTP Port

FTP User

FTP Password

Use passive mode

Upload to folder

Existing files will be overwritten!

In order to send files to FTPS, it is necessary to create jks storage with private key file, and specify path to it in the field “Key store file” and password for it in “Key store password”.

See details and example how to create jks file and password here: <http://xacmlinfo.org/2014/06/13/how-to-keystore-creating-jks-file-from-existing-private-key-and-certificate/>

The last part of parameters in Cloud Backup dialog allows controlling the behavior of Cloud backup.

- **Delete local prepared copy** – by default it is On. This parameter specify that Cloud backup job deletes compressed copy of the replication segment after the successful upload to the target server. If you don’t want to keep these copies on the master server, keep the parameter enabled.
- **Delete local prepared file copy** – by default is Off. It means status means that replication segment will be not deleted by FBDataGuard after uploading. It can be useful if you want to keep the full history of changes in replication segments, but, be careful; in case of an intensive write activity replication segments can occupy a lot of space (Terabytes).
- **Send Ok report** – send email to the specified in Alerts address every time when replication segment is uploaded. By default it is off.

As a result, FBDataGuard will upload encrypted and compressed replication segments to the remote server. To decompress and decrypt them into the regular replication segments, another instance of HQbird FBDataGuard should be installed on the replica server, and Cloud Backup Receiver job should be configured – see more details in the section [Database: File Receiver](#).

FTP over SSH

Transfer Replication Segments / billing / FTP 1

Upload to FTP

Upload to FTP FTP FTP over SSL/TLS FTP over SSH

Key store file

FTP Server

FTP Port

FTP User

FTP Password

Upload to folder

Existing files will be overwritten!

Cancel Save

To use FTP over SSH with private key authentication, please specify the full path to it in “Key store file”, other parameters are similar to usual FTP.

3.2.18. Database: Transfer Files

The purpose of "Transfer Files" job is to send backup files from master to replica server. In the case of distributed environment, when the network connection between master and replica server is unstable, or with high latency, or when servers are in the different geographical regions, the best way to transfer files will be through FTP or FTP over SSH.

Below we will consider how to setup "Transfer Files" for this task.

First, the database server should be configured to save backup files into the some local folder — by default, it will be `#{db.default-directory}/backup` — as it is shown in the example below:

Transfer Files / billing
✕

Enable/Disable cloudbackup job

Activate cron expression

Monitor this folder

Filename template

Compression level

Encrypt when compressing

Where to upload

#	Type	Server:Port	User	Path		
1	Disabled				⚠	⚙
2	Disabled				⚠	⚙
3	Disabled				⚠	⚙
4	Disabled				⚠	⚙
5	Disabled				⚠	⚙

Failed connection attempts to disable FTP (nodes 2-5)

Delete local prepared file copy

How many old (sent) files to keep

Send Ok report

Perform fresh backup

Figure 39. Transfer File configuration

Then we can setup **Transfer Files** job to monitor this folder for the new backup files and upload them to the remote FTP server.

As you can see at the screenshot above, **Transfer Files** job checks folder, specified in “Monitor this folder” with an interval, specified in “**Check period, seconds**”. Please note – Transfer Files sends files in the order of their names, not dates.

By default, Transfer Files compresses and encrypts backup files before send them. The default password is “**zipmasterkey**” (without quotes), which can be specified in the field “**Encrypt when compressing**”. FBDataGuard creates the compressed and encrypted copy of the backup and upload it to the specified target server.

To disable packing and encryption, uncheck the “**Encrypt when compressing**” checkmark.

FTP/FTPS/FTPS over SSH

There are several types of target servers: FTP, FTP over SSL/TLS, FTP over SSH. When you select the necessary type, dialog shows mandatory fields to be completed.

You can select up to 5 simultaneous remote servers to upload backups. Below you can see the configuration dialog for FTP.

Transfer Files / dbmaster / FTP 1

Upload to FTP

Upload to FTP FTP FTP over SSL/TLS FTP over SSH

FTP Server

FTP Port

FTP User

FTP Password

Use passive mode

Upload to folder

Existing files will be overwritten!

Cancel Save



If you don't have FTP installed on the target server with Windows, install Filezilla – it is very popular fast and lightweight FTP-server for Windows.



Replication segments will be uploaded to the subdirectory specified in the “Upload to folder”. By default, this is */dababase0/\${db.id}*, where *db.id* is the identifier of the database inside the DataGuard. The replica about this *db.id* does not know anything, so you need to register it manually in “Unpack to directory” (see [File Receiver](#)).

FTP over SSL/TLS

Transfer Files / dbmaster / FTP 1

Upload to FTP

Upload to FTP FTP FTP over SSL/TLS FTP over SSH

FTP Server

FTP Port

FTP User

FTP Password

Use passive mode

Upload to folder

Existing files will be overwritten!

Cancel Save

In order to send files to FTPS, it is necessary to create jks storage with private key file, and specify path to it in the field “Key store file” and password for it in “Key store password”.

See details and example how to create jks file and password here: <http://xacmlinfo.org/2014/06/13/how-to-keystore-creating-jks-file-from-existing-private-key-and-certificate/>

The last part of parameters in Cloud Backup dialog allows controlling the behavior of Cloud backup.

- **Delete local prepared copy**— by default it is On. This parameter specifies that Transfer Files job deletes compressed copy of the file after the successful upload to the target server. If you don’t want to keep these copies on the master server, keep the parameter enabled.
- **Delete local prepared file copy**— by default is Off. It means status means that file will be not deleted by FBDataGuard after uploading. It can be useful if you want to keep the full history of changes in files, but, be careful; in case of an intensive write activity such files can occupy a lot of space (Terabytes).
- **Send Ok report**— send email to the specified in Alerts address every time when replication segment is uploaded. By default, it is off.
- **Perform fresh backup**— disabled by default. Transfer Files remembers the last number of file it sends. If you need to start again from scratch, from file 1, enable this parameter. Please note that it will automatically become disabled after the resetting of the counter.

As a result, FBDataGuard will upload encrypted and compressed files to the remote server. To decompress and decrypt them into the regular files, another instance of HQbird FBDataGuard should be installed on the replica server, and File Receiver job should be configured — see more

details in the section [Database: File Receiver](#).

FTP over SSH

Transfer Files / dbmaster / FTP 1 ✕

Upload to FTP

Upload to FTP
 FTP
 FTP over SSL/TLS
 FTP over SSH

Key store file 🗑️

FTP Server 🗑️

FTP Port 🗑️

FTP User 🗑️

FTP Password 🗑️

Upload to folder 🗑️

Existing files will be overwritten!

Cancel
Save

To use FTP over SSH with private key authentication, please specify the full path to it in “Key store file”, other parameters are similar to usual FTP.

Sending verified and incremental backups through Cloud Backups

Cloud Backup also can be used to send any files to FTP/FTPS/etc. For example, you can setup Cloud Backup to look for FBK files, produces by Verified Backup Job, and schedule to upload to the remote FTP server.

It is necessary to remember that number of stored backups should be less than the number of files to be preserved by Cloud Backup (specified in the parameter “How many files to keep”. By default, Cloud Backup keeps 10 last sent files, and Verified backup has 5 most recent backup files, so it work Ok, but if you will reduce the number of kept files in Cloud Backup, it will delete extra files according “Filename template”.

The same can be done for incremental backups.

3.2.19. Database: Pump Files

The purpose of the "Pump Files" task is to transfer files from one directory accessible to the DataGuard to some other location, usually remote, with the possibility of using various methods that can be connected to the DataGuard in the form of plugins and selectable in the task

configuration with the ability to set unique for each plugin parameters. HQbird includes two file transfer plugins: fpt and sftp. There are other file transfer plugins. Transfer plugins are jar files and are located in the Firebird DataGuard/plugins folder.

Let's consider the options and parameters of this job.

The screenshot shows a configuration window titled "Pump Files / billing". It contains the following settings:

- Enable/Disable File-Pump job
- CRON: 0 0/5 * ? * *
- Watch for files in folder: \${db.default-directory}/backup
- Filemask to pump: *.fbk;*.zbk
- Exclude file-mask: temp*.*;*.fuploaded
- Compression level: NORMAL
- Encrypt files when packing (password field is obscured with dots)
- Pump method: Send to ftp
- FTP Server: 127.0.0.1
- FTP Port: 8721
- Login: admin2
- Password: strong password2
- Remote dir: /dataguard/pumped/
- Passive Mode
- Keep NN files: 10
- Send OK-report on every pump

Buttons: Cancel, Save

Figure 40. Options available for the ftp file transfer plugin.

- **Filemask to pump**—whitelist, according to which files are selected for copying. Represent masks of file names. Must be separated by comma.

- **Exclude file-mask**—blacklist is a mask of file names that should be excluded from the transfer. The blacklist takes precedence over the whitelist.
- **Pump method**— file transfer method (plugin).

Pump Files / billing
✕

Enable/Disable File-Pump job

CRON

Watch for files in folder:

Filemask to pump

Exclude file-mask

Compression level

Encrypt files when packing

Pump method

SFTP Server

SFTP Port

Login

Password

Remote dir

Optional JKS File

Keep NN files:

Send OK-report on every pump

Figure 41. Options available for the sftp file transfer plugin.

The algorithm of this task is as follows:

1. At each iteration of the task, a list of files is generated for the directory for monitoring files to be sent. Masks are used to select the list of files: "Filemask to pump" and "Exclude file-mask".

2. For each selected file (from the list from step 1, in ascending date order from the lastModified file), the following is performed:
 - a. If the packing option is set, the file is packed (if the file is not of zero size). The name of the packed file is formed by adding a hardcoded extension: `.zipfilepump`. The file is packed in the same directory. If the file turns out to be of zero size, the algorithm will consider that the file has not been completed yet and will interrupt sending the rest of the files with a corresponding message.
 - b. The file sending task is configured for one of several possible sending options using optional plugins (see below). Depending on whether the packing option was enabled or not, the original or packed file is sent using the specified algorithm (in the current version it is ftp or sftp).
 - c. After sending, if the packing option was selected, the packed file is deleted.
 - d. The original file is renamed by adding the extension `.fuploaded`.
3. The algorithm proceeds to send the next file from the list. The total number of files sent during the iteration and their original (unpacked) size are summed up for display in the widget
4. Upon completion of sending all files from the generated list, the directory is revolving cleaned, from which files are deleted by mask `*.fuploaded`. That is, a list of all such files is created, it is sorted by the time of the last modification, and all old ones are deleted, except for the last "Keep NN files".

Upon completion of sending, if the "Send OK-report on every pump" checkbox is checked, then the user will be sent a report on the number and size of files sent at the current iteration.

3.2.20. Database: File Receiver

In general, Cloud Backup Receiver is designed to decompress files from zip archives, and the most often it is used in the pair with Cloud Backup to transfer archived replication segments.

Cloud Backup Receiver checks files in the folder specified in "**Monitor directory**", with interval equal to "**Check periods, minutes**". Its checks only files with specified mask according "**Filename template**" (**arch** by default) and specified extension (*.replpacked* by default), and if it encounters such files, it decompresses and decrypts them with the password, specified in "**Decrypt password**", and copies to the folder, specified in "**Unpack to directory**".

If parameter "**Monitor for replication**" is enabled, Cloud Backup Receiver also will check that received file is actually a replication segment (it has specific header), and if it is not, it will raise an appropriate warning.

File Receiver / billing
✕

Enabled

Check period, seconds

15

Watch for incoming files in

\${default-output-directory}/FTP/master001
 🗑

Unpack to folder

\${db.repparam_log_archive_directory}
 🗑

Filename template

.journal

🗑

Extension for packed files

.repacked

🗑

Decrypt password

.....
 🗑

Alert if number of unpacked files more than

30

🗑

Warn if the newest file in unpack folder is older than (minutes):

360

🗑

Send Ok report

Monitor for reinitialization

Prefix for incoming reini- files

reinidb_

🗑

Cancel

Save

There are the following additional parameters:

- **Alert if number of unpacked files more than** – by default is 30. If there is a long queue of replication segments to be unpacked, it can be a problem with a replica database, so HQbird sends alert to attract administrator's attention.
- **Warn if the newest file in unpack folder is older than (minutes)** – if the most recent file (usually, replication segment) is too old (more than 360 minutes), the replication process can be broken, and HQbird sends an appropriate alert.
- **Send Ok report** – by default it is Off. If it is On, HQbird sends an email about each successful unpacking of the segment. It can be too often for replication segments, because they are arriving every 30-180 seconds, and Ok for normal files like verified or incremental backups.
- **Perform fresh unpack** – disabled by default. Cloud Backup Receiver remembers the last number of replication segment it unpacked. If you need to start unpacking from scratch, from segment 1 (for example, after re-initialization of replication), enable this parameter. Please note that it will automatically become disabled after the resetting of the counter.

After setup of Cloud Backup Receiver, configure the replica to look for replication segments: set in the “Log archive directory” the same path as in “Cloud Backup Receiver” → “Unpack to directory”.

Database replication configuration: "billing" ✕

Dataguard ID: 9b6e6443-de7e-428c-af75-a7b9b520f533

Replication role Off Master Replica

Working mode Synchronous Asynchronous

Log archive directory 🗑

Source database GUID (optional) 🗑

Optional parameters

Verbose

<<less

Close Save

Embedded FTP server

HQbird has embedded FTP server, which is off by default. It is suitable to use embedded FTP server to receive replication segments.

In order to enable embedded FTP server, it is necessary to edit the *ftpsrv.properties* configuration file, which is located in *C:\HQbirdData\config* or */opt/hqbird/ftpsrv.properties*

By default, it contains the following:

```
#path in ftpsrv.homedir must be escaped "ftpsrv.homedir=c:\\ftp\\pub"
# or backslashed for ex: "ftpsrv.homedir=c:/ftp/pub"

ftpsrv.enable = false

ftpsrv.port = 8721

ftpsrv.defuser=admin2

ftpsrv.defpsw=strong password2

ftpsrv.homedir=
```

It is necessary to change **ftpsrv.enabled** to **true** and specify the home directory for FTP in

ftpsrv.homedir parameter. Also, it is recommended to use non-default username and password.

After that, restart FBDataGuard service, and check availability of the FTP.



Attention — Linux users!

On the Linux, FBDataGuard service runs under **firebird** user, so FTP home directory also should have permission for user **firebird**.

3.2.21. Database: Low-level metadata backup

“Database: Low level metadata backup” is one of the key jobs of DataGuard, it ensures database protection at low level.

First of all, this job stores raw metadata in special repository, so in case of heavy corruption (due to hardware failure, for example) of database it is possible to use this repository to recover database.

The second purpose of this job is to constantly check all important system tables for consistency. Every 20 minutes it walks through all important system tables in the database and ensures that there are no errors at metadata level.

The third purpose is to warn administrator about too many formats for each tables.

There is an implementation limit in Firebird to have 256 formats per table, however even several formats can greatly increase a chance of hard corruption and can slow down the performance. It is recommended do not change tables structure at production database and keep only one format per each table. If it’s not possible, administrator should try to perform backup/restore more often to transform all formats into the single one.

Database metadata backup configuration / test ✕

Enabled

Check period, minutes:

Store metadata in:

Date format for folder name:

Folder name prefix:

Max formats:

3.2.22. Database: Validate DB

Validation of Firebird database requires exclusive access: i.e., no users should be connected during

validation. “Database: Validate DB” job shuts down the database and performs validation of database, and then turns it on.

By default, this job is OFF. Please consider carefully, is it possible to provide exclusive access for database. Validation can also take significant time.

Update validation configuration / test

Enabled

Schedule: 0 0 23 ? * MON-FRI

Shutdown timeout, sec: 10

Shutdown mode: FORCE

Cancel Save

Using configuration dialog, you can enable/disable this job, set time to run, set the shutdown timeout (time to wait before launch validation), and also shutdown mode (FORCE, ATTACH, TRANSNATIONAL). If you have no deep knowledge n what you are doing, it’s better to keep default parameters.

“Database: Validate DB” will send alert with critical status if there will be any errors.

Also, Firebird will write errors into *firebird.log*, and they will appear in the alerts generated by “Server log” job.

3.2.23. Database: Sweep Schedule

FBDataGuard includes special job to run an explicit sweep, in case if automatic sweep was disabled. By default, job is disabled.

Sweep schedule

Sweep schedule: UNKNOWN [scheduled 0 0 23 ? * MON-SUN]

The recommendation is to schedule explicit sweep with disconnection of long-running transactions for all databases where such transactions are detected. The recommended period is once per day (usually during the night, after backup’s completing).

By default, sweep is set to 23-00, which can be not a good time, because default verified backup starts at the same time, so better change it.

Sweep schedule / test
✕

Enabled

Sweep is scheduled to start at this time

Executable

Parameters

Disconnect connections with long-running active transactions before sweep

Do not disconnect processes with name pattern

Disconnect processes older than

Use NN CPU cores to sweep:

0 0 23 ? * MON-SUN

gfix.exe

-sweep

%(\|/)(fbsvcmgr|gstat|gfix|gbak)(.exe){0,1}

30

N/A

Cancel

Save

Please note: by default, check mark “**Disconnect all connections with long-running active transactions before sweep**” is enabled. It means that HQbird will find and disconnect long-running transactions (more than 30 minutes) before sweep — in order to make sweep efficient. If long-running active transactions will be not disconnected, sweep cannot clean old records versions.

“**Do not disconnect processes with name pattern**”—in this parameter specify SIMILAR TO expression for processes names which will be not disconnected. By default, we exclude gbak, gstat and fbsvcmgr processes.

“**Disconnect all processes older than (minutes)**”—HQbird will disconnect processes which have long-running active writeable transactions, by default threshold is 30 minutes. The practical upper limit for this parameter is 1440 minutes (it is highly unlikely that transaction does something useful more than 1 day).

“**Use multiple cores to sweep**”—HQbird Enterprise can use multiple cores to perform sweep operation, in order to make sweep 4-6 times faster. We recommend to specify no more than 1/2 CPU cores in case of the single database on the server, or 1/4 of CPU cores if there are several databases. For example, if you have 16 cores and 1 big database, set this parameter to 8, if there are several big databases, set 4.

3.2.24. Database: Delta

If you are using incremental backups (or Dump backup), this job is critically important. It watches for delta-files lifetime and size, and warns if something goes wrong. Forgotten delta-files are the often reason of corruptions and significant losses of data.

This jobs finds all delta files associated with database and check their age and size. If one of these parameters exceeds thresholds “Maximum delta size” or “Maximum delta age”, administrator will receive the alert and database status will be set to CRITICAL.



If delta file of the protected database was corrupted, it is possible to extract data from it using metadata from the original database file or repository from “Low-level metadata backup” job.

Delta file monitoring / test
✕

Enabled

Check period, minutes:

5

Maximum delta size, bytes:

52428800

Maximum delta age, minutes:

360

Cancel

Save

3.2.25. Database: Disk space

This job watches for all objects related with database: database files (including volumes of multi-volume database), delta-files, backup files and so on.

“Database: Disk space” job analyzes the growth of database and estimate will there be enough free space for the next operation like backup (including test restore) on the specific hard drive.

It generates several types of alerts. Problems with disk space are in the top list of corruption reasons, so please pay attention to the alerts from this job.

This job also contributes data to the server space analysis graph ().

By default, this job is enabled.

Using configuration dialog, you can specify check period and thresholds for free space. The first reached threshold will be alerted. To set threshold only in % of disk space, you need to set explicit space in bytes to 0.

Space monitoring / test
✕

Enabled:

Check period, minutes:

Free space minimum, %:

Free space minimum, bytes:

3.2.26. Database: Database statistics

This job is very useful to capture performance problems and perform overall check of database at low-level without making backup.

Database statistics configuration / test
✕

Enabled

Schedule:

Store statistics in:

Statistics archive depth:

Statistics file name pattern:

We recommend running this job every day and storing a history of statistics report.

Then, with HQbird Database IBAnalyst it is possible to find problems with database performance and get useful recommendations how to fix them.



As a useful side effect, gstat visits all database pages for tables and indices, and ensures that all of them are correct.

3.2.27. Database: Replica Check

This task allows you to check the availability of the replica database. After a specified period, it changes the value of the specified generator and compares the value of the generator on the replica side and the master database.

Replica Check / billing
✕

Enabled

Check period	10
Master generator name	EMP_NO_GEN 🗑
Replica generator name	🗑
Min diff to alert	1000
Replica server name	127.0.0.1 🗑
Replica firebird server port	3054
Replica database path	replicadb 🗑
Replica username to connect	SYSDBA 🗑
Replica password	masterkey 🗑
Cypher key name	🗑
Cypher key value	🗑

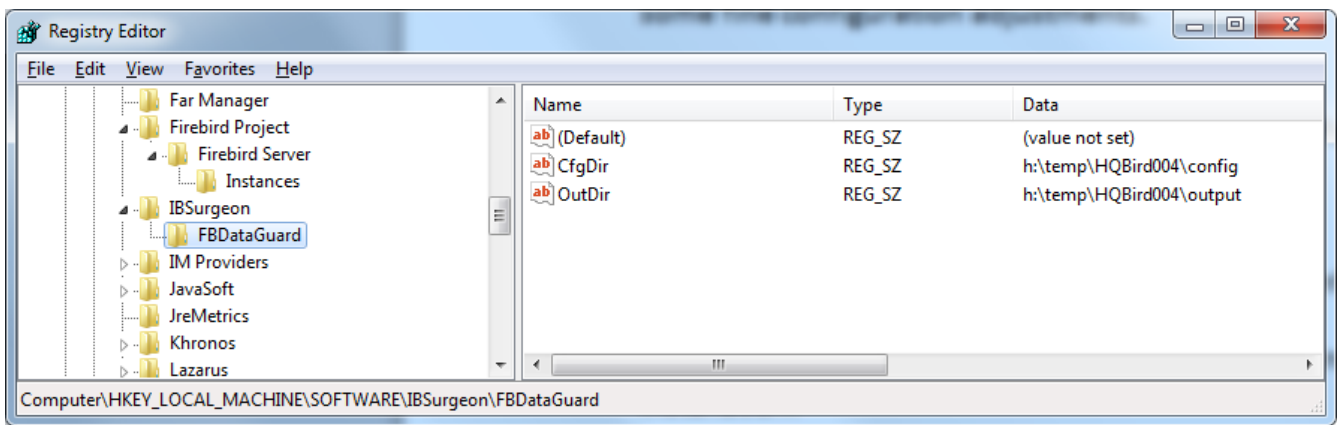
Min diff to alert— the difference between the values of the generator on the master and replica side, after which alter are sent.

3.3. FBDataGuard tips&tricks

FBDataGuard allows changing its setting not only through web-console, but also using direct modification of configuration files. This can be useful when you need to install FBDataGuard in silent mode (no interaction with user), to bundle it with third-party software, or to perform some fine configuration adjustments.

3.3.1. Path to FBDataGuard configuration

During the start FBDataGuard looks for in registry for configuration and output paths:



These values specify the paths to FBDataGuard configuration and output folder — these values are chosen during installation.

3.3.2. Adjusting web-console port

One of the most frequently asked questions is how to adjust port for web-console application (by default it is 8082), It can be done by changing port setting in file `%config%\agent\agent.properties` (`%config%` is `C:\HQbirdData\config` or `/opt/hqbird/conf`).

```
server.port = 8082 #change it
```

`%config%` - folder to store configuration information, it is specified in .

3.3.3. How to change password for Admin user

You can specify its password in the file `access.properties` (in `C:\HQbirdData\config` or `/opt/hqbird/conf`)

```
access.login=admin
```

```
access.password=youradminpasswordforhqbird
```

After setting the password, restart FBDataGuard, and new password will be encrypted and applied.

3.3.4. Guest user for HQbird FBDataGuard

There is read-only user to access HQbird FBDataGuard, with the name guest.

```
access.guest-login=guest
```

```
access.guest-password=yournewpassword
```

3.4. Appendix: CRON Expressions

All jobs in FBDataGuard have time settings in CRON format. CRON is very easy and powerful format to schedule execution times.

3.4.1. CRON Format

A CRON expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Seconds	YES	0-59	, - * /
Minutes	YES	0-59	, - * /
Hours	YES	0-23	, - * /
Day of month	YES	1-31	, - * / L W
Month	YES	1-12 or JAN-DEC	, - * /
Day of week	YES	1-7 or SUN-SAT	, - * / L #
Year	NO	empty, 1970-2099	, - * /

So cron expressions can be as simple as this: `* * * * ? *` or more complex, like this: `0 0/5 14,18,3-39,52 ? JAN,MAR,SEP MON-FRI 2002-2010`

3.4.2. Special characters

- `*` ("all values")—used to select all values within a field. For example, `"*` in the minute field means "every minute".
- `?` ("no specific value")—useful when you need to specify something in one of the two fields in which the character is allowed, but not the other. For example, if I want my trigger to fire on a particular day of the month (say, the 10th), but don't care what day of the week that happens to be, I would put `"10"` in the day-of-month field, and `"?"` in the day-of-week field. See the examples below for clarification.
- `-`—used to specify ranges. For example, `"10-12"` in the hour field means "the hours 10, 11 and 12".
- `,`—used to specify additional values. For example, `"MON,WED,FRI"` in the day-of-week field means "the days Monday, Wednesday, and Friday".
- `/`—used to specify increments. For example, `"0/15"` in the seconds field means "the seconds 0, 15, 30, and 45". And `"5/15"` in the seconds field means "the seconds 5, 20, 35, and 50". You can also specify `"/` after the `"*` character – in this case `"*` is equivalent to having `"0"` before the `"/`. `"1/3"` in the day-of-month field means "fire every 3 days starting on the first day of the month".
- `L` ("last")—has different meaning in each of the two fields in which it is allowed. For example, the value `"L"` in the day-of-month field means "the last day of the month"—day 31

for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means “7” or “SAT”. But if used in the day-of-week field after another value, it means “*the last xxx day of the month*”—for example “6L” means “*the last Friday of the month*”. When using the “L” option, it is important not to specify lists, or ranges of values, as you’ll get confusing results.

- W (“*weekday*”)—used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify “15W” as the value for the day-of-month field, the meaning is: “*the nearest weekday to the 15th of the month*”. So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However, if you specify “1W” as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not “jump” over the boundary of a month’s days. The “W” character can only be specified when the day-of-month is a single day, not a range or list of days.



The “L” and “W” characters can also be combined in the day-of-month field to yield “LW”, which translates to “*last weekday of the month*”.

- #—used to specify “the nth” XXX day of the month. For example, the value of “6#3” in the day-of-week field means “*the third Friday of the month*” (day 6 = Friday and “#3” = the 3rd one in the month). Other examples: “2#1” = the first Monday of the month and “4#5” = the fifth Wednesday of the month. Note that if you specify “#5” and there is not 5 of the given day-of-week in the month, then no firing will occur that month.



The legal characters and the names of months and days of the week are not case sensitive. MON is the same as mon.

3.4.3. CRON Examples

Here are some full examples:

Expression	Meaning
0 0 12 * * ?	Fire at 12pm (noon) every day
0 15 10 ? * *	Fire at 10:15am every day
0 15 10 * * ?	Fire at 10:15am every day
0 15 10 * * ? *	Fire at 10:15am every day
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day

Expression	Meaning
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month
0 15 10 L * ?	Fire at 10:15am on the last day of every month
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last Friday of every month during the years 2002, 2003, 2004 and 2005
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.



Pay attention to the effects of '?' and '*' in the day-of-week and day-of-month fields!

3.4.4. Notes

Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).

Be careful when setting fire times between mid-night and 1:00 AM - “daylight savings” can cause a skip or a repeat depending on whether the time moves back or jumps forward.

More information is here <http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html>

3.5. Configuring firebird.conf for the best performance

HQbird includes set of optimized configuration files for all Firebird versions from 1.5 to 3.0 – they are located in *HQBird|Configurations*.

If you did not perform a justified tuning of *firebird.conf* or you are using default *firebird.conf*, consider to use one of the optimized files from this collection.

There are three variants of Firebird configuration files for every Firebird architecture: balanced, read-intensive and write intensive. We always recommend to start with balanced *firebird.conf*. Then we recommend to measure actual ratio between reads and writes using HQbird MonLogger tool (tab “Aggregated Performance Statistics”). In 90% of cases there are much more reads than

writes, so the next step is to try read-optimized firebird configuration file.

Firebird configuration greatly depends on the hardware, so if you want to tune Firebird properly, please also read “[Firebird Hardware Guide](#)”, it will help you to understand what parameters must be tuned.

For the deep tuning of high-load Firebird databases IBSurgeon offers Firebird Database Optimization Service: <https://ib-aid.com/en/firebird-interbase-performance-optimization-service/>

Also, HQbird FBDataGuard analyses the database health and sends alerts with intelligent suggestions to increase specific parameters in *firebird.conf*, like TempCacheLimit or LockHashSlots.

Attention!



If you have specified many page buffers in the header of your database and installed SuperClassic or Classic, it can affect Firebird performance. To avoid the potential problem, set page buffers in the header of your database to 0, it will ensure that the value from *firebird.conf* will be used:

```
gfix -buff 0 -user SYSDBA -pass masterkey disk:\path\database.fdb
```


Chapter 4. Monitoring

4.1. Monitoring with HQbird FBDataGuard

4.1.1. Overview

HQbird monitors all aspects of Firebird server and database functioning, and includes continuous monitoring and detailed monitoring.

The continuous monitoring is performed by HQbird FBDataGuard. It is low-invasive, but very effective monitoring which can help to find and resolve the majority of issues with databases performance and stability.

FBDataGuard performs the following monitoring activities:

- Optional performance monitoring with TraceAPI and MON\$
- Monitoring of transactions markers dynamics (Next, OAT, OIT, OST, active transactions)
- Monitoring of lock table activity (queues, deadlocks, mutexes)
- Monitoring of Firebird log (errors, warnings, messages)
- Number of connected users
- Free space monitoring for server, databases and their backups
- Health monitoring through analysis of database metadata and general availability of server and databases
- Number and size of Firebird temporary files (sorting, etc)
- Indices monitoring: health and activity check
- General validation of Firebird database
- Backups statuses monitoring
- Replica availability monitoring

FBDataGuard can graphically represent information gathered during the monitoring, for example, transactions and number of users:

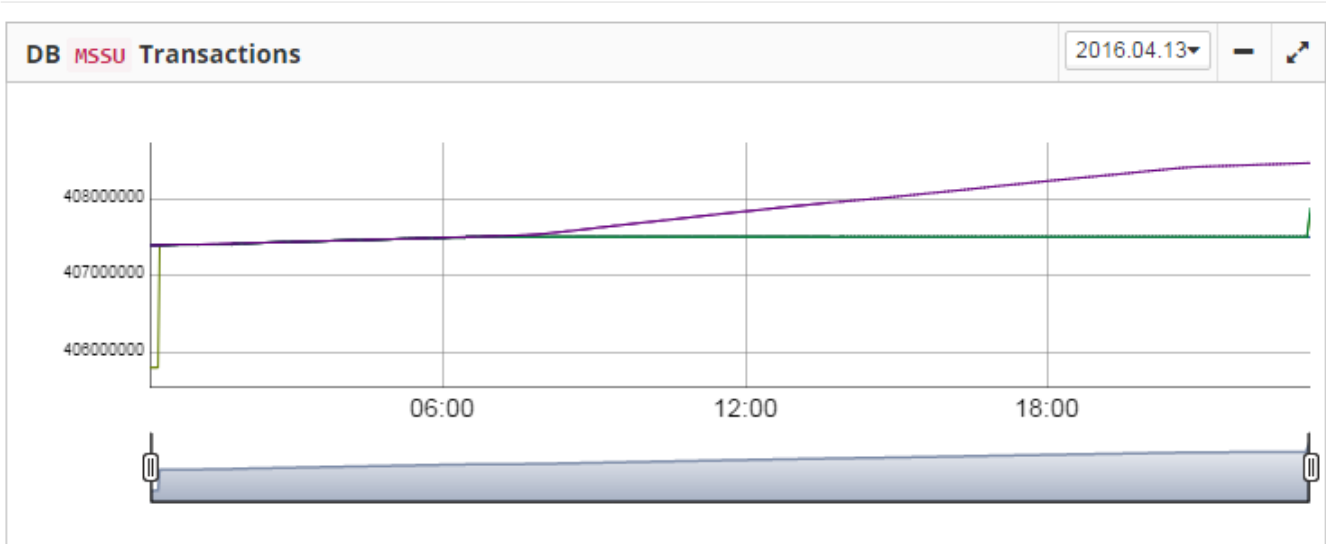


Figure 42. Transaction dynamics in FBDataGuard.

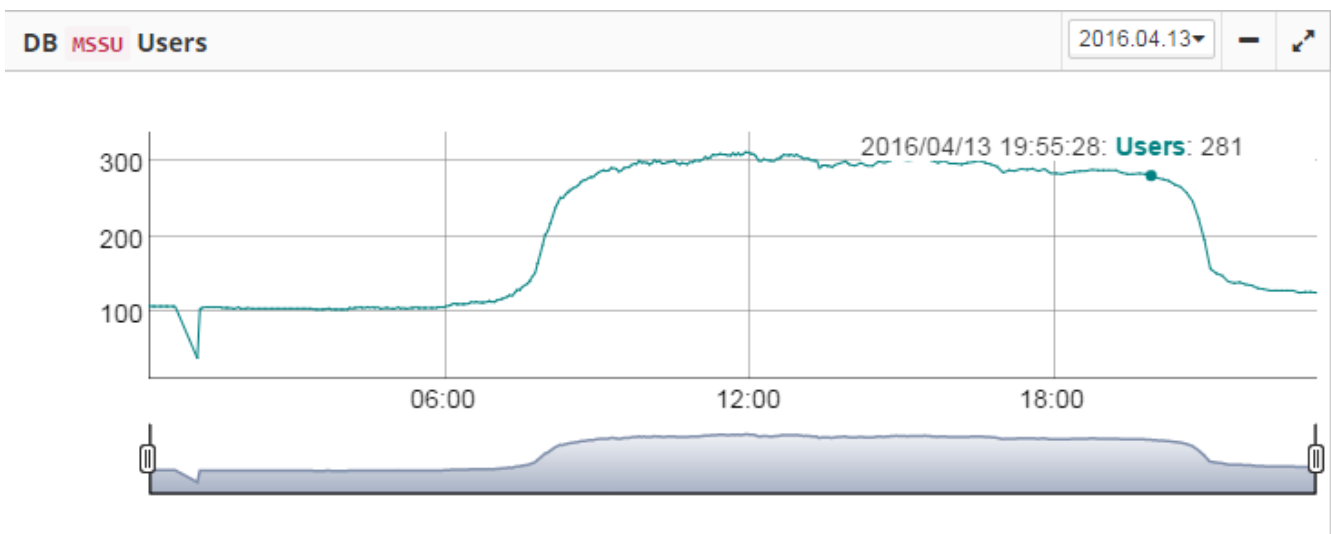




Figure 43. Number of users.

4.1.2. Automatic monitoring with FBDataGuard (Trace API and MON\$)

In the version 2020, HQBird introduces the new approach for the automatic performance monitoring with Firebird MON\$ tables and TraceAPI.

Now it is possible to schedule the regular check of database performance for every HQBird (Standard, Professional, Enterprise) in less than 1 minute.

For this just open tab Performance and setup monitoring for transactions and queries:

Databases						
No	Database nickname	Alias/Path	TR. time	Queries time		
1	dbmaster	f:/fbdata/3.0/billing.fdb	Off	Off		

In order to setup Performance monitoring, specify its mandatory parameters in the dialog:

The first mandatory parameter is “**Enable performance monitoring**” — it must be enabled to run traces by schedule.

The next important parameters are “Start trace session at” and “Stop trace session”. They contain CRON expressions which specify when tracing starts and stops.

By default, trace is set to start at 10-30 and to end at 11-00. It is recommended to adopt tracing schedule for your needs. Below you can see the table with some popular options.

CRON expression for		Description
Start	End	
0 0 * ? * *	0 10 * ? * *	Run trace every hour from 0 to 10 minutes
0 0 8 ? * *	0 0 17 ? * *	Run trace every day from 8-00 to 17-00
0 30 10,13,15 ? * *	0 0 11,14,16 ? * *	Run trace sessions every day from 10-30 to 11-00, 13-30 to 14-00 and from 15-30 to 16-00

The next important parameter is time threshold for the slow queries, it is set in the field “Log SQLs with execution time more than”. In this field you need to set time threshold (in milliseconds), after exceeding it logs will be stored and analyzed.

By default, the time is set to 1000 milliseconds, or 1 second. It means, that only queries which take more than 1 seconds, will be logged and analyzed.

We recommend keeping 1000 ms as a basic value, until your database is very slow: in this case 3000-5000 ms can be a good start.

“Send email” check mark indicates if there necessity to send the performance report. The email settings from Alerts configuration will be used to send performance report.

For more advanced settings, “Performance Monitoring” dialog has additional parameters (normally, you don’t need to adjust them).

Performance monitoring (TraceAPI) ✕

Enable performance monitoring

Output folder (no need to change it)

Start trace session at

Log SQLs with execution time more than (ms)

Stop trace session

Send email

Less

Configuration template

Database filter AUTO FILENAME ALIAS MANUAL

Database name filter

Trace format AUTO 2.5 3/4

Keep recent reports

Ignore comments

- **“Configuration template”** — name of the configuration template file which should be used for trace settings
- **“Database filter”** — how the database should be identified. Usually AUTO is enough, it will trace specified database. In case of Filename or Alias it will use filename or alias to filter database events. «Manual» provides an ability to set any regular expression, to trace several databases, for example, or more than one alias for the single database.
- **“Database name filter”** it is used in case of “MANUAL” selection.
- **“Trace format”** — AUTO means automatic selection, 2.5 or 3.0 will force format for 2.5 or 3.0. Usually there is no need to change it.
- **“Keep recent reports”** — it specifies how many reports should be kept in the “Output folder” for possible retrospective usage.

As a result of this job, HQbird will generate the performance report, which will be stored in the Output folder as a file with the extension **html**, and it will be sent by email (in case if “Send email” is enabled). Also the most recent performance is available for review and download in the HQBird interface.

The screenshot shows the HQbird performance analysis interface. At the top, there is a 'Databases' table with columns: No, Database nickname, Alias/Path, TR. time, and Queries time. Below this, there is a section for 'Long-running active writeable transactions' which currently shows 'No report'. The main part of the interface displays a detailed performance report for a specific query, including the following text:

```

RANK 1 of 19; LINE: 711; TIME: 2624ms; INFO: 2624 ms, 20479 read(s), 1317962 fetch(es)
2019-11-03T20:53:12.8860 (4316:0000000187C0040) EXECUTE_STATEMENT_FINISH
F:\FBADATA\3.0\BILLING.FDB (ATT_287, SYSDBA:NONE, NONE, TCPV6:::1/64767)
c:\HQbird\Firebird30\isql.exe:852
(TRA_688, CONCURRENCY | WAIT | READ_WRITE)
Statement 1404:
-----
select count(*) from service where code_service > 456852
PLAN (SERVICE INDEX (PK_SERVICE))
1 records fetched
2624 ms, 20479 read(s), 1317962 fetch(es)

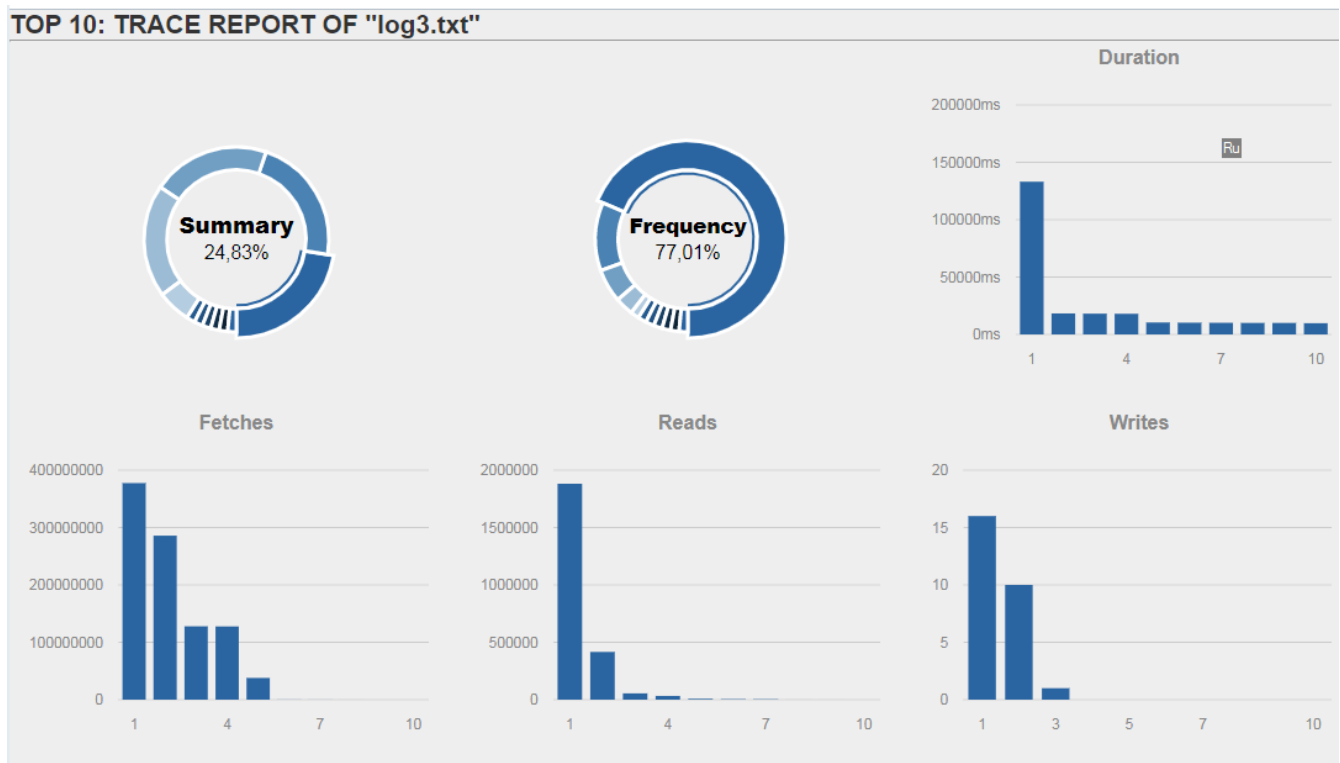
Table           Natural  Index  Update  Insert  Delete  Backout  Purge  Expunge
-----
SERVICE        1297481
Statement hash 3ca777e6d4cdb69d65a4d9fd3168cc10f2bbdbc
    
```

4.1.3. What can we see in the performance report?

The HQbird FBDataGuard performance analysis provides 3 types of reports:

1. list of queries sorted by their time — “Sort by duration”;
2. list of queries sorted by its frequency — “Sort by frequency”;
3. list of queries sorted by the total time (i.e., summary execution time for queries with the same text and various parameters) — “Sort by summary”.

In the beginning of the report you will see the graphical representation of most problematic SQL queries:



When you click “Sort by duration” (it is a default option), you will see SQL queries and stored

procedures which took the longest time to execute first.

Normally there will be long-running reports and other big SQLs.

SORT BY DURATION [VIEW [PROCESS SUMMARY](#) OR SORT BY [DURATION](#), [FREQUENCY](#), [TIME-SUMMARY](#), [FETCHES](#), [WRITES](#), [READS](#)]

RANK 1 of 19; LINE: 711; TIME: 2624ms; INFO: 2624 ms, 20479 read(s), 1317962 fetch(es)

```
2019-11-03T20:53:12.8860 (4316:0000000187C0040) EXECUTE_STATEMENT_FINISH
F:\FBDDATA\3.0\BILLING.FDB (ATT_287, SYSDBA:NONE, NONE, TCPv6:::1/64767)
c:\HQBird\Firebird30\isql.exe:852
(TRA_688, CONCURRENCY | WAIT | READ_WRITE)
```

Statement 1404:

```
-----
select count(*) from service where code_service > 456852
-----
PLAN (SERVICE INDEX (PK_SERVICE))
1 records fetched
2624 ms, 20479 read(s), 1317962 fetch(es)
```

Table	Natural	Index	Update	Insert	Delete	Backout	Purge	Expunge
SERVICE		1297481						

Statement hash 3ca777e6d4cdb69d065a4d9fd3168cc10f2bbddc

When you click on “Sort by frequency” link in the header of the report, you will see most frequent queries: i.e., those queries which started frequently (among logged queries).

SORT BY FREQUENCY [VIEW [PROCESS SUMMARY](#) OR SORT BY [DURATION](#), [FREQUENCY](#), [TIME-SUMMARY](#), [FETCHES](#), [WRITES](#), [READS](#)]

RANK 1 of 12; FREQUENCY: 21.05% (4 of 19); took: 6.78% (287 of 4231 ms); FETCH:763; READ:85; WRITE:0; MARK:44

```
2019-11-03T20:42:07.4130 (4316:0000000187C0B40) EXECUTE_STATEMENT_FINISH
F:\FBDDATA\3.0\BILLING.FDB (ATT_288, SYSDBA:NONE, UTF8, TCPv6:::1/64781)
D:\ospanel\modules\http\Apache_2.4-PHP_7.2-7.3-x64\bin\httpd.exe:7188
(TRA_632, CONCURRENCY | WAIT | READ_WRITE)
```

Statement 158:

```
-----
EXECUTE PROCEDURE ACL_UTILS.WRITE_SERVICE(?, ?, ?, ?, ?, ?)
param0 = integer, "1"
param1 = integer, "<NULL>"
param2 = smallint, "0"
param3 = varchar(1020), "/horses/main/ru"
param4 = varchar(128), "t1p01o3qhlapm0hu56ie87n4qa"
param5 = varchar(60), "127.0.0.1"
0 records fetched
111 ms, 29 read(s), 189 fetch(es), 11 mark(s)
```

Table	Natural	Index	Update	Insert	Delete	Backout	Purge	Expunge
RDB\$INDICES		22						
RDB\$GENERATORS		1						
RDB\$RELATION_CONSTRAINTS		3						
ITEM		59						
PRICE		3						
SERVICE				1				
WEBUSER		2						

Statement hash e90ef2f7734f14e502fea12ebc9c645e9458985d

For example, in this case the statement SP_GETINVOICE_REPORT was run 46 times. It means that this query heavily affects the overall performance, and it should be optimized first.

When you click on Sort by summary, you will see the queries which took the most part of the time (among logged queries). These queries usually are the best candidates for the optimization.

By default, this monitoring is off. To enable it, click on «Enable transactions monitoring». In general, it is enough, this monitoring does not require further setup.

Let's consider its settings:

- When to log transactions: This parameter defines when to check MON\$ tables for long-running active transactions. By default, it is set to run every 5 minutes (see CRON statement). You can make less often on heavy loaded databases, up to once per hour.
- Output folder: it is a service parameter.
- Show transactions older then (minutes): it specifies the time threshold to show the transaction (and associated connection) in the list of long-running-transactions. By default, this threshold is 60 minutes: it means that writeable transactions which started more than 1 hour ago, will be considered as long-running.
- Send alert if oldest active transaction is older then (minutes): the same, but it triggers alert and, if email notifications are enabled, the automatic email with the details of long running active transaction. The text of the alert looks like the following:

There is a long running active transactions: it was started at 11/13/17 1:19 PM (and run at least 107 minutes) from ::1/51068 by C:\HQBird\Firebird30\isql.exe. Such transactions block garbage collection, please perform transactions analysis with HQbird MonLogger.

- Show only NN oldest active transactions: it specifies how many records will be shown in the list with long-running transactions.

The list of long-running active transactions is shown on the screenshot below:

No	Database nick name	Alias/Path to database	TR. time	Queries time
1	test	h:\EMPLOYEE_30.FDB	1 min	0 sec

test : Long-running active writeable transactions																	
No	User	Connected	Att ID	Att Name	Transaction ID	Stat ID	Oldest transaction	State	Protocol	Address	Process	Role	Charset ID	Server PID	Remote PID	GC	
1	SYSDBA	2017-11-13 18:05:09.6	82460	H:\EMPLOYEE_30.FDB	27659	13	2017-11-13 18:05:09.644	0	TCPv6	::1/51044	C:\HQBird\Firebird30\isql.EXE	NONE	0	4296	1528	1	
2	SYSDBA	2017-11-13 18:05:09.6	82460	H:\EMPLOYEE_30.FDB	27660	13	2017-11-13 18:05:09.644	0	TCPv6	::1/51044	C:\HQBird\Firebird30\isql.EXE	NONE	0	4296	1528	1	

Here you can see that isql.exe started 2 long-running active transactions to the database *h:\employee_30.fdb* at November 13, 18-05.

4.1.5. How to select a tool for detailed monitoring

FBDataGuard is the first line of a defense for Firebird database; once FBDataGuard encounters something suspicious inside the monitored areas, it sends an alert with description of the issue.



Important!

If you have several Firebird servers, we offer HQbird Control Center application which gathers alerts data from the Firebird servers and databases and shows them at the single screen. Contact our for more details.

After receiving such alert from FBDataGuard the database administrator should proceed with detailed investigation of the problem.

The choice of tool for detailed monitoring depends on the type of detected problem.

If FBDataGuard reports long-running active transaction (Next-OAT), it is necessary to use **HQbird Mon\$Logger** to detect the source of currently running active transaction.

If stuck of oldest interesting transaction is reported, database administrator must plan an explicit sweep to clean uncollected garbage with FBDataGuard sweep job (if it is necessary) and then plan tracking of forced rollbacks with Performance Monitoring in FBDataGuard, or, if it is an old version of Firebird (2.1 or older), with **FBScanner**.

If users report slowness problem with some queries, Perfusion or FBScanner should be used.

If there is unusual spikes in transaction behavior, **IBTransactionMonitor** can be a good addition to HQbird FBDataGuard to clarify the situation.

The problems with general database performance and occasional or periodic slowness require an analysis of database structure, which can be done only with HQbird Database Analyst.

Below we will consider how to work with HQbird monitoring tools in more details.

4.2. Monitoring with MON\$ tables: HQbird MonLogger

HQbird MonLogger is a tool to analyze monitoring tables output in Firebird and find problems with slow SQL queries, wrongly designed transactions (long-running transactions, transactions with incorrect isolation level, etc) and identify problematic applications.

MonLogger can connect to Firebird database with performance problems and identify what is the reason of slowness: is it some user attachment, slow SQL query or long-running transaction?

MonLogger supports Firebird 2.1, 2.5, 3.0 and 4.0 – for older Firebird versions or InterBase please use FBScanner.

MonLogger can show you:

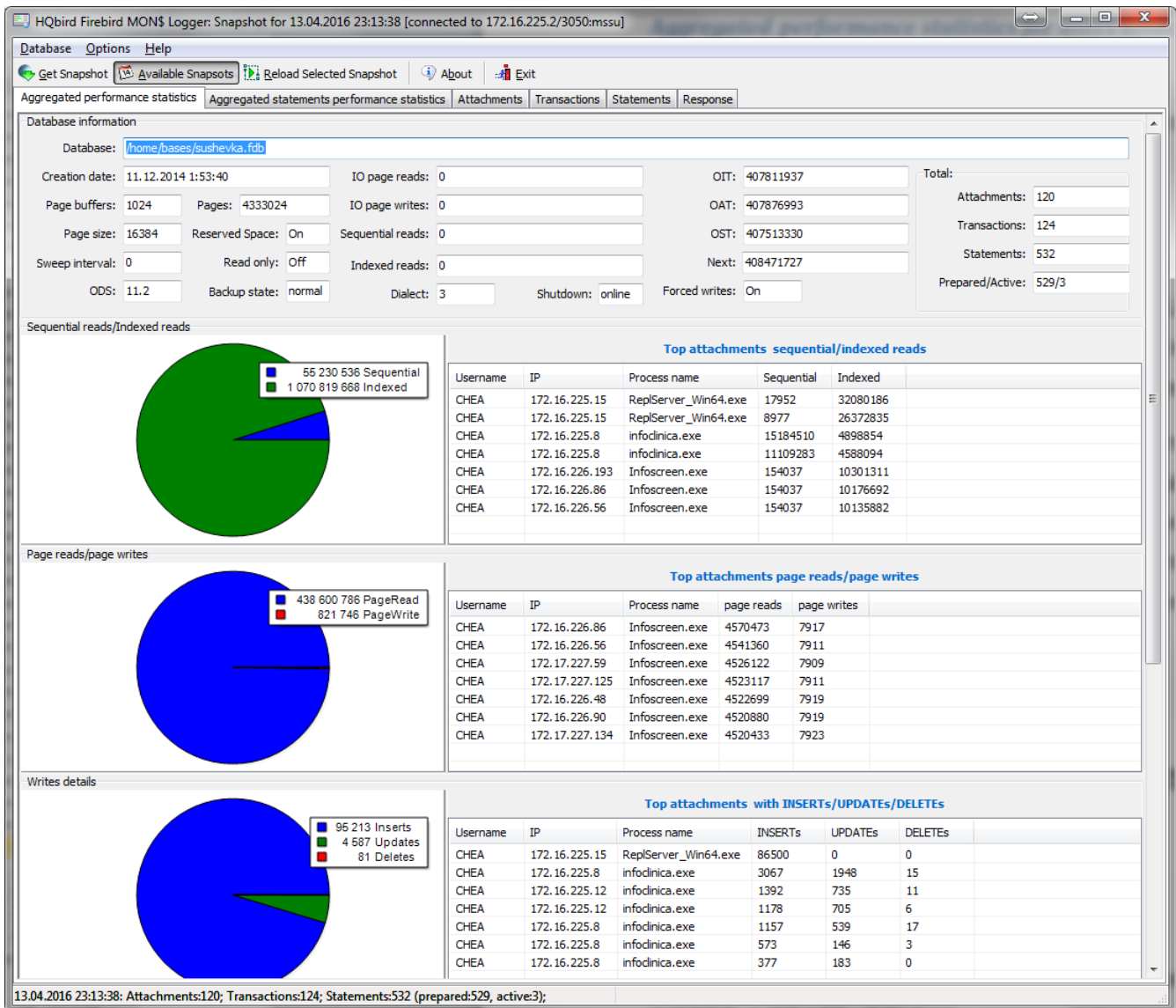
- Top attachments with highest number of IO operations, non-indexed and indexed reads
- Top SQL statements with highest number of IO operations, non-indexed and indexed reads
- Problematic transactions: long-running transactions, transactions with erroneous isolation level, read/write transactions, and related information: when they started, what applications started these transactions, from what IP address, etc
- Attachments and statements with the most intensive garbage collection actions

- Read/write ratio, INSERTS/UPDATE/DELETE ratio, and more.

After connection to the database where you want to find performance problems, several snapshots of monitoring tables should be done – click on “Get Snapshot” to take snapshot.

4.2.1. Aggregated performance statistics for users attachments

At the first screen we can see aggregated statistics for database connections, and identify connections with the biggest problems:



Sequential reads / Indexed reads

“Sequential reads / Indexed reads” shows us total ratio between sequential (non-indexed) reads and indexed reads in application. Usually number of non-indexed reads should be low, so big percent of sequential reads is sign that many SQL queries have NATURAL execution plans, and they could be a reason of slow response time.

Click on record in “TOP attachments: sequential/indexed reads” will bring you to tab “Attachments”, where you can see more details about Attachment, and then jump to tab “Transactions” or “Statements”, where you will see transactions and attachments linked with selected attachment (if checkmark “Link to selected attachment” is on, otherwise all

transactions/statements for all attachments will be shown).

Write details

“Write details” gives you an overview of write operations: ratio between INSERTs/UPDATEs/DELETEs among all database attachments. In the table of top writers you can see attachments with the biggest number of write operations. It is useful to identify applications or software modules which performs excessive number of update or deletes (which are the most dangerous operations in terms of garbage collections).

Garbage collection details

What garbage collection operations mean?

- Purge — engine removes back-versions, only primary version is in database.
- Expunge — both primary version and all back-versions were deleted.
- Back-out — remove only primary version (due to rollback).

Usually we can associate purge with UPDATE operation, Expunge with DELETE, and Backout with rollback of INSERT or UPDATE. Many backouts could mean that there is a problem with transaction management in the application.

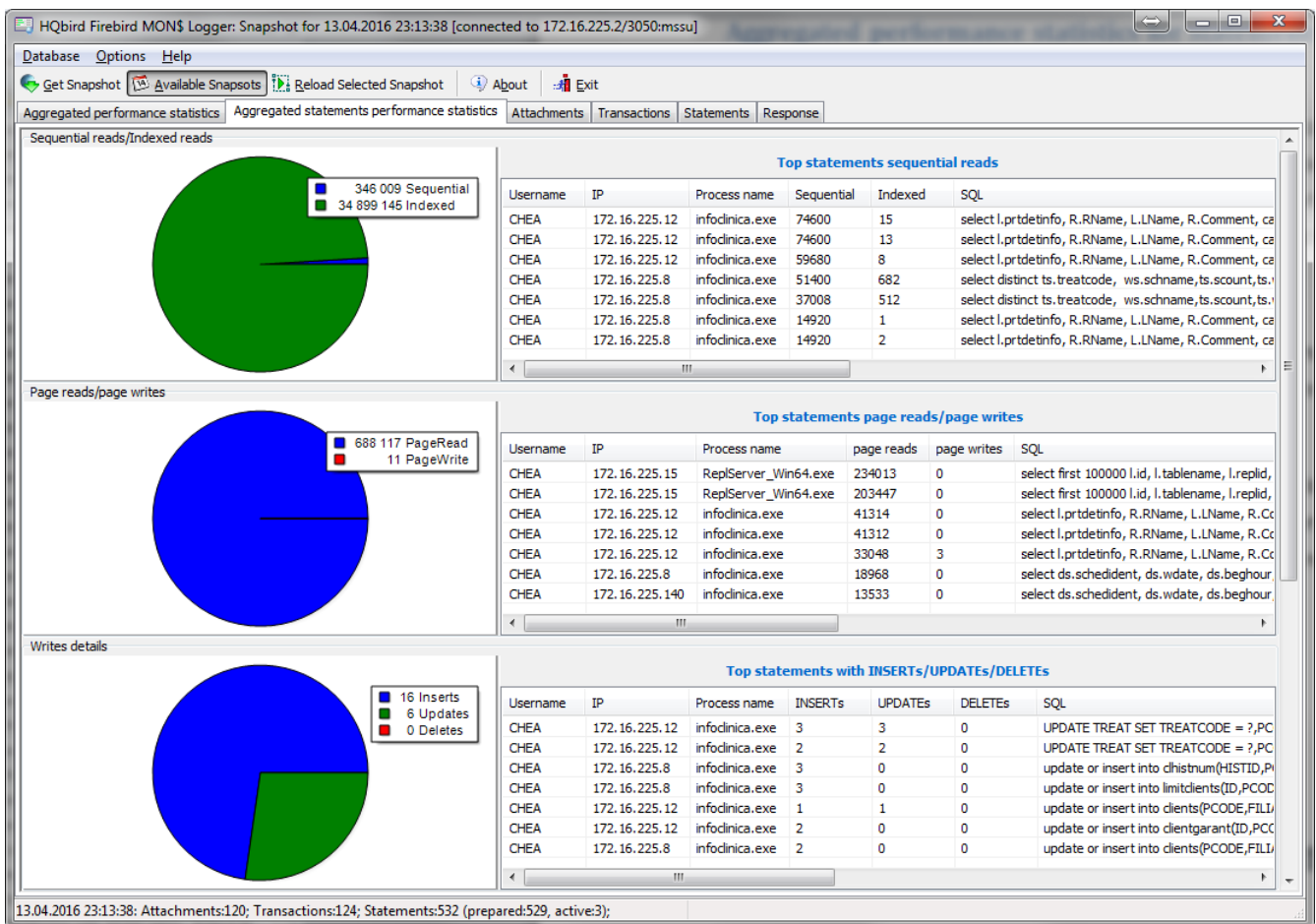
Memory usage

“Memory usage” graph shows us total memory used by all active attachments now, and peak of allocated memory for them in the past.

List of top attachments by memory usage shows us the biggest memory consumers among your attachments. It is useful to find applications or software modules with excessive memory usage.

4.2.2. Aggregated performance statistics for statements

At the second tab you can find aggregated performance statistics for statements.



This statistics better reflects the momentary situation in the database – since monitoring tables collect information since the beginning of each object life, statements you can see here are those which were running during the moment when snapshot was taken.

Sequential reads / Indexed reads

In this list we can see top statements which perform many sequential reads from the database. Usually such statements require SQL tuning – either through indices tuning, or through SQL query redesign.

To tune the query, check its execution plan: usually it is possible to improve query speed by eliminating NATURAL in plans with new indices or query redesign. Click on the statement in this list to open tab “Statements”, where you can find more details about selected statement, and jump to associated transaction or attachment.

Page reads/page writes

This graphs and list shows brief information about top statements which perform many reads – it means that they consume significant IO and can affect performance of other queries. SQL statements with peak values should be carefully checked for optimal performance.

Write details for statements

At this graph you can see what writing SQL statements were doing at the moment when snapshot of monitoring tables was taken, and identify UPDATES and DELETES which made many changes in the database.

Garbage collection details for statements

At this graph we can see how many garbage collection operations were done by statements running at the moment of snapshot.

Memory usage for statements

Unlike aggregated memory usage statistics for attachment, statements' memory usage can show us list of exact statements which consume a lot of memory at the moment.

4.2.3. Attachments

The third tab is “Attachments”. You can open this tab directly to jump there by clicking one of the records at “Aggregated performance statistics”.

user	role	remote_address	started at	attachment_id	gc	Remote process	record_seq_reads	record_idx_reads	record_inserts	record_updates	record_deletes	r
CHEA	NONE	172.16.225.8	13.04.2016 19:38:36	11437916	Yes	infodlnica.exe	15184510	4898854	1157	539	17	
CHEA	NONE	172.16.225.8	13.04.2016 12:26:05	11421275	Yes	infodlnica.exe	11109283	4588094	3067	1948	15	
CHEA	NONE	172.16.225.12	13.04.2016 19:40:20	11437959	Yes	infodlnica.exe	7022236	2813958	1392	735	11	
CHEA	NONE	172.16.225.8	13.04.2016 15:55:49	11430140	Yes	infodlnica.exe	1872220	2310958	377	183	0	
CHEA	NONE	172.16.225.8	13.04.2016 13:22:43	11424008	Yes	infodlnica.exe	1627368	8551870	573	146	3	
CHEA	NONE	172.16.225.12	13.04.2016 22:53:15	11441112	Yes	infodlnica.exe	912643	810178	1178	705	6	
CHEA	NONE	172.16.225.176	13.04.2016 7:52:05	11409832	Yes	infodlnica.exe	766062	2978342	272	157	8	
CHEA	NONE	172.16.225.140	13.04.2016 16:09:25	11430689	Yes	infodlnica.exe	430605	715065	3	3	0	
CHEA	NONE	172.16.225.12	13.04.2016 19:58:46	11438363	Yes	infodlnica.exe	155495	539147	161	63	3	
CHEA	NONE	172.16.226.65	13.04.2016 1:08:18	11406409	Yes	Infoscreen.exe	154059	9724361	0	0	0	
CHEA	NONE	172.16.226.187	13.04.2016 1:08:54	11406494	Yes	Infoscreen.exe	154059	9712519	0	0	0	
CHEA	NONE	172.16.226.169	13.04.2016 1:08:12	11406394	Yes	Infoscreen.exe	154059	10044138	0	0	0	
CHEA	NONE	172.16.226.139	13.04.2016 1:08:14	11406400	Yes	Infoscreen.exe	154059	9721625	0	0	0	
CHEA	NONE	172.16.226.149	13.04.2016 1:08:19	11406412	Yes	Infoscreen.exe	154059	10053226	0	0	0	
CHEA	NONE	172.16.226.55	13.04.2016 1:08:24	11406425	Yes	Infoscreen.exe	154059	9953475	0	0	0	
CHEA	NONE	172.16.226.64	13.04.2016 1:08:13	11406398	Yes	Infoscreen.exe	154059	9909128	0	0	0	
CHEA	NONE	172.16.226.95	13.04.2016 1:08:13	11406397	Yes	Infoscreen.exe	154037	10020220	0	0	0	
CHEA	NONE	172.16.226.184	13.04.2016 1:08:57	11406502	Yes	Infoscreen.exe	154037	9903042	0	0	0	
CHEA	NONE	172.17.227.45	13.04.2016 1:08:13	11406399	Yes	Infoscreen.exe	154037	10072340	0	0	0	
CHEA	NONE	172.16.226.68	13.04.2016 1:08:27	11406441	Yes	Infoscreen.exe	154037	10049719	0	0	0	
CHEA	NONE	172.16.226.87	13.04.2016 1:08:14	11406401	Yes	Infoscreen.exe	154037	9965138	0	0	0	
CHEA	NONE	172.16.226.193	13.04.2016 1:08:28	11406444	Yes	Infoscreen.exe	154037	10301311	0	0	0	
CHEA	NONE	172.16.226.176	13.04.2016 1:08:16	11406405	Yes	Infoscreen.exe	154037	9900171	0	0	0	
CHEA	NONE	172.16.226.101	13.04.2016 1:08:16	11406406	Yes	Infoscreen.exe	154037	10009501	0	0	0	
CHEA	NONE	172.17.227.96	13.04.2016 1:08:28	11406445	Yes	Infoscreen.exe	154037	9942458	0	0	0	
CHEA	NONE	172.16.226.136	13.04.2016 1:08:38	11406472	Yes	Infoscreen.exe	154037	9748900	0	0	0	
CHEA	NONE	172.16.226.94	13.04.2016 1:08:41	11406476	Yes	Infoscreen.exe	154037	10015656	0	0	0	
CHEA	NONE	172.17.227.63	13.04.2016 1:08:32	11406460	Yes	Infoscreen.exe	154037	9857614	0	0	0	
CHEA	NONE	172.17.227.93	13.04.2016 1:08:30	11406453	Yes	Infoscreen.exe	154037	9741158	0	0	0	
CHEA	NONE	172.16.226.180	13.04.2016 1:08:56	11406501	Yes	Infoscreen.exe	154037	9710737	0	0	0	
CHEA	NONE	172.16.226.143	13.04.2016 1:08:23	11406419	Yes	Infoscreen.exe	154037	9854902	0	0	0	
CHEA	NONE	172.16.226.157	13.04.2016 1:08:12	11406395	Yes	Infoscreen.exe	154037	9714968	0	0	0	
CHEA	NONE	172.16.226.57	13.04.2016 1:08:24	11406424	Yes	Infoscreen.exe	154037	9831570	0	0	0	

“Attachments” shows the list of users connected to the Firebird database, with many useful details: USER and ROLE of attachment, start time and ID of attachment, is there garbage collection enabled for the attachment, name of remote process which established an attachment, and several accumulated performance counters for the attachment: number of sequential reads [done by attachment since its start], number of indexed reads, number of inserts, updates and deletes, as well records backouts, purges and expunges.

By default, some of columns of attachment are switched off, to show only most important information.

Of course, every time you click on attachment, you can jump to transactions running inside it, and

then to statements. There is a checkbox in the left upper corner of Transactions and Statements tabs, which controls the behavior—when checked, only transactions and statements marked by selected attachment ID, will be shown.

4.2.4. Transactions

Tab “Transactions” shows active transactions at the moment when snapshot was taken.

started at	transaction_id	attachment_id	state	isolation_mode	lock_timeout	read_only	auto_commit	auto_undo	record_seq_reads	record_idx_reads	re
13.04.2016 7:52:19	407545493	11409832	active	read committed record version	timeout -1	read only	No	Yes	207979	801941	
13.04.2016 7:52:36	407545755	11409832	active	read committed record version	timeout -1	read only	No	Yes	210984	522155	
13.04.2016 8:01:58	407555165	11409832	idle	read committed record version	timeout -1	read only	No	Yes	0	150	
13.04.2016 8:23:50	407580809	11410943	active	read committed record version	timeout -1	read only	No	Yes	0	3430716	
13.04.2016 9:14:47	407644463	11412867	active	read committed record version	timeout -1	read only	No	Yes	0	166660	
13.04.2016 10:26:38	407732169	11409832	idle	read committed record version	timeout -1	read only	No	Yes	0	1	
13.04.2016 12:12:08	407857456	11409832	active	read committed record version	timeout -1	read only	No	Yes	83169	157340	
13.04.2016 12:26:14	407873598	11421275	active	read committed record version	timeout -1	read only	No	Yes	843	256133	
13.04.2016 12:28:37	407876700	11421275	idle	read committed record version	timeout -1	read only	No	Yes	0	1854	
13.04.2016 12:28:48	407876993	11421275	active	concurrency	timeout -1	read write	No	Yes	0	2	
13.04.2016 12:29:39	407876993	11421275	active	concurrency	timeout -1	read write	No	Yes	0	2	
13.04.2016 13:24:19	407938807	11424008	active	read committed record version	timeout -1	read only	No	Yes	659618	913602	
13.04.2016 13:24:21	407938850	11424008	idle	read committed record version	timeout -1	read only	No	Yes	629	2073	
13.04.2016 13:26:28	407941240	11424008	active	read committed record version	timeout -1	read only	No	Yes	592183	1205794	
13.04.2016 13:26:34	407941329	11424008	idle	read committed record version	timeout -1	read only	No	Yes	0	2	
13.04.2016 13:36:34	407952718	11424600	active	read committed record version	timeout -1	read only	No	Yes	775	25479	
13.04.2016 13:37:13	407953466	11424600	idle	read committed record version	timeout -1	read only	No	Yes	0	150	
13.04.2016 13:39:27	407955814	11421275	active	read committed record version	timeout -1	read only	No	Yes	57550	63844	
13.04.2016 15:26:01	408063655	11421275	active	concurrency	timeout -1	read write	No	Yes	0	2	
13.04.2016 15:55:58	408096795	11430140	active	read committed record version	timeout -1	read only	No	Yes	923	287339	
13.04.2016 15:57:53	408098881	11430140	idle	read committed record version	timeout -1	read only	No	Yes	0	1635	
13.04.2016 16:10:18	408112846	11430689	active	read committed record version	timeout -1	read only	No	Yes	755	90673	
13.04.2016 17:18:16	408190047	11433362	active	read committed record version	infinite wait	read only	No	Yes	3090	2068	
13.04.2016 17:20:14	408192461	11433362	active	read committed record version	infinite wait	read write	No	Yes	0	12429	
13.04.2016 17:24:54	408197321	11430689	idle	read committed record version	timeout -1	read only	No	Yes	0	150	
13.04.2016 19:39:10	408341407	11437916	active	read committed record version	timeout -1	read only	No	Yes	11539	471037	
13.04.2016 19:40:31	408343078	11437959	active	read committed record version	timeout -1	read only	No	Yes	1637	453671	
13.04.2016 19:41:05	408343691	11437959	idle	read committed record version	timeout -1	read only	No	Yes	0	1468	
13.04.2016 19:42:47	408345715	11437916	idle	read committed record version	timeout -1	read only	No	Yes	0	1578	
13.04.2016 19:42:48	408345735	11437916	active	concurrency	timeout -1	read write	No	Yes	0	3	
13.04.2016 19:43:51	408347076	11437959	active	concurrency	timeout -1	read write	No	Yes	0	2	
13.04.2016 19:46:55	408350549	11437959	active	concurrency	timeout -1	read write	No	Yes	0	2	

If checkbox “Link to selected attachment” is enabled, only transactions for selected attachment will be shown, otherwise all transactions are shown.

One of the most important characteristics is a lifetime of transactions: since Firebird is designed to work with short write transactions, it is important to keep them as short as possible. MonLogger highlights transactions with isolation modes and read-write settings which hold Oldest Active Transaction and therefore provoke excessive record versions to be not cleared. If you see such transaction and it started a while ago, it means that it can be responsible for excessive records versions.

Sort on “started at” column and look for old transactions, marked in red: all writeable transactions and read only snapshots stuck Oldest Active Transaction and provoke excessive record versions to be hold. Identify where these transactions started (right-click and select “View parent attachment”) and fix your code to commit this transaction earlier.

4.2.5. Statements

started at	statement_id	attachment_id	transaction_id	state	sql_text	record_seq_reads	record_idx_reads	record_inserts	record_updates
N/A	9447143	11430140	0	IDLE	select l.prtdefinfo, R.RName, L.LName, R.Comment, cas	14920	2	0	0
N/A	9378286	11437916	0	IDLE	select l.prtdefinfo, R.RName, L.LName, R.Comment, cas	14920	1	0	0
N/A	633382	11409832	0	IDLE	select t.pcode, payplan, t.doctype, t.naraddclose, t.trea	8962	98203	0	0
N/A	9610440	11437959	0	IDLE	select distinct a.agrid, a.agnum, a.agname, l.lid, l.short	2414	17	0	0
N/A	9102254	11438363	0	IDLE	select jid, jname, cashid from jpersons	1692	0	0	0
N/A	9597079	11437959	0	IDLE	select jid, jname, cashid from jpersons	1692	0	0	0
N/A	9634244	11441112	0	IDLE	select jid, jname, cashid from jpersons	1692	0	0	0
N/A	9335793	11440038	0	IDLE	select jid, jname, cashid from jpersons	1692	0	0	0
N/A	9644275	11437916	0	IDLE	select case when tp.dcode is null then d.fullname else (280	15002	0	0
N/A	9647284	11421275	0	IDLE	select case when tp.dcode is null then d.fullname else (119	4113	0	0
N/A	9597080	11437959	0	IDLE	select cashid, cashname, jid	69	0	0	0
N/A	4796406	11424600	0	IDLE	SELECT DBPATH FROM FILIALS WHERE ISMAIN = ?	63	0	0	0
N/A	9634245	11441112	0	IDLE	select cashid, cashname, jid	46	0	0	0
N/A	9593799	11441112	0	IDLE	SELECT DBPATH FROM FILIALS WHERE ISMAIN = ?	42	0	0	0
N/A	9102255	11438363	0	IDLE	select cashid, cashname, jid	23	0	0	0
N/A	9335794	11440038	0	IDLE	select cashid, cashname, jid	23	0	0	0
N/A	9585627	11430140	0	IDLE	SELECT DBPATH FROM FILIALS WHERE ISMAIN = ?	21	0	0	0

```

Text
select cashid, cashname, jid
from cashref
where filial = ? and (cast(? as bigint) <= 0 or jid = ? or jid is null or cashid = ?)
order by cashname

```

13.04.2016 23:13:38: Attachments:120; Transactions:124; Statements:532 (prepared:529, active:3);

Tab “Statements” shows statements active at the moment of snapshot: if you need to catch all statements FBPerfMon or FBScanner should be used (all these tools are part of IBSurgeon Optimization Pack).

If “Link to selected attachment” is enabled, only statements for specific attachment will be shown, otherwise all active statements are in the list.

Some statements have no associated transaction id (=0): these queries are prepared, but not executed.

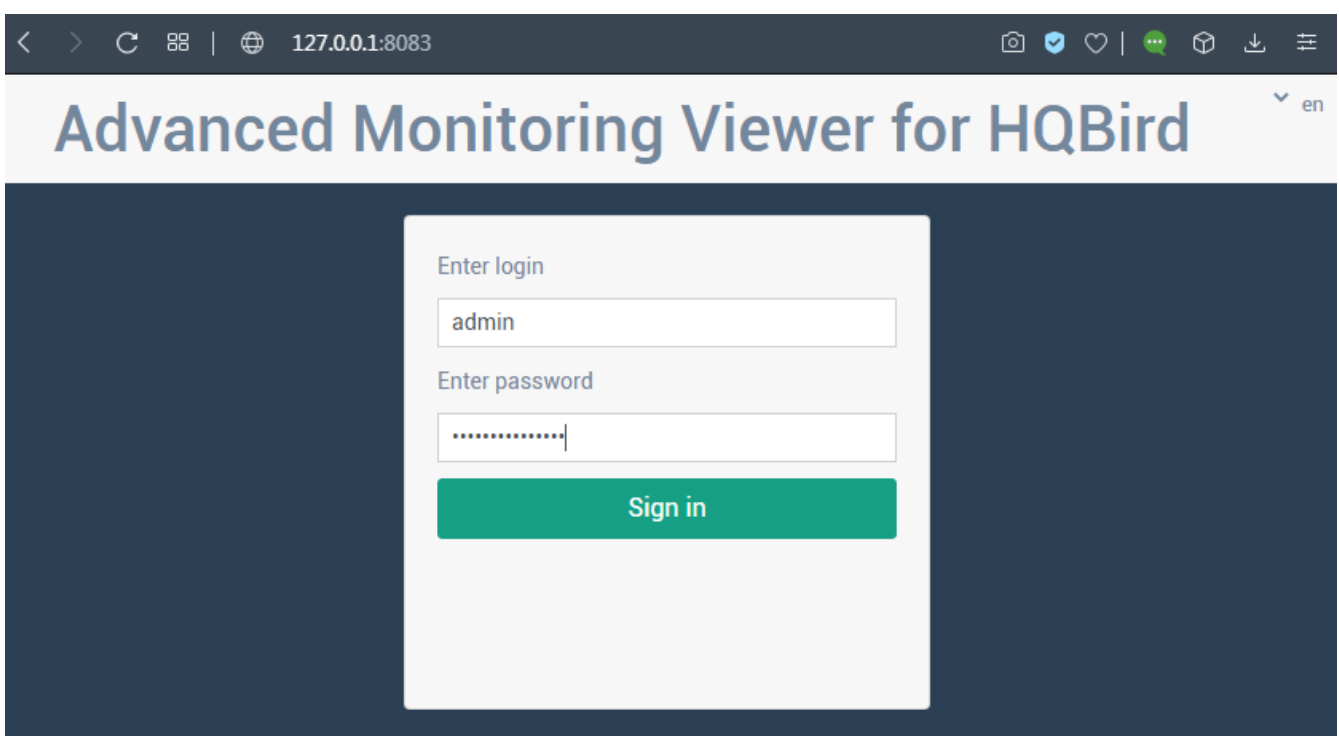
4.3. Advanced Monitor Viewer

Advanced Monitor Viewer allows graphical display of additional performance counters. They are based on both trace data and data from monitoring tables, plus additional system utilities such as wmic (Windows) are used.

To launch the "Advanced Monitor Viewer" click on the corresponding item of the Start menu "IBSurgeon/HQbird Server Side 2022/Advanced Monitor Viewer" or run the script *AVM/quick_start.cmd*.

After successful launch, the page <http://127.0.0.1:8083> will open in your default browser.

You will be prompted to log in:

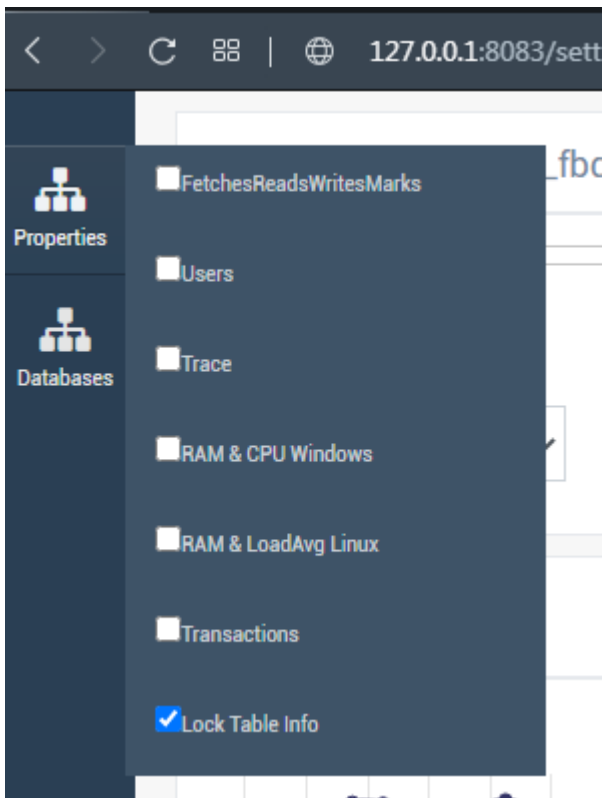


The default login and password are the same as for DataGuard: "admin / strong password".

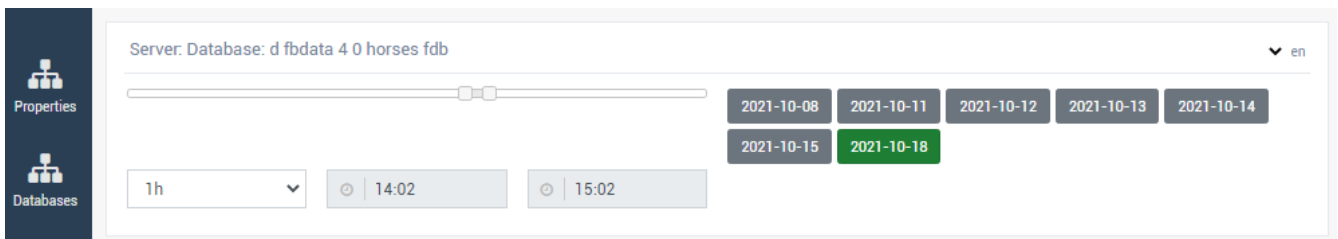
After successful authentication, a page will open with a panel on which various graphs are located, displaying the system load at different points in time.

On the left side of the page, you will see two buttons: "Properties" and "Databases". The first one opens a context menu for selecting counters that will be displayed on the charts.

On the left side of the page, you will see two buttons: "Properties" and "Databases". The first one opens a context menu for selecting counters that will be displayed on the charts. The second, opens the context menu in which you can select the database for which these counters are displayed. The database must be registered for monitoring with DataGuard.



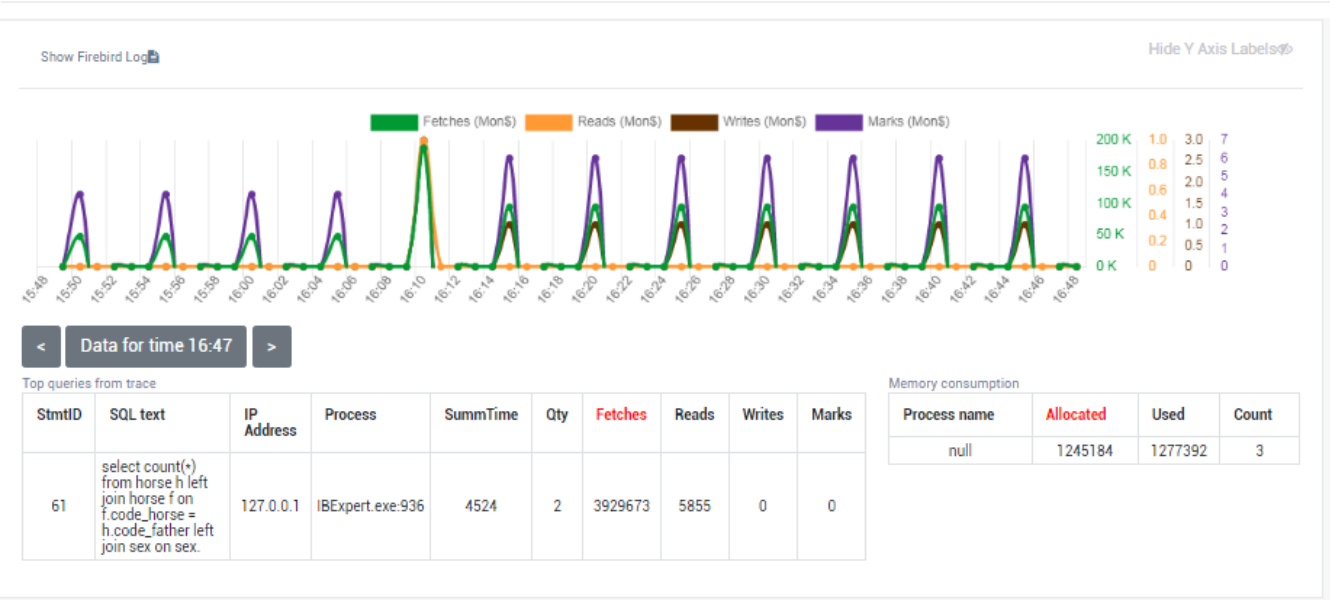
At the top of the page, the name of the database, bookmarks with dates are displayed, as well as the time interval for which the performance counters are displayed. You can change the viewing date and select the desired interval.



The following counters can be displayed graphically:

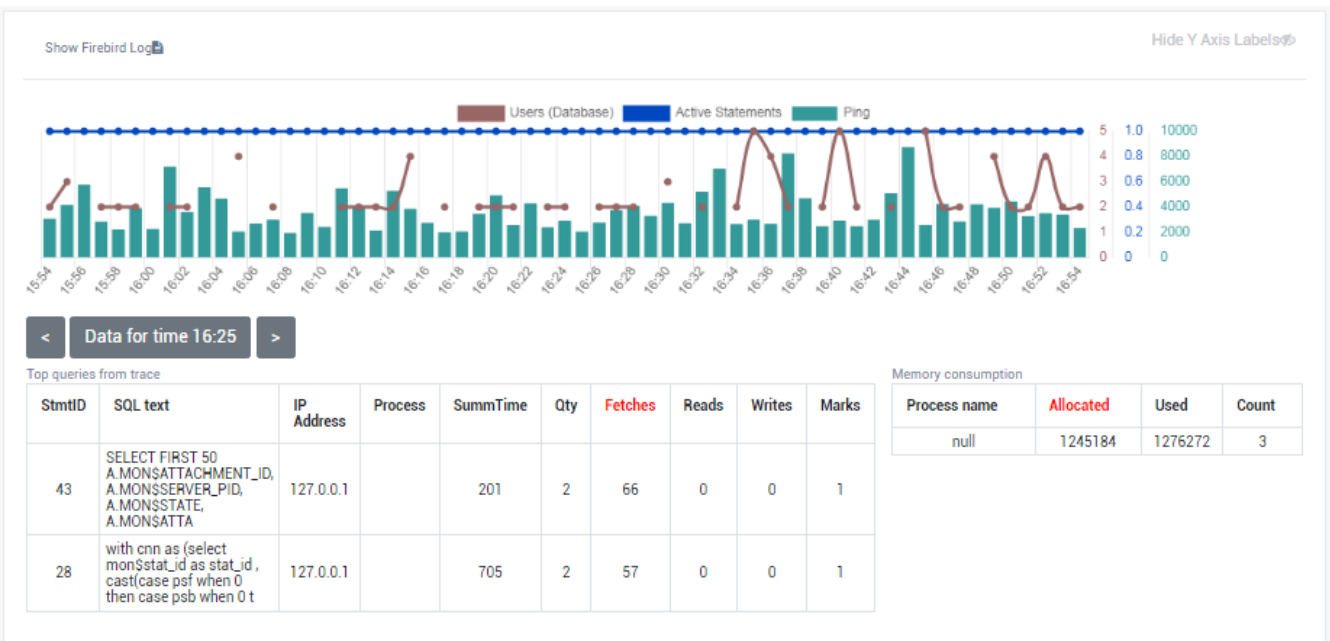
4.3.1. FetchesReadsWritesMarks

The graph displays the performance counters Fetches, Reads, Writes, Marks based on monitoring tables. You can drill down to each time point by clicking on it or selecting "Data for time" from the list.



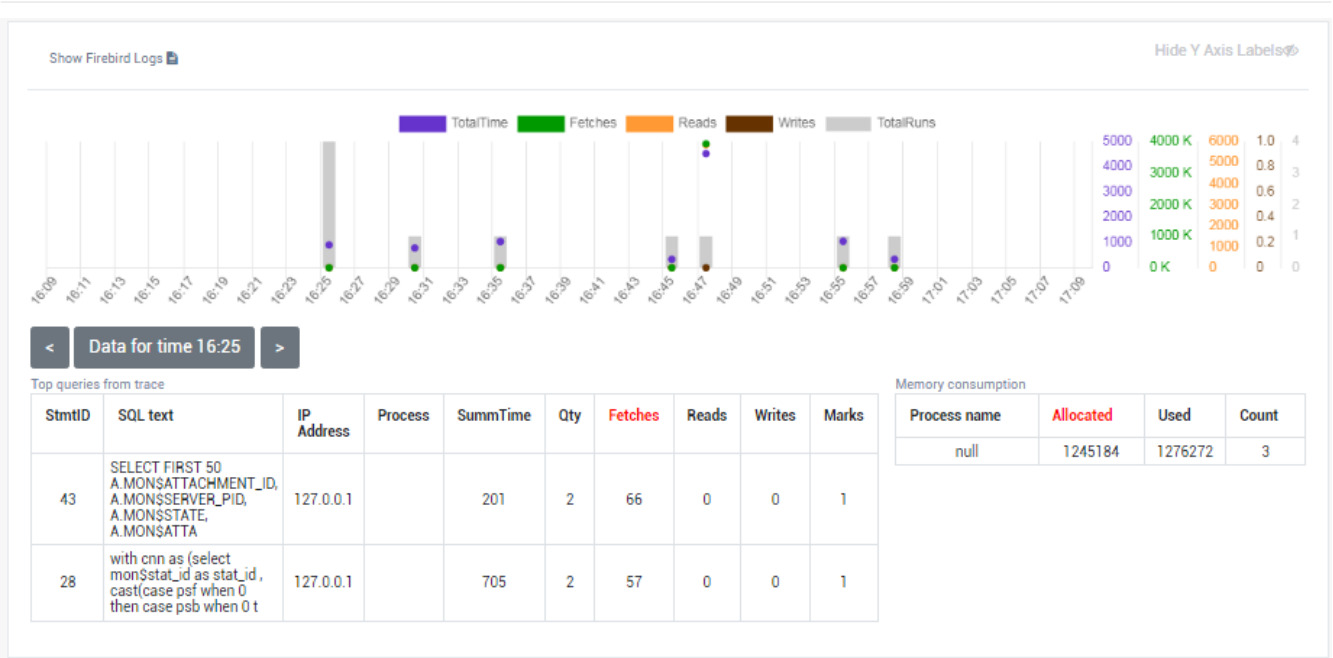
4.3.2. Users

The graph displays the number of active users and requests, as well as the ping time. You can drill down to each time point by clicking on it or selecting "Data for time" from the list.



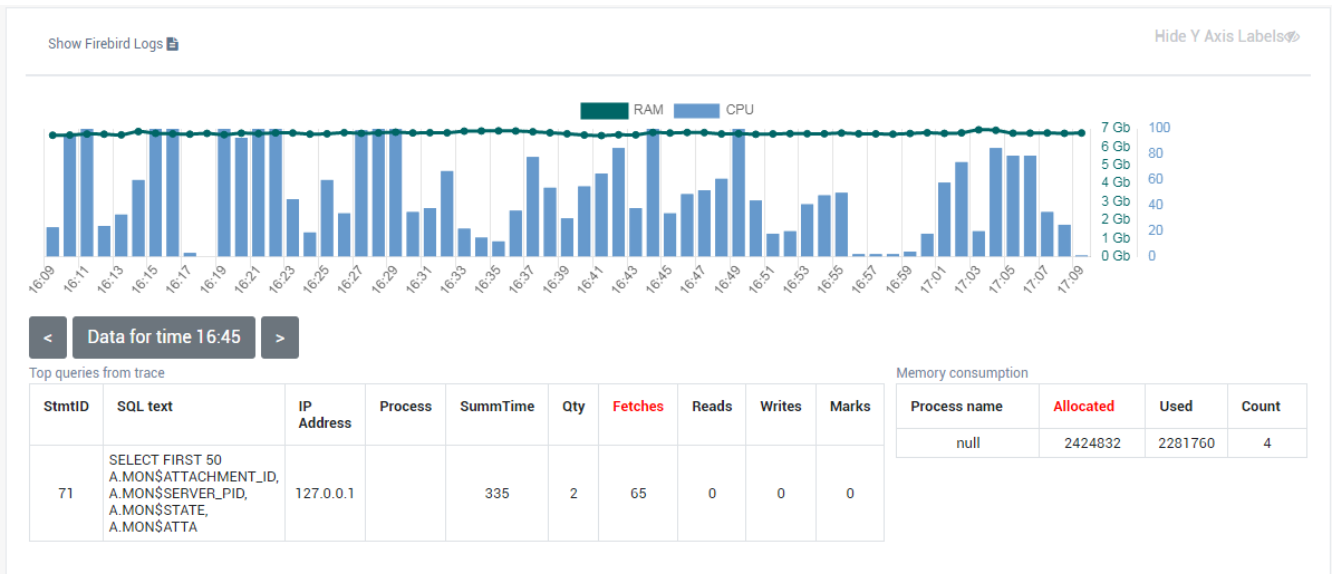
4.3.3. Traces

The graph displays the performance counters Fetches, Reads, Writes, Marks and statement execute time based on data from trace logs. You can drill down to each time point by clicking on it or selecting "Data for time" from the list.



4.3.4. RAM and CPU Windows

The graph displays the consumed memory, as well as the processor load based on tracking by the wmic utility.

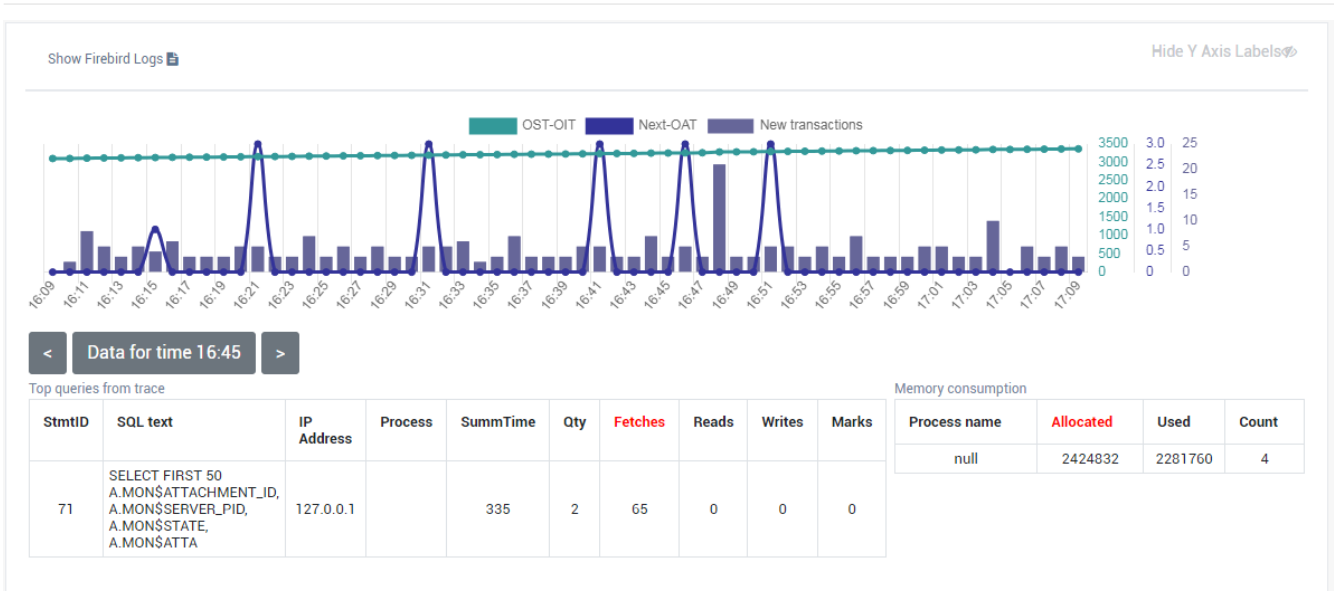


4.3.5. RAM and LoadAvg Linux

The same as "RAM and CPU Windows", only in Linux.

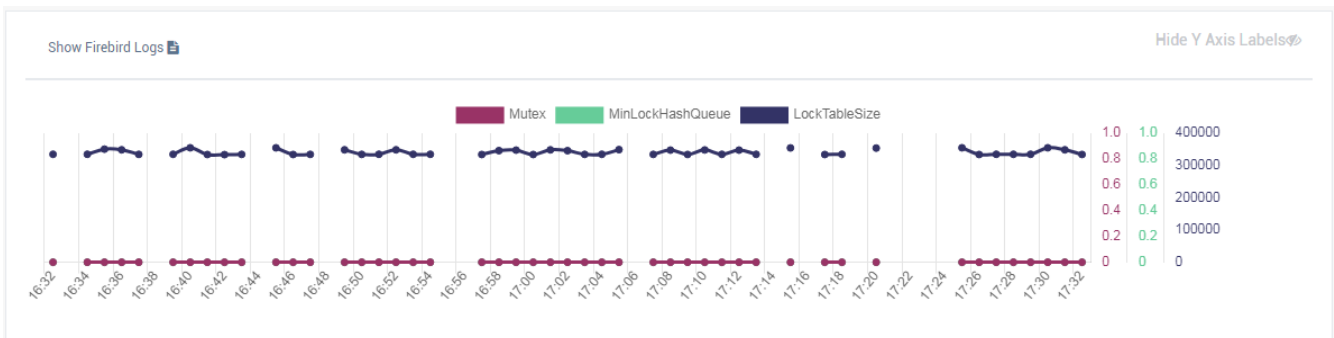
4.3.6. Transactions

The graph displays the number of active transactions and the gap between the counters OST-OIT, Next-OAT.



4.3.7. Lock Table Info

The graph displays data to the load on the lock manager (relevant in Classic and SuperClassic).



4.4. Monitoring with HQbird FBScanner

4.4.1. What is FBScanner?

FBScanner (Firebird Scanner) is a tool included in HQbird advanced distribution of Firebird, which can monitor and view all traffic between Firebird and InterBase servers and their client applications. It shows the real-time activity of connected clients:

- Connections (IP/Name, duration, CPU load),
- Queries (query text, status, parameters)
- Transactions (with parameters).

FBScanner can log all SQL traffic to text files and external Firebird database, it includes FBScanner LogAnalyzer module to analyze SQL performance.

FBScanner can be used to profile database applications, monitor user activity, and manage database connections (including client disconnects on Classic, SuperClassic and SuperServer architectures). It's also ideal for troubleshooting INET errors, as well as auditing existing applications and performance tuning.

FBScanner supports Firebird (V1.x, V2.0, V2.1 and V2.5), InterBase (V4.0 to 2009/XE3). It is a useful tool for analyzing production databases, especially if the application has been developed by third-party and there is no source code available.

FBScanner is transparent as far as the database application is concerned and does not require any changes in application or database source code, logic or configuration.

4.4.2. Issues that FBScanner can help to resolve

- Real-time monitoring of connections. FBScanner shows all connections to the selected database server: the IP/DNS name of connected client, database and connection time.
- Real-time monitoring of SQL queries. For each connection FBScanner shows all the currently running SQL queries along with their transaction parameters.
- Detection of the oldest connection and the oldest active transaction to allow you to analyze that may have non-optimal transaction behavior or incorrect transaction design or show users who might be using the application in a manner that may be affecting performance.
- Client disconnects. Check that disconnections are taking place correctly. You can also use FBScanner to disconnect users in order to perform maintenance or database upgrades.
- FBScanner allows the routing of specific applications or particular users to allow you to zoom in on specific applications or users.
- You can log SQL queries. For debugging or for security FBScanner can log all the selected traffic to a special database for further analysis. FBScanner includes LogAnalyzer tool to find bad queries and ineffective SQL plans.

4.4.3. Performance Impact

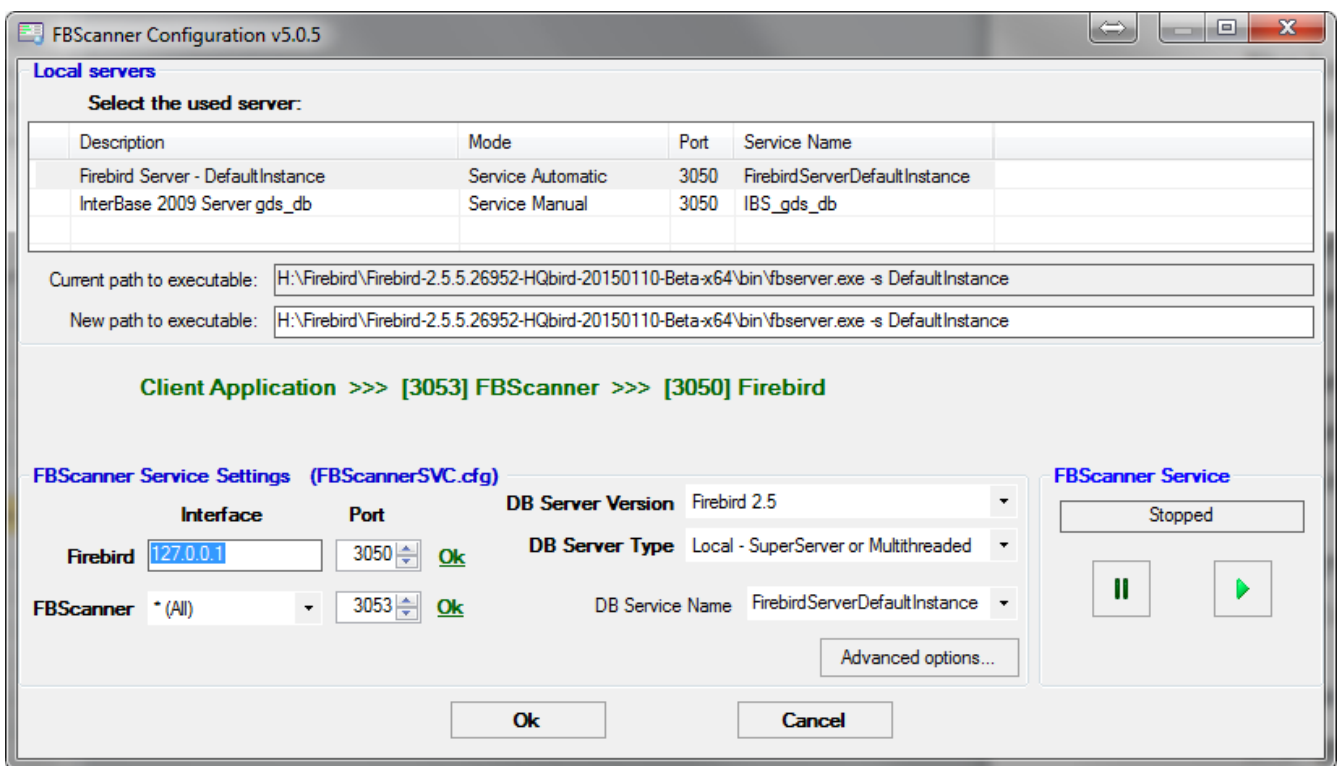
FBScanner does not change anything in transferred SQL traffic and works simply like a transparent proxy, so all applications will work normally.

FBScanner consume approximately 50-150Mb of memory (for 30-100 active clients), it is known that FBScanner adds approximately 150ms for every SQL statement.

4.4.4. How to configure FBScanner for local computer?

To configure FBScanner start “FBScanner Service Settings” from Start menu (*IBSurgeon\HQbird Server Side\Firebird SQL Scanner*). This tool will help you to setup both basic and advanced configuration parameters for FBScanner.

The basic configuration parameters are shown at the main screen of “FBScanner Configuration”. It scans Windows registry for installed Firebird services and show them in the grid.



By default Firebird uses port 3050 for network connections. FBScanner works as a transparent TCP proxy – it redirects all SQL traffic from and to Firebird clients to another.

FBScanner offers to change Firebird port to 3053, in order to start its own instance at 3050. FBScanner checks for the port usage and if either 3050 or 3053 are used by other software (not Firebird), it will warn you with red caption “Port used” near new “Port” text box.

The green figure in the center of “FBScanner Configuration” main screen briefly shows how client applications SQL traffic will be passed.

At the figure below you can see that FBScanner found Firebird 1.5 instance, and offers to change its port to 3053, in order to set own instance to listen at 3050.

Such default scenario will give the maximum compatibility with existing Firebird clients (i.e., end-user applications).

To approve the changes, click “Ok”, otherwise “Cancel”.



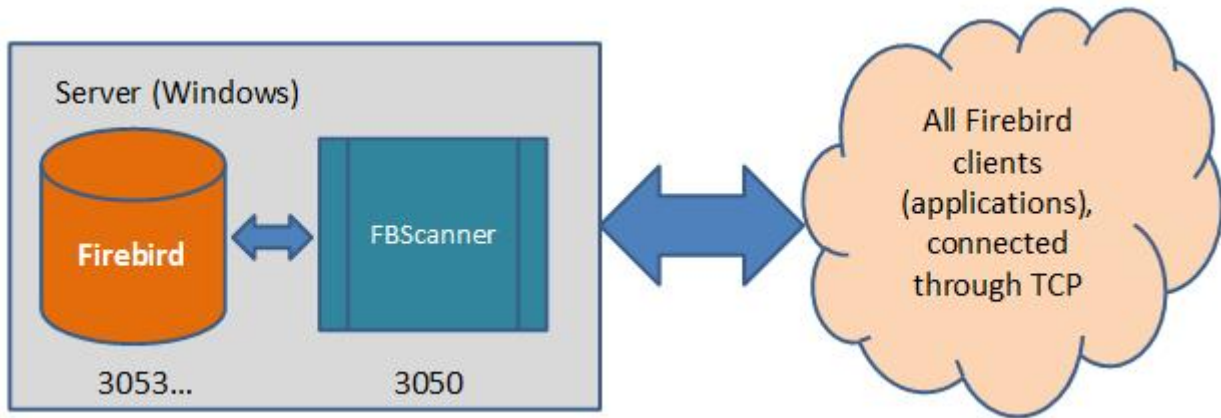
If FBScanner settings were changed, FBScanner Service will be restarted, and all existing Firebird connections will be dropped! Be careful with changing FBScanner settings in production environment. FBScanner will ask your permission to restart, please decide carefully.

4.4.5. How to setup FBScanner for remote computer?

FBScanner can route SQL traffic not only as local proxy, but from another computer too. To understand the difference and discover consequences, let’s walk through details.

The basic (and default) configuration of FBScanner implies that it works on the same computer where Firebird is working, and process all SQL traffic from Firebird clients (i.e., end-user

applications) which use default connection string (and, therefore, port 3050).



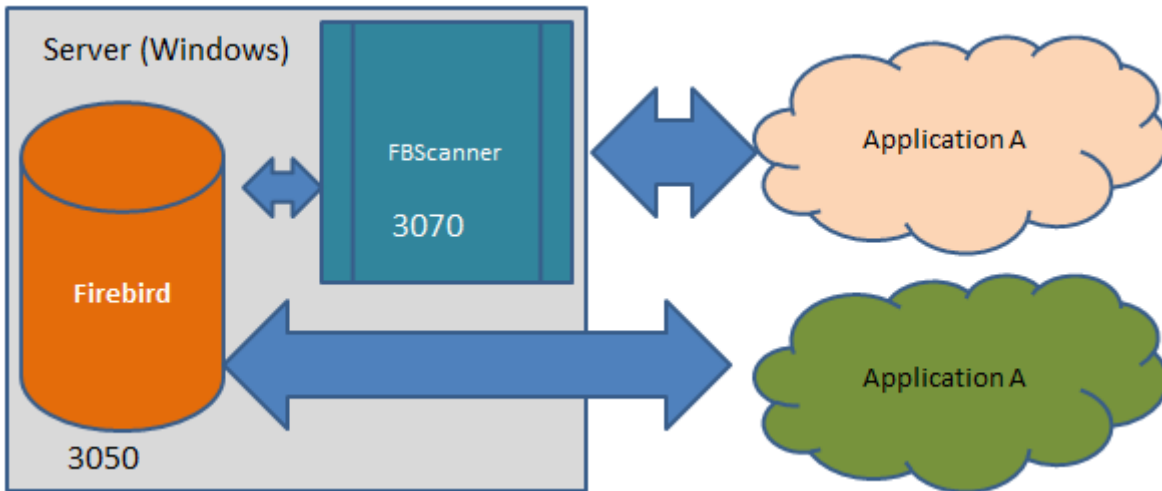
Sometimes it's not convenient to setup FBScanner to process all requests, for example, in case of:

- Only several (may be, the single workstation) workstations need to be profiled/logged
- Only certain application or narrow functionality need to be profiled
- Developers need to check some SQL code on the live database — gather SQL log with execution statistics, plans, etc.
- Heavy load (too many workstations). In case of heavy load FBScanner can consume resources of the main server, and it's better to move FBScanner (as well as FBScanner log, if it's enabled, to the dedicated computer).
- Linux server. If Firebird works on Linux, it's possible to route SQL traffic through remote instance of FBScanner on Windows.

In these cases the good idea is to setup FBScanner at the remote computer and pass only part of SQL traffic through it. It also makes possible to perform necessary analysis of SQL without changing ports or other configuration at server — the only needed adjustment will be change host name in client applications' connections strings.

One of the frequent use cases for setting up FBScanner in remote configuration is using it as debug console for developer computer, so developer can see in real-time (with FBScanner LogViewer) or afterwards (with FBScanner LogAnalyzer) all SQLs from own computer to the Firebird server.

At the figure below you can see how it can look like:



Now let's back to the configuration and see how easy to setup FBScanner to route SQL traffic at the remote computer.

At the bottom of the main screen of “FBScanner Configuration” you can see the following default settings (for Firebird 2.5 example we considered above):

FBScanner Service Settings (FBScannerSVC.cfg)		DB Server Version
Interface	Port	Firebird 2.5
Firebird	127.0.0.1 3050 Ok	DB Server Type
FBScanner	*(All) 3053 Ok	Local - SuperServer or Multithreaded
		DB Service Name
		FirebirdServerDefaultInstance
Advanced options...		

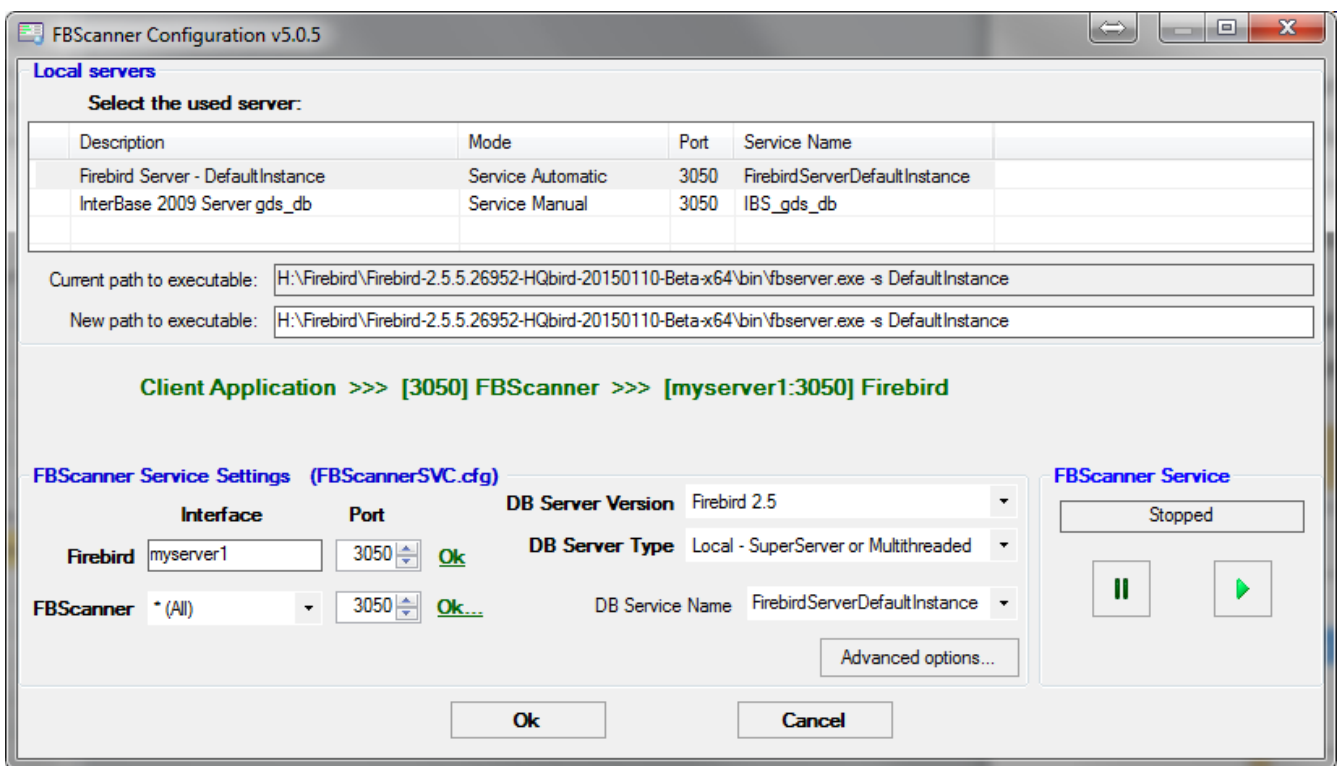
In order to setup FBScanner to route SQL traffic to the remote Firebird, we need to change “Server Type” from “Local...” to “Remote”. It will change the main screen of the configuration tool.

First of all, we need to specify network name (or IP) of the computer with Firebird instance and port where it will be used — it should be entered into “Interface” text box.

Then we need to specify Firebird version – in our example it's Firebird 1.5.

FBScanner instance also has “Interface” — it's the list of network adapters found at the computer. If you need to bind FBScanner to one of them and disable connections from other network adapters, choose one of the adapters from the drop-down list. By default FBScanner will accept Firebird clients' requests from all network adapters.

Below you can see the example of FBScanner configuration to route SQL traffic to remote Firebird instance, which resides on **myserver1** computer and works on default port 3050.



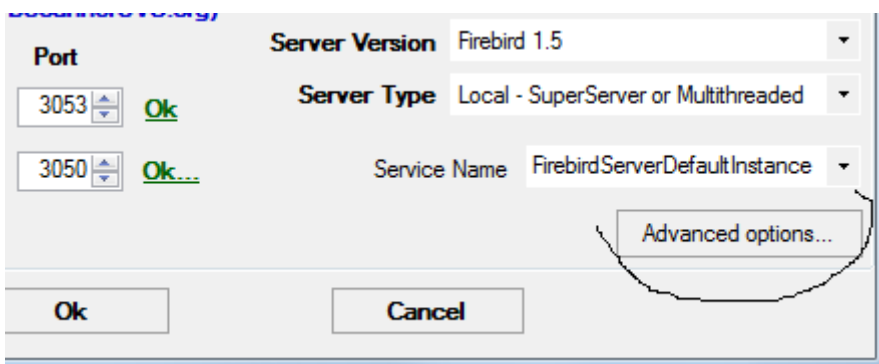
Click “Ok” to confirm new settings, and FBScanner will route SQL requests to the remote Firebird.



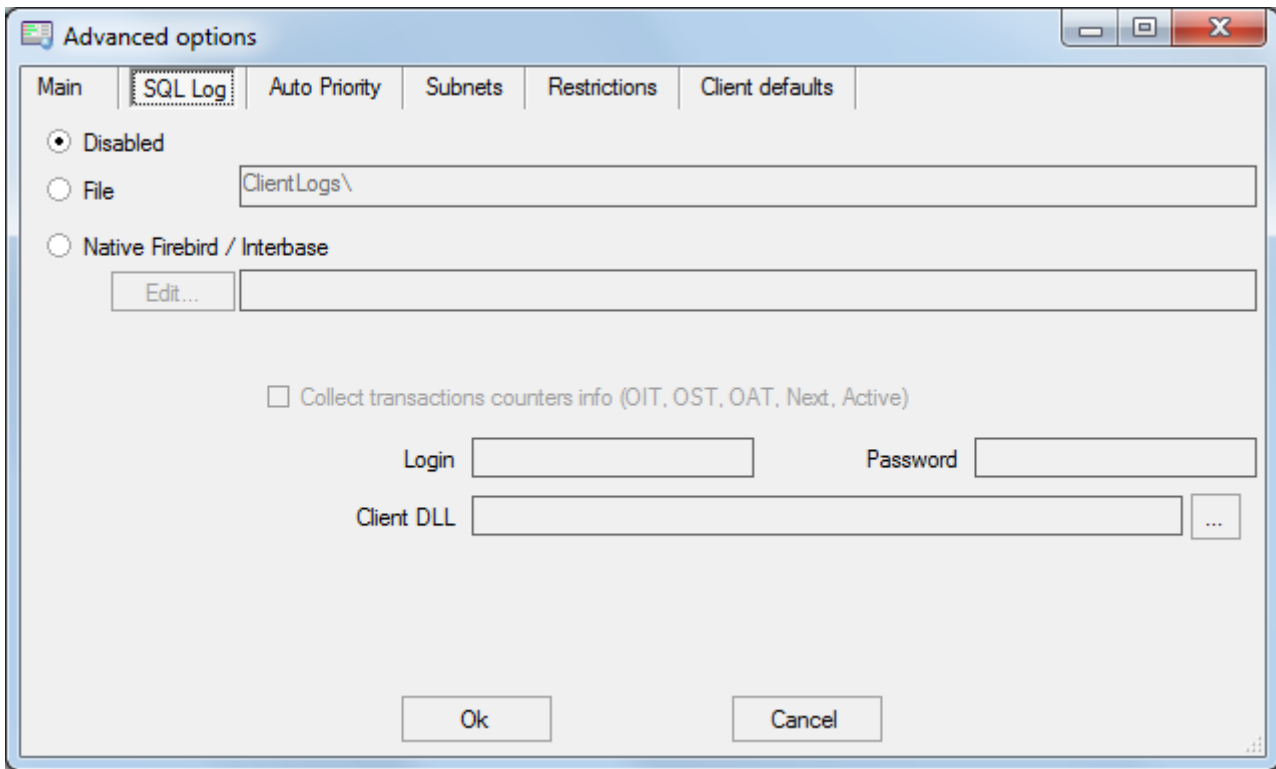
If you need to pass SQL traffic from client applications through remote FBScanner, please change Firebird appropriate connection string. For example, if originally client applications have connected with “**myserver1:C:\Database\data.fdb**”, in order to pass SQL traffic through FBScanner in this example you need to change connection string to “**computer1: C:\Database\data.fdb**” (where computer1 is the network name of the computer where FBScanner works).

4.4.6. How to setup logging?

From Start menu run “Firebird Scanner\FBScanner Settings”, then click button “Advanced options” (in the right bottom of the main screen).



At the dialog click tab “SQL log”.



By default logging is disabled.



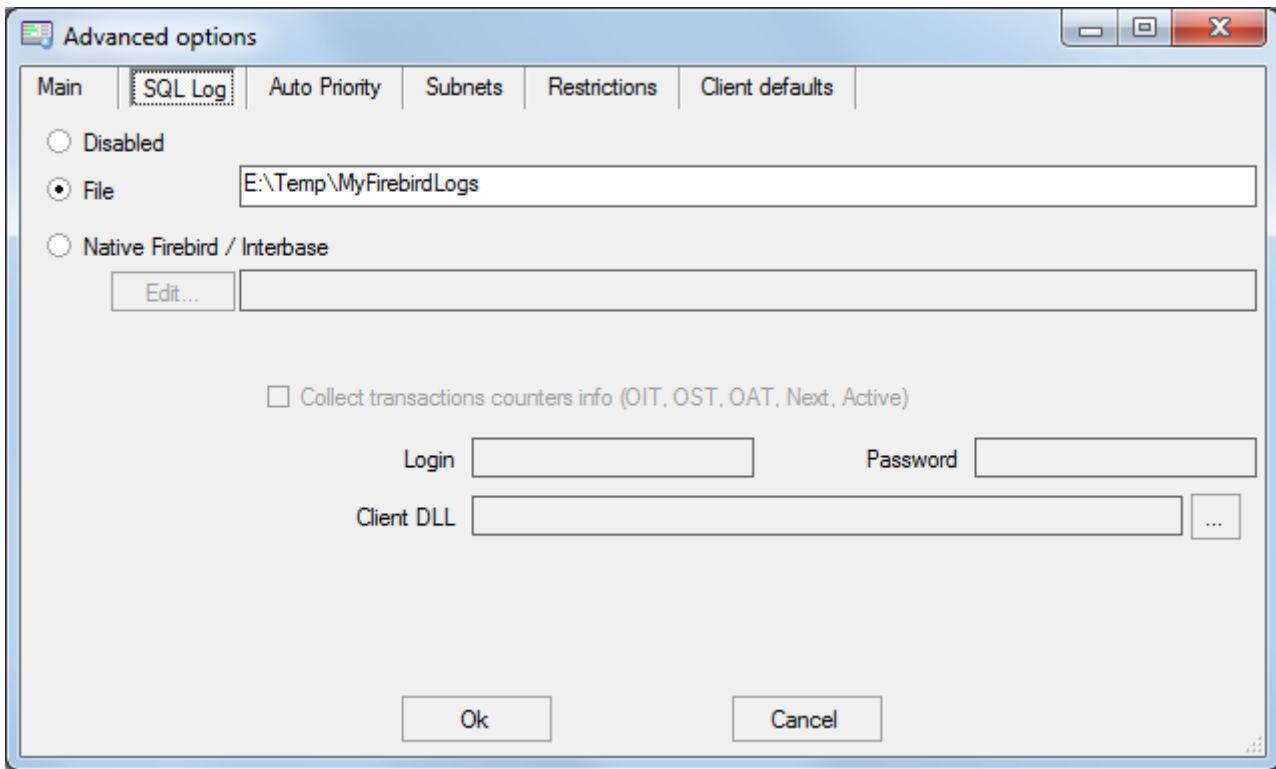
It's important to understand that logging to SQL database will write all SQL operations, including transactions, connects, etc. It means that SQL log database will consume the same amount of resources (CPU, HDD, etc) as the main database does. Due to this fact for heavy load environments we recommend to use remote configuration of FBScanner for SQL logging.

There are 2 options for logging — to file and to Firebird log database.

Logging to text files

File logging creates text file for each connection where FBScanner writes SQL and transactions operators. We recommend file logging for debug purposes and during development — it's suitable to investigate linear SQL code. If there are a lot of connections, file logging becomes not very suitable.

To enable file logging, click radio button near “File” option and set folder where to store file logs (check that specified folder exists first!):



Then click "Ok".



Enabling logging will require restart of FBScanner Service, so all current connections will be dropped. FBScanner will ask your permission to do it immediately.

Example of text file logging

For the following isql commands

Use CONNECT or CREATE DATABASE to specify a database

```
SQL> connect "localhost:E:\Temp\TEST15_2.FDB";
Database: "localhost:E:\Temp\TEST15_2.FDB"
```

```
SQL> create table t1(i1 integer, c1 varchar(150));
SQL> create table t2(i2 integer, b1 blob);
SQL> select count(*) from t1;
```

```
COUNT
=====
0
```

```
SQL> insert into t1(i1, c1) values(1, 'test');
SQL> select count(*) from t1;
```

```
COUNT
=====
1
```

```
SQL> exit;
```

FBScanner created the following log:

```
/* Log created by FBScanner v2.7.19
14.01.2011 16:06:07
    Client IP      = 127.0.0.1
    Client Name    = ibsurgeon3
    Client Process = isql [1884]
*/
CONNECT '127.0.0.1/3053:E:\Temp\TEST15_2.FDB' USER 'SYSDBA';

/* 14.01.2011 16:06:09 */
/* TrID=20; */
SET TRANSACTION READ WRITE WAIT SNAPSHOT;

/* 14.01.2011 16:06:09 */
/* TrID=22; isc_tpb_version1, isc_tpb_write, isc_tpb_read_committed, isc_tpb_wait,
   isc_tpb_no_rec_version */
SET TRANSACTION READ WRITE WAIT ISOLATION LEVEL READ COMMITTED NO RECORD_VERSION;

/* 14.01.2011 16:06:19 */
/* QrID=26 TrID=22; EXECUTE */
create table t1(i1 integer, c1 varchar(150));

/* 14.01.2011 16:06:19 */
/* QrID=26 TrID=22; INFO */

/* 14.01.2011 16:06:19 */
```

```

/* TrID=22; */
COMMIT;

/* 14.01.2011 16:06:33 */
/* TrID=27; isc_tpb_version1, isc_tpb_write, isc_tpb_read_committed, isc_tpb_wait,
   isc_tpb_no_rec_version */
SET TRANSACTION READ WRITE WAIT ISOLATION LEVEL READ COMMITTED NO RECORD_VERSION;

/* 14.01.2011 16:06:33 */
/* QrID=31 TrID=27; EXECUTE */
create table t2(i2 integer, b1 blob);

/* 14.01.2011 16:06:33 */
/* QrID=31 TrID=27; INFO */

/* 14.01.2011 16:06:41 */
/* TrID=32; isc_tpb_version1, isc_tpb_write, isc_tpb_read_committed, isc_tpb_wait,
   isc_tpb_no_rec_version */
SET TRANSACTION READ WRITE WAIT ISOLATION LEVEL READ COMMITTED NO RECORD_VERSION;

/* 14.01.2011 16:06:41 */
/* QrID=36 TrID=20; EXECUTE */
select count(*) from t1;

/* 14.01.2011 16:06:41 */
/* QrID=36 TrID=20; INFO */

/*
   Fetch count      = 1
*/

/* 14.01.2011 16:07:11 */
/* QrID=38 TrID=20; EXECUTE */
insert into t1(i1, c1) values(1, 'test');

/* 14.01.2011 16:07:17 */
/* QrID=40 TrID=20; EXECUTE */
select count(*) from t1;

/* 14.01.2011 16:07:17 */
/* QrID=40 TrID=20; INFO */

/*
   Fetch count      = 1
*/

/* 14.01.2011 16:07:26 */
/* TrID=32; */
COMMIT;

/* 14.01.2011 16:07:26 */

```

```

/* TrID=27; */
COMMIT;

/* 14.01.2011 16:07:26 */
/* TrID=20; */
COMMIT;

```

As you can see, file log is useful to understand how SQL commands were run inside the single connect.

Logging to Firebird database

Before you start with SQL log, it's necessary to understand some implementation details, which can be important for production systems.

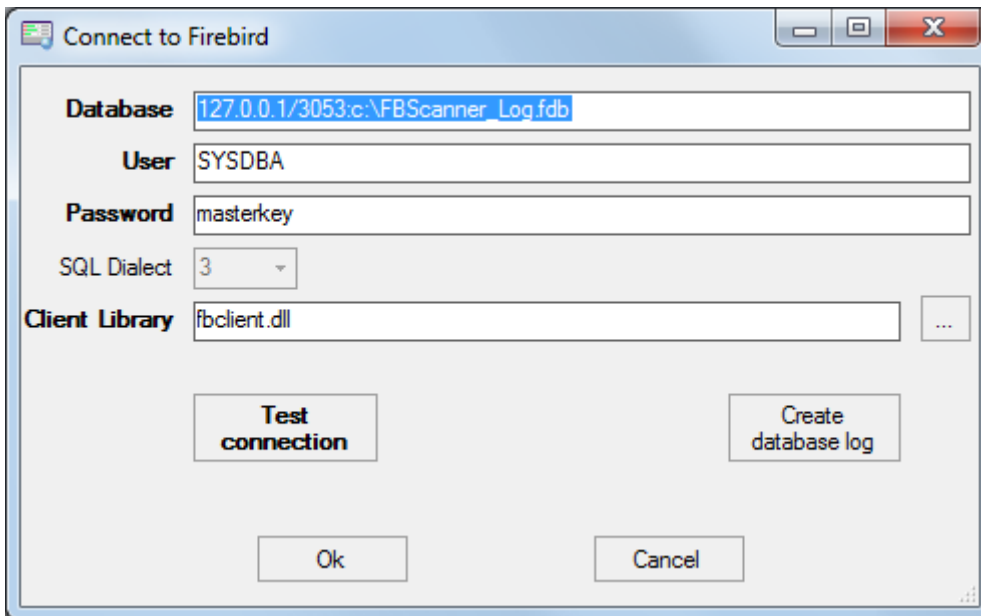
In general logging to Firebird database is implemented in the straightforward way: FBScanner service writes all traffic to the external Firebird database. Firebird database with log can be at the same computer where FBScanner resides, or at the remote computer.

Please consider the following requirements for SQL log configuration:

- Log database (and appropriate Firebird instance) should be in Firebird 2.5 format (since FBScanner 2.7.15). If you are forced to use FBScanner at the computer with another Firebird version, you need to use embedded Firebird 2.5 to store log.
- SQL traffic from all logged connections is written into the single table, with appropriate markers (from what computer, application, user, etc. this particular record was created).
- Log database can consume significant amount of resources in case of heavy load. For many connections it's recommended to setup FBScanner and Firebird log database at dedicated computer.
- In many cases it's not necessary to log all connections, because they repeat the same set of SQL queries. Careful investigation of the single connection can be the most effective way to find performance problems.

To enable SQL logging, click on "SQL" radio button. It will enable appropriate text boxes and controls.

First of all, click button "Edit".



If you intend to use the same Firebird instance to log SQL traffic, you need to specify connections string with explicit and direct port. In our example it will be port 3053, and connection string looks like **127.0.0.1/3053:C:\FBScanner_log.fdb**

In this dialog you also need to specify how to connect to database with log.

If there is no database with specified name, create new database — click “Create database log”.

Test connection with log database — click “Test connection”.

Click “Ok” to save settings.

Transactions markers

FBScanner can gather information about transactions markers (in the same way like IBSurgeon Transaction Monitors does). Gathered information will be shown as graphs in FBScanner Log Analyzer.

For this purpose, FBScanner runs separate connect, which requires Login, Password and path to the appropriate client dll (if you track Firebird 1.5 with FBScanner, *fbclient.dll* from 1.5 will be required).

If you decide to gather transactions markers information, mark checkbox “Collect transactions counters info” and fill out Login, Password and Client DLL fields.

Using Embedded Firebird 2.5 for SQL log

If you need to use SQL log at the computer where old Firebird is used (1.0, 1.5, 2.0, 2.1 or even InterBase), it's recommended to use Firebird 2.5 Embedded to store log.

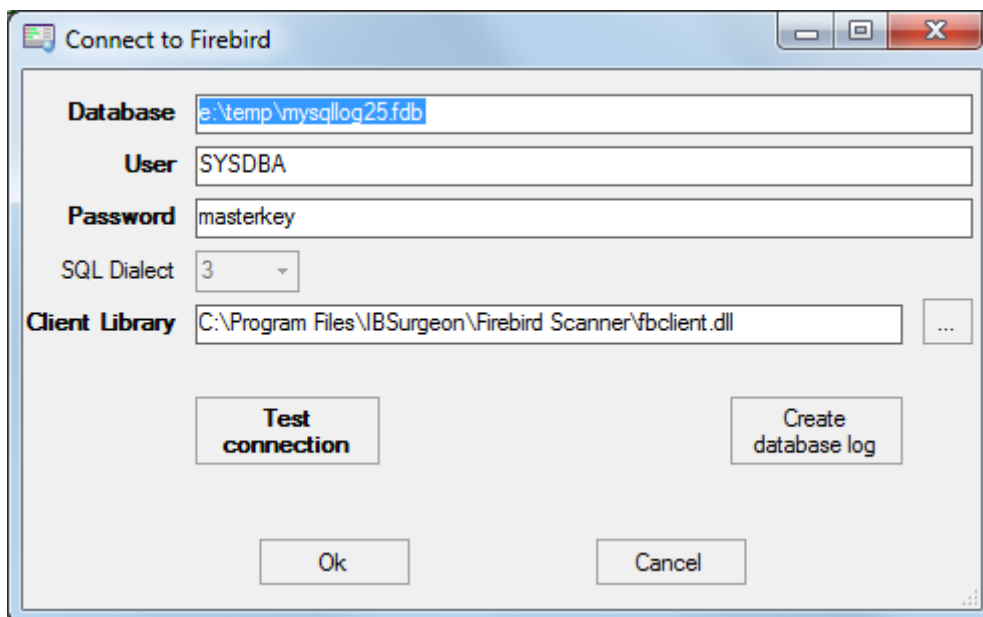
You can download Firebird 2.5 Embedded from www.firebirdsql.org.

Unpack the archive right into the FBScanner folder (*C:\Program Files\IBSurgeon\Firebird Scanner* by default) and rename *fbembed.dll* into *fbclient.dll*.

Folder structure will look like this

doc	14.01.2011 17:25
en-US	14.01.2011 13:40
intl	14.01.2011 17:25
it-IT	14.01.2011 13:40
pt-BR	14.01.2011 13:40
RealtimeConfig	14.01.2011 15:46
ru-RU	14.01.2011 13:40
udf	14.01.2011 17:25
aliases.conf	23.09.2010 16:16
error.log	12.09.2010 0:40
fbclient.dll	17.09.2010 12:15
FBSscanner.log	14.01.2011 16:03
fbscanner.subnets	14.01.2011 17:26
FBSscanner_RegInfo.xml	14.01.2011 14:15
FBSscannerCFG.exe	09.12.2010 0:47
FBSscannerCFG.InstallState	14.01.2011 13:40

After that run “Advanced options”, tab “SQL logging”, radio button “SQL” and click “Edit”, then in the “Client library” point to the renamed *fbclient.dll*, as it shown below.



In Embedded Firebird *fbclient.dll* represents the whole engine. It works inside the process of *FBSscanner* and there is no interaction with other installed Firebird instances, both full and embedded.

4.4.7. How to analyze *FBSscanner* log?

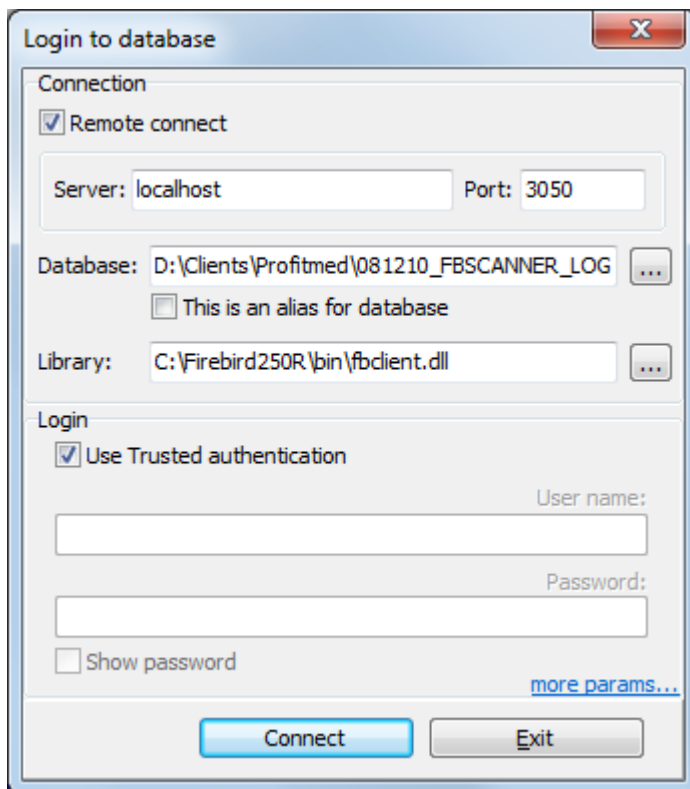
Many users told us that they did not realize how many queries, transactions and other operations are performed by their software. As you remember, *FBSscanner* stores all information into the single table. It uses self-links to reduce the amount of stored information and it makes raw log hard to read and understand.

To facilitate log analysis we have created new module in FBScanner — LogAnalyzer. It's available in IBSurgeon Deploy Center for all FBScanner users (inside “Download” section).

LogAnalyzer requires Firebird 2.5 to work with log database. It also creates new indices and runs heavy reporting queries, so it's recommended the following procedure:

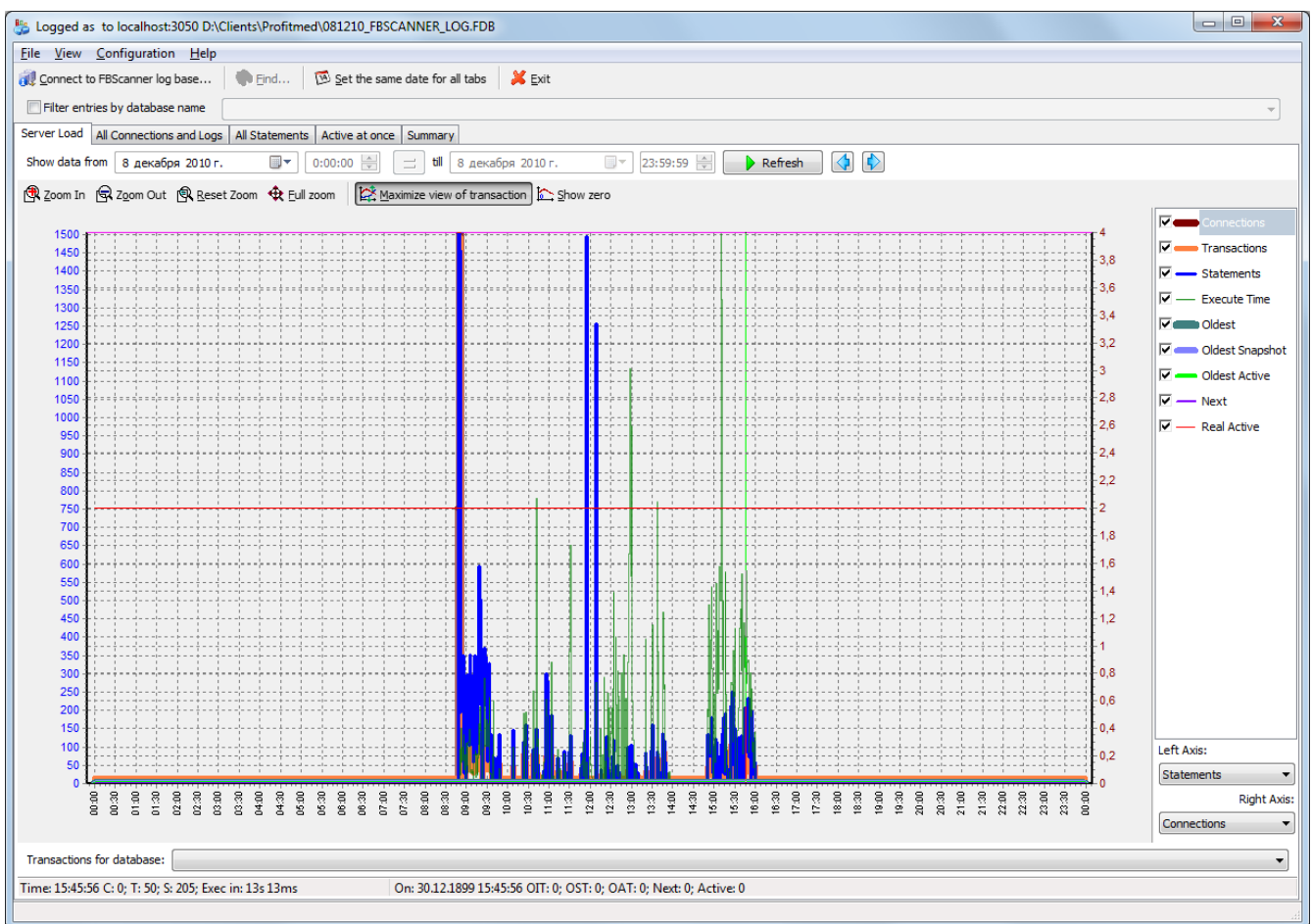
1. Setup logging and gather statistics for at least 1 day
2. Copy log database to another computer with Firebird 2.5
3. Connect to the copy of log database and perform analysis at the developer's computers
4. Copy updated versions of log databases as necessary

To analyze log database, start LogAnalyzer and click “Connect to FBScanner log base”, then fill out connection parameters and select log database.



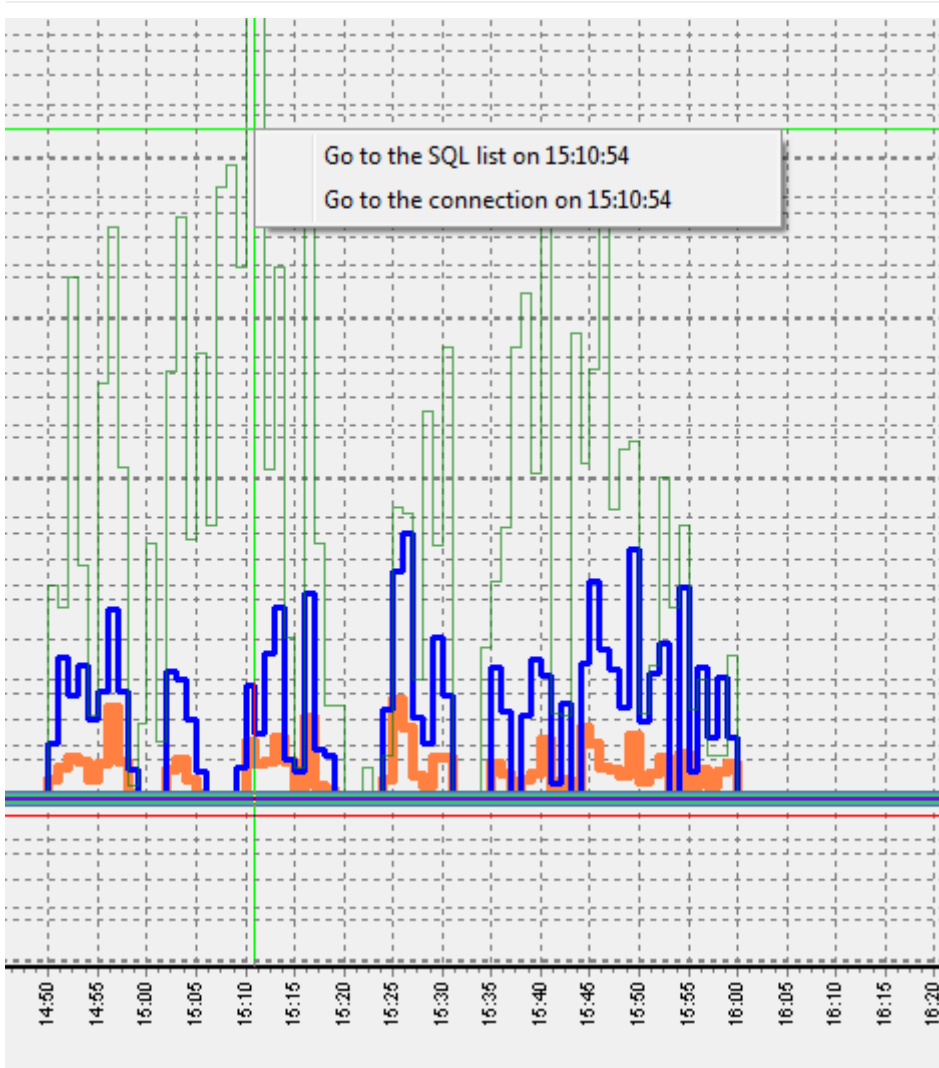
At first start LogAnalyzer will create necessary indices, it can take several minutes.

After that LogAnalyzer will show the last available day in the log at the “Server Load” tab:



“Server Load” tab shows how many SQL queries were run per minute, and how much time they took to execute. Effectively it shows server load, i.e., number of queries and their execution times.

Zoom in (button in the top left corner of the tab “Server load”), drag graph by holding right-button of the mouse and select the peak you are interested to investigate — click right-button to show popup-menu



It will show you tab “All statements”, where you can browse SQL queries

211232	186212	186215	17299	08.12.2010 15:11:0	0	0	08.12.2010 15:11:0	select d.*, dt.*, d1.xec_5034	select d.*, dt.*, d1.xec_5034	PLAN (LIN INDEX (LIN_EIDDOC	1	0	08.12.2010
211256	186212	186215	17300	08.12.2010 15:11:0	0	437	08.12.2010 15:11:0	SELECT	SELECT	PLAN JOIN (XIS1101 INDEX (I	47	0	08.12.2010
211235	186212	186215	17301	08.12.2010 15:11:0	0	0	08.12.2010 15:11:0	SELECT	SELECT	PLAN (XECINT INDEX (IDX_XEC	1	0	08.12.2010
211236	186212	186215	17303	08.12.2010 15:11:0	0	0	08.12.2010 15:11:0	SELECT o\$database_kind		PLAN (RDB\$DATABASE NATUR	1	0	08.12.2010


```

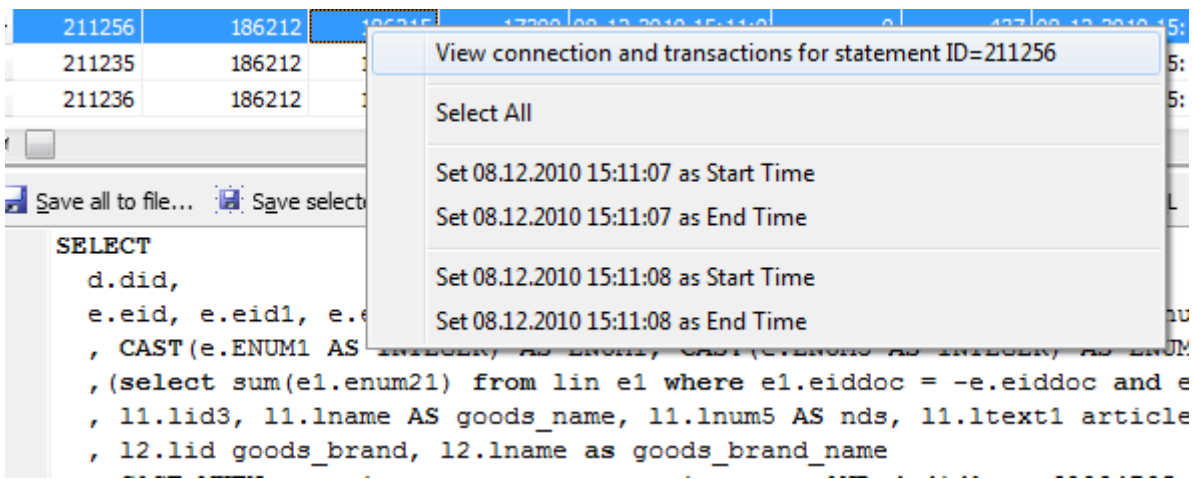
SELECT
  d.did,
  e.eid, e.eid1, e.eid2, e.eid3, e.eid4, e.eid5, e.eid6, e.enum4, e.enum14, e.eint2
, CAST(e.ENUM1 AS INTEGER) AS ENUM1, CAST(e.ENUM3 AS INTEGER) AS ENUM3, e.enum9, e.enum15, e.e
, (select sum(e1.enum21) from lin e1 where e1.eiddoc = -e.eiddoc and e1.eid1 = e.eid1) add_sum
, 11.lid3, 11.lname AS goods_name, 11.lnum5 AS nds, 11.ltext1 article
, 12.lid goods_brand, 12.lname as goods_brand_name
, CASE WHEN pr.prime_cost_opt > pr.prime_cost AND d.did1 <> 60034705 THEN pr.prime_cost_opt EL
, 12.lid1
, 14.lname AS state_name
, (SELECT f3 FROM percent (e.enum4,e.enum5)) p_price
, e.enum1 * e.enum4 AS enum14_ent
, CAST(0 AS INTEGER) AS selected
, m.managername
  
```

```

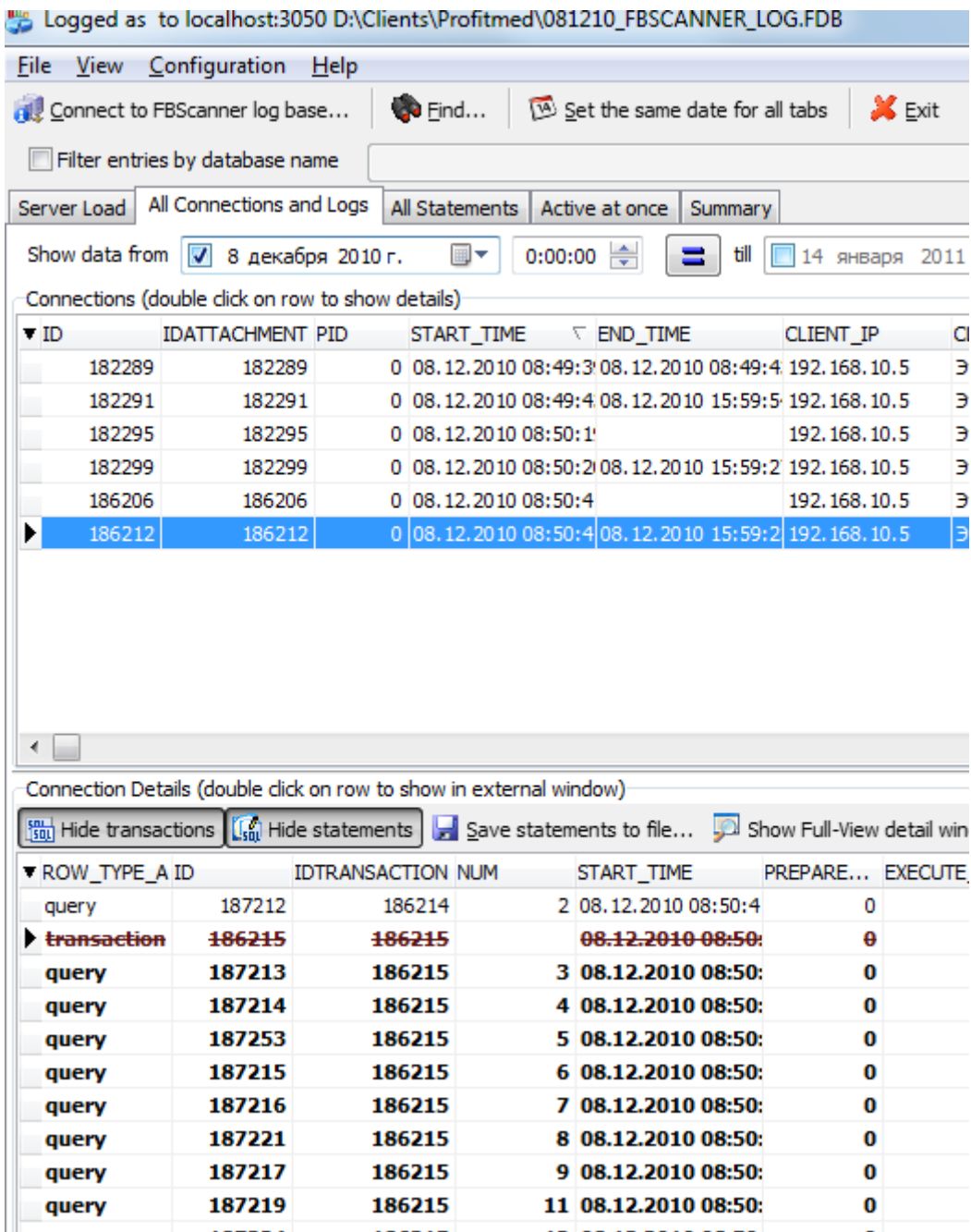
PLAN JOIN (XIS1101 INDEX (IDX_XECINT5), LIB INDEX (R
PLAN (LIB INDEX (RDB$PRIMARY11))
PLAN JOIN (XIS1100 INDEX (IDX$XECINT_OWNER_PARENT_NO
PLAN (DLAST INDEX (RDB$PRIMARY5))
PLAN (LIN INDEX (LIN_EIDDOC_EID1_EID3_EID5))
PLAN (XECINT INDEX (IDX_XECINT5))
PLAN (XECNUM INDEX (IDX_XECNUM4))
PLAN (LIB INDEX (RDB$PRIMARY11))
PLAN (PERCENT NATURAL)
PLAN (E1 INDEX (LIN_EIDDOC_EID1_EID3_EID5))
PLAN JOIN (JOIN (JOIN (JOIN (JOIN (JOIN (JOIN (D IN
  
```

Select any query to see its text and, if plan logging feature is enabled, its plan.

To follow the execution flow, you can right-click on the query and look for connection and transactions for this query



LogAnalyzer marks bold queries in the same transaction:



You can sort queries and, for example, find query with the longest execution time:

▼ ROW_TYPE_A ID	IDTRANSACTION NUM	START_TIME	PREPARE...	EXECU...	END_TIME
query	207407	186215	14361 08.12.2010 12:57:	0	32594 08.12.2010
query	211270	208928	17338 08.12.2010 15:11:2'	0	30624 08.12.2010 1
query	207518	201265	14539 08.12.2010 13:00:0'	0	14766 08.12.2010 1
query	207372	201225	14392 08.12.2010 12:58:0'	0	14437 08.12.2010 1
query	214522	186215	19618 08.12.2010 15:43:	0	14109 08.12.2010
query	214109	186215	19176 08.12.2010 15:37:	0	13672 08.12.2010
query	211668	186215	17139 08.12.2010 15:09:	0	12470 08.12.2010
query	208263	201460	15296 08.12.2010 13:37:5'	0	11702 08.12.2010 1
query	207597	186215	14544 08.12.2010 13:00:	0	10937 08.12.2010

To know more about this query — double-click on it and see more details

The screenshot shows a window titled "SQL" with a toolbar containing "Copy SQL", "Copy Plan", and "Close this window". Below the toolbar are tabs for "Plain SQL", "Unprepared SQL", and "Info". The main area displays a SQL query:

```

SELECT  d.did,d.dcode,d.ddate1,d.ddate3,d.ddate4,d.did1,d.did6,d.did8,d.did10
        , d.dstate,d.date,d.dparent,d.dnum1,d.dnum2,d.dnum5,d.dnum8,d.dint2,d.dde
        , d.dnum9
        , l.lname, l.ltext1
        , (SELECT FIRST 1 rf2_b_line(val,1) FROM xecblob WHERE id = dt.address) a
        , (SELECT lname FROM lib WHERE lid = d.did8) trend
        , (SELECT kname FROM cfg WHERE kid = d.dstate) state_name
--      , (SELECT lname FROM lib WHERE lid = d.did9) manager_name
        , lib2800.lname as manager_name
        , lib5800$1.lname as dolgnost
        , lib5800$2.lname as gruppa
        , lib5800$3.lname as otdel
        , (SELECT lname FROM lib WHERE lid = d.DAUTHOR) author_name
        , CASE WHEN dt.day_in_day = 2201 THEN 'Да' ELSE 'Нет' END day_in_day_name
        , CASE WHEN dt.in_night = 2201 THEN 'Да' ELSE 'Нет' END in_night_name

```

Below the query is the execution plan:

```

PLAN (XECINT INDEX (IDX_XECINT5))
PLAN (XECINT INDEX (IDX_XECINT5))
PLAN (XECDATE INDEX (IDX_XECDATES5))
PLAN JOIN (XECINT INDEX (IDX_XECINT5),PMTWERTREE INDEX (PK_PMTWERTREE))
PLAN (X770100 INDEX (IDX_XECINT5))
PLAN JOIN (XI1 INDEX (IDX_XECINT5),XD1 INDEX (IDX_XECDATES5))
PLAN JOIN (JOIN (XI2 INDEX (XECINT_IDX2),XV1 INDEX (RDB$PRIMARY42)),XD1 INDEX

```

On the right side of the window, there are buttons for "Plan" and "Error Message".

4.4.8. How to track 10054 errors, disconnects and failed login attempts?

FBScanner automatically logs all 10054 errors, disconnects and failed login attempts with detailed description in the *FBScanner.log* file, which is in FBScanner main directory.

```
19.08.2010 21:43:09
```

```
Connect Error
```

```
Client IP      = 192.10.1.2
Client Name    =
DB Name       =
DB User       = MORTON
Client Process = SUPC [5520]
Client Process (by fbclient) = E:\TEMP\TEST1.EXE [5520]
STATUS        = [file is not a valid database]
```

```
19.08.2010 21:43:25
```

```
Login Failed
```

```
Client IP      = 127.0.0.1
Client Name    = ibsurgeon3
DB Name       = C:\Program Files\Jupiter2010\Data\data.gdb
DB User       = MORTON
Client Process = Jupiter.exe [3032]
Client Process (by fbclient) = E:\TEMP\TEST1.EXE [3032]
STATUS        = [Your user name and password are not defined.
```

```
Ask your database administrator to set up a Firebird login.]
```

4.4.9. Backup/restore and mass load operations

To perform operations which do not require monitoring or debugging, like backup and restore or mass load of records (in billing systems) we recommend bypassing FBScanner service.

If FBScanner is installed in default recommended configuration, i.e., on port 3050 and Firebird is on port 3053, connection strings should be like this

```
server_name/3053:Disk:\Path\database.fdb
```

example of connection string

```
connect "localhost/3053:C:\TEMP\database.fdb" user "SYSDBA" password "masterkey";
```

Example of using backup command

```
gbak.exe -b -g -v -user SYSDBA -pass masterkey
localhost/3053:C:\TEMP\database.fdb C:\temp\backup.gbk
```

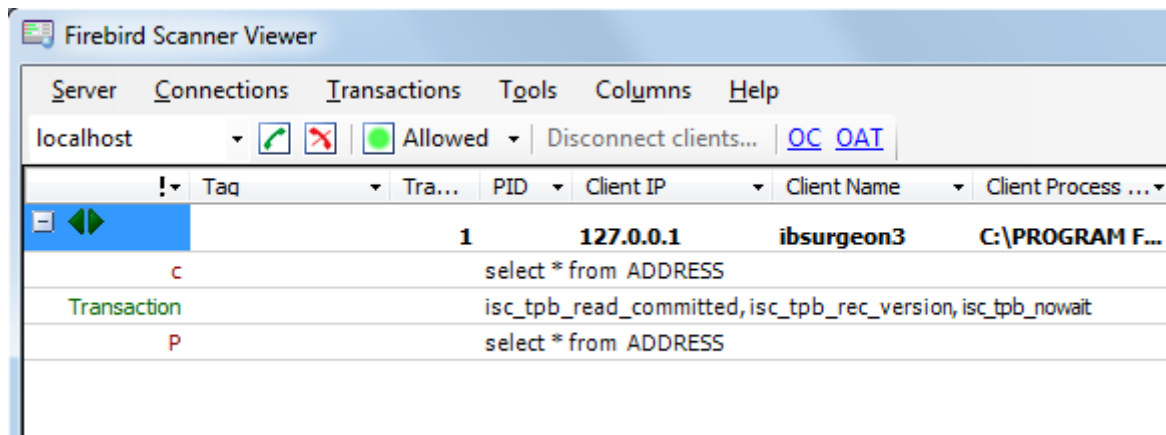
and, of course, using local connection string will always bypass FBScanner:

```
gbak.exe -b -g -v -user SYSDBA -pass masterkey
C:\TEMP\database.fdb C:\temp\backup.gbk
```

4.4.10. Real-Time Monitoring: FBScanner Viewer

To monitor connections, queries and transaction in real-time FBScanner includes special tool namely FBScanner Viewer.

FBScanner Viewer shows momentary snapshot of SQL traffic between Firebird and monitored client applications.



In the first column we can see type of record — connection, statements or transaction.

In the table below you can find description of all columns at main page of FBScanner Viewer (some columns are hidden by default, use menu Columns to turn them on/off):

Column title	Column description
! (first column)	<p>Indicates type of record in FBScanner Viewer — there are separate set of values for SQL statements, transactions and connections. They are described in the next table below.</p> <p>Sign “!” in the title of this column means active filter — click on the triangle at right side of sign “!” to adjust it.</p>

Column title	Column description
Tag	<p>Green/red background shows CPU Usage in % (red — Kernel, green — Firebird).</p> <p>Text is shows tags value (if it was specified in SQL query).</p> <p>Example how to set tag values: [source] ---</p> <pre>SELECT * FROM RDB\$DATABASE /*FBSCANNER\$CON_NAME=MyConnect; FBSCANNER\$TR_NAME=MyTransaction; FBSCANNER\$ST_NAME=SomeImportantQuery; */; ---</pre> <p>Also in this column you will see execution of gbak and gfix tools.</p>
Transaction Count	<p>Applicable for connection row. Number of active transactions in the connection is shown.</p> <p>It's very useful to find applications with auto-commit and other ineffective transaction management issues.</p>
PID	Process ID for Firebird. Only for Classic Architecture
Client IP	IP of connection
Client Name	DNS of connection (if possible to resolve)
Client Process Name	<p>Starting from Firebird 2.1, <i>fbclient.dll</i> shows name of client application. For example,</p> <p><i>C:\Program Files\Firebird\Firebird_2_1\bin\isql.exe</i></p>
Priority	Priority of Firebird instance (Classic only)
Database	Database name or its alias, as it appears in the connection string
User	Users name — for example, SYSDBA (it does not supported Trusted Authentication)
Role	Role of user
Start	For connection row — connection time, for transaction — start time of transaction, for statement — query start time.
Time	'NOW' — Start; Time from the start moment

Column title	Column description
Last Activity	Time of last action for current connect/transaction/statement.
Inactive	'NOW' — Last Activity; Period of inactivity
Latest Retaining	Time of the most recent “COMMIT RETAINING” or “ROLLBACK RETAINING” in the current transaction
Retaining	'NOW' — Latest Retaining
Received	Bytes, received by client
Sent	Bytes, sent by client
CPU Time	Shows overall time consumed in connection/transaction/query. If there is more than 1 query in transactions, execution time of all queries will be summarized. The same rule is for connection time calculation.
Prepare Time	Execute Time
Fetch Count	Applicable only for statements. Number of rows, as it's reported by <i>fbclient.dll</i>
Protocol	Firebird protocol version for current session.
Version	Version of <i>fbclient.dll/gds32.dll</i> . Version detection is not 100% correct: minor versions are considered as the same, JayBird and .NET Provider are considered as the same, InterBase 8.x = InterBase 9.x

In the following table you can see details for the values appeared in the first column in FBScanner Viewer for SQL statements rows:

Flag	Description
A	Allocated. Initial phase of SQL query life cycle
P	Prepared. Indicates that statement was prepared
E	Execute. Query is being executing at the moment
C	Closed statement. Execution is finished
D	Dropped statement.
F	Fetching is in progress
f	Fetching is in progress, but suspended at the moment (recordset is not fetched)
c	Closed cursor. All data was fetched.

Tags

Tags allow assigning readable identifiers (names) to Connections, Queries and Transactions. You just need to add these commentaries:

```
SELECT COUNT(*) FROM RDB$DATABASE
/* FBSCANNER$CON_NAME=My_application;
   FBSCANNER$TR_NAME=Read_only_transaction_N1;
   FBSCANNER$ST_NAME=Customers_list_query; */
```

- **FBSCANNER\$CON_NAME=** sets the name of connection. After the first assignment this name will be kept during the whole connection life.
- **FBSCANNER\$TR_NAME=** sets the name of transaction. After the first assignment this name will be used during the whole life of transaction.
- **FBSCANNER\$ST_NAME=** sets the name of query.

Tags are showed in the first column in FBScanner Viewer grid, and it's possible to filter tags by their names.

Tags are useful to quickly answer the following frequent questions:

What program has launched this query? (developers need to mark with **FBSCANNER\$CON_NAME** tag each database connection)

What is the transaction for this query? (developers need to use **FBSCANNER\$TR_NAME** tag to mark transactions)

- What is this very long query? (developer can mark long queries with readable names like "Annual report").

FBScanner Viewer Menu

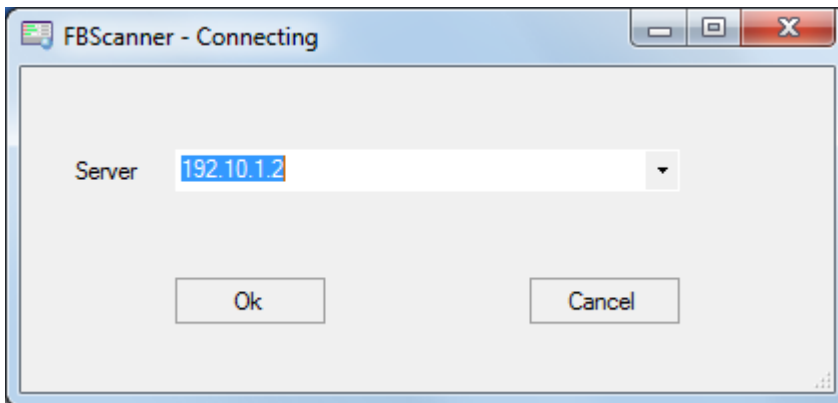
FBScanner Viewer offers wide range of options to make debugging and optimization easier, which are accessible through its menu:

- **Server**
 - Connect To
 - Disconnect To
 - Recent Servers
 - Exit
- **Connections**
 - Disconnect
 - Disconnect Clients...
 - Kill Process
 - Latest Queries

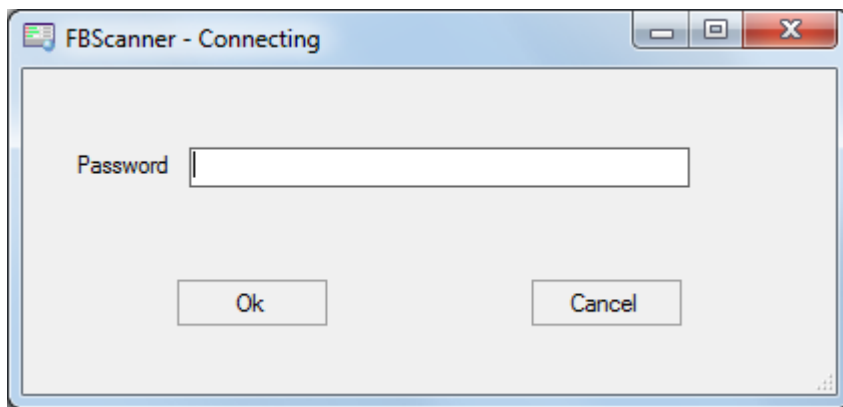
- Oldest Connection
- Process Priority...
- Ping Client
- Ping All Clients
- Extract Plans
- **Transactions**
 - OAT
- **Tools**
 - **View Style**
 - Database Administrator (connections only)
 - Database Developer (without transactions)
 - Database Developer (with transactions)
 - Language — English, Italian, Russian, Portuguese
 - Plugins
 - Options
- **Columns** — list of columns
- **Help**

Server

To connect to the FBScanner Service select Service\Connect To. The following dialog will appear:



After selecting the server FBScanner Viewer will ask for password. There are 2 passwords — for read-only access and for administrator (full) access. By default the password for read-only access is blank.



To setup passwords for FBScanner Viewer access you need to go to “FBScanner Configuration” — “Advanced Settings”.

Server\Disconnect disconnects FBScanner Viewer from FBScanner Service.

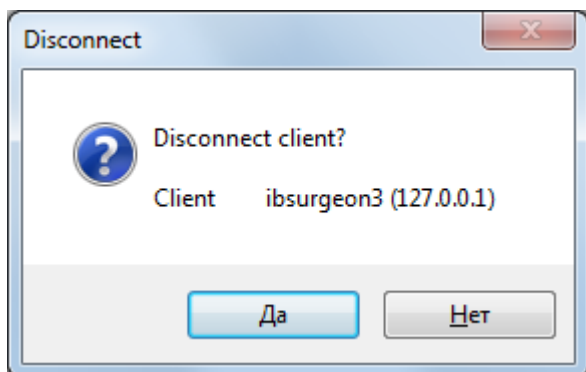
Server\Recent Servers shows list of most recent FBScanner Services where FBScanner Viewer connected to.

Exit closes FBScanner Viewer.

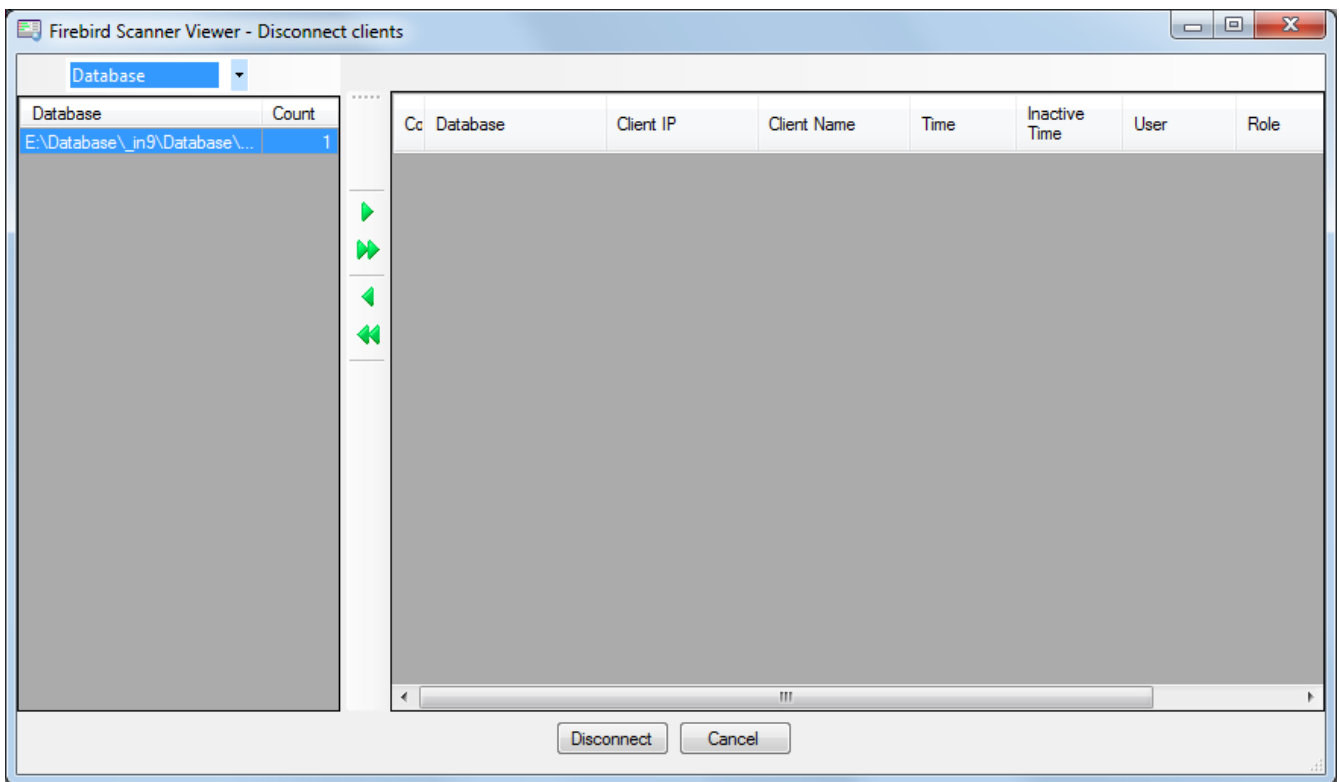
Connections

“Disconnect”, “Disconnect clients” and “Kill Process” menu options are available only when connected to FBScanner Service with administrative rights.

Disconnect will ask to close the current connection (highlighted in the main FBScanner Viewer grid):



“**Disconnect clients**” runs the following dialog:



In the right side there is a list of connections, represented by databases names, or clients, or user, according the filter above.

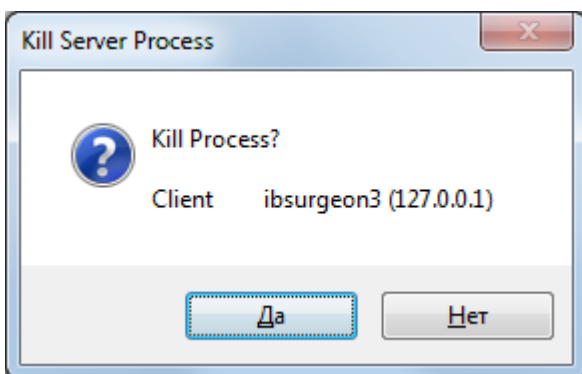
Using > and < buttons, administrator can select connections to be disconnected and then click “Disconnect” button.

Disconnect will be done by emulation of 10054 error — there will be appropriate record(s) in the *firebird.log* (*interbase.log*) and in *FBScanner.log*.

Kill

There are few cases when you need to kill Firebird process, and we do not recommend it.

“Kill process...” asks for explicit killing of Firebird process, and it works only at local FBScanner and Classic Architecture:



It will not work with SuperServer or SuperClassic architectures.

“**Latest Queries**” shows list of 20 most recent queries in the selected connection:

Number	Query Start	Processor Time	Prepare Time	Execute Time	Fetch Count	SQLCODE	SQL
1	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	7	0	select cast(1 as integer), cast(d.rdb\$character_set_name as
2	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	3	0	select RDB\$FIELD_NAME, RDB\$SYSTEM_FLAG
3	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	417	0	select RDB\$RELATION_NAME, RDB\$SYSTEM_FLAG,
4	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	5	0	select RDB\$RELATION_NAME, RDB\$OWNER_NAME
5	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	1	0	select RDB\$PROCEDURE_NAME, RDB\$OWNER_NAME
6	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	0	0	select T.RDB\$TRIGGER_NAME, T.RDB\$TRIGGER_INACTIVE,
7	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	2	0	select RDB\$GENERATOR_NAME
8	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	0	0	select RDB\$EXCEPTION_NAME, RDB\$MESSAGE,
9	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	9	0	select RDB\$FUNCTION_NAME
10	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	1	0	select RDB\$RELATION_NAME from RDB\$RELATIONS
11	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	0	0	select RDB\$ROLE_NAME, RDB\$OWNER_NAME FROM RDB\$ROLES
12	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	1780	0	select RDB\$INDEX_NAME, RDB\$RELATION_NAME,
13	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	51	0	select RDB\$CHARACTER_SET_ID, RDB\$CHARACTER_SET_NAME,
14	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	145	0	select RDB\$COLLATION_ID, RDB\$CHARACTER_SET_ID,
15	3/17/2011 1:36:14	00:00:00.0000000	00:00:00.000	00:00:00.000	1	0	select RDB\$RELATION_NAME from RDB\$RELATIONS
16	3/17/2011 1:36:15	00:00:00.0000000	00:00:00.000	00:00:00.000	1	0	select RDB\$EXTERNAL_FILE from RDB\$RELATIONS
17	3/17/2011 1:36:15	00:00:00.0000000	00:00:00.000	00:00:00.000	1	0	select RDB\$RELATION_TYPE from RDB\$RELATIONS
18	3/17/2011 1:36:15	00:00:00.0000000	00:00:00.000	00:00:00.000	1	0	SELECT DISTINCT RDB\$FIELD_NAME FROM

It's useful for ad-hoc debugging, it works like “Rewind” button.



For full-fledged logging of SQL traffic enable SQL logging feature in FBScanner Service, and use FBScanner LogAnalyzer to look through the log.

“**Oldest Connection**” shows the oldest connection in the grid.

“**Process Priority**” is applicable only for local FBScanner installation with Classic architecture. It enables to set process priority for Classic instances.

“**Ping Client**” allows to check – is selected connection still alive?

“**Ping All Clients**” checks all connections in the same way.

“**Extract plans**” starts plan extracting for selected connect. Extracted plans are shown in the grid, and also stored in the SQL (or text) log. If logging is not enabled, nothing happens. To enable plan extraction for all connects, use appropriate setting in “FBScanner Configuration”.

Transactions

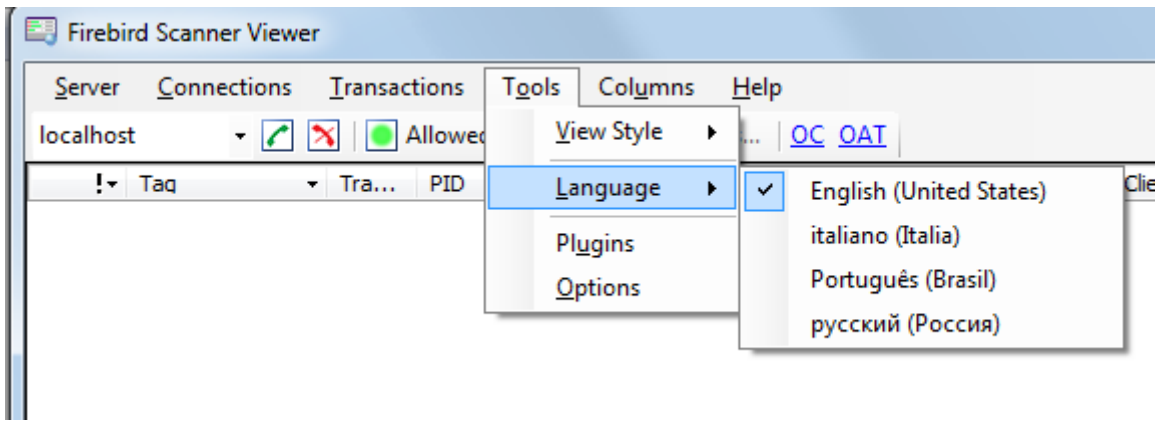
The single option **Transactions\OAT** will put selection in the grid to the oldest active transaction.

Tools

In menu “Tools” we can see several options. With “**View Style**” user can select the most suitable representation of grid data:

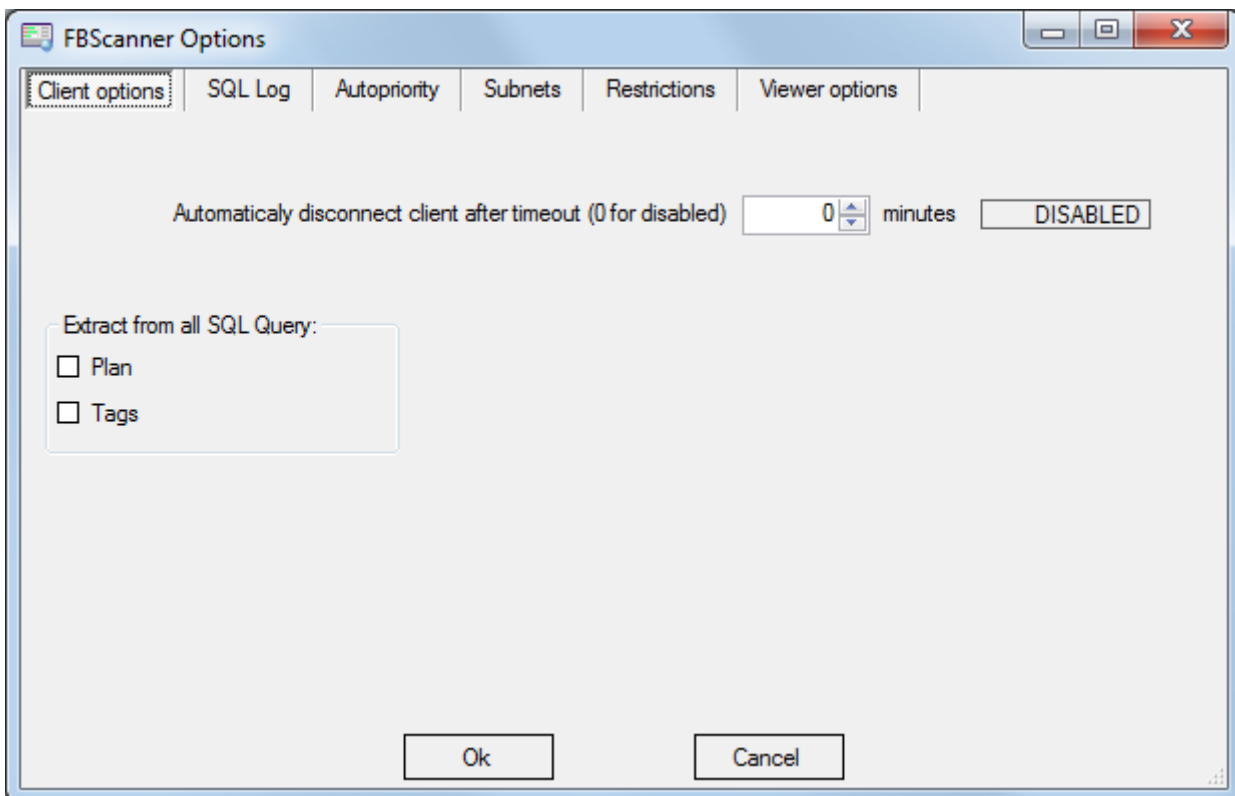
- Database Administrator (connections only)
- Database Developer (without transactions)
- Database Developer (with transactions)

FBScanner Viewer is localized in 4 languages. Use **Tools\Language** to switch between languages:



“**Plugins**” option enables plugins. For more information please contact support@ib-aid.com

“**Options**” is another way to change some of FBScanner Service parameters.



Please consider appropriate session of this guide for details of FBScanner Service Configuration.

SQL log structure

FBScanner stores SQL traffic in the following table:

```

CREATE TABLE FBSCANNER$LOG
(
  ID                BIGINT NOT NULL,
  IDATTACHMENT      BIGINT,
  IDTRANSACTION     BIGINT,
  PID               INTEGER,
  ROW_TYPE          INTEGER NOT NULL,
  CLIENT_IP         VARCHAR(24),
  CLIENT_NAME       VARCHAR(256),
  CUSTOM_NAME       VARCHAR(256),
  SUBNET_NAME       VARCHAR(256),
  DB_FILENAME       VARCHAR(512),
  DB_USER           VARCHAR(512),
  DB_ROLE           VARCHAR(512),
  START_TIME        TIMESTAMP DEFAULT 'NOW' NOT NULL,
  END_TIME          TIMESTAMP,
  LAST_ACTIVITY     TIMESTAMP DEFAULT 'NOW' NOT NULL,
  LAST_RETAINING    TIMESTAMP,
  WORK_TIME         INTEGER DEFAULT 0 NOT NULL,
  CPU_TIME_USER     INTEGER DEFAULT 0 NOT NULL,
  CPU_TIME_PRIVILEGED INTEGER DEFAULT 0 NOT NULL,
  FETCH_COUNT       INTEGER DEFAULT 0 NOT NULL,
  RESULT            INTEGER,
  SQL_TEXT          BLOB SUB_TYPE 1 SEGMENT SIZE 80,
  SQL_TEXT2         BLOB SUB_TYPE 1 SEGMENT SIZE 80,
  SQL_PLAN          BLOB SUB_TYPE 1 SEGMENT SIZE 80,
  PREPARE_TIME      INTEGER DEFAULT 0 NOT NULL,
  EXECUTE_TIME      INTEGER DEFAULT 0 NOT NULL
);

```

Logical structure

There are 3 levels of hierarchy in this table:

- ID — primary key
- IDATTACHMENT and IDTRANSACTION — foreign keys referenced to FBSCANNER\$LOG.ID
- ROW_TYPE — hierarchy level (0, 1, 2)

Table 1. Level 1. Connection.ROW_TYPE = 0

PID	Process ID (only for local FBScanner)
ROW_TYPE	0
CLIENT_IP	IP address of client
CLIENT_NAME	DNS name
CUSTOM_NAME	Connection tag (if assigned in query text)
SUBNET_NAME	Logical name of subnet. See file FBScanner.subnets

DB_FILENAME	Database alias or full database path
DB_USER	User name
DB_ROLE	User role
START_TIME	Start of connection
END_TIME	End of connection

Table 2. Level 2. Connection.ROW_TYPE = 1

IDATTACHMENT	Connection ID
ROW_TYPE	1
CUSTOM_NAME	Transaction tag (if assigned)
START_TIME	Transaction start time
END_TIME	Transaction end time
LAST_RETAINING	Time of most recent commit retaining or rollback retaining
RESULT	0 – transaction is active 1 – Commit 2 – Rollback
SQL_TEXT	Transaction flags

Table 3. Level 2. Connection.ROW_TYPE = 2

IDATTACHMENT	Connection ID
IDTRANSACTION	Transaction ID
ROW_TYPE	2
CUSTOM_NAME	Query tag (if assigned)
START_TIME	Query start time
WORK_TIME	Time till the answer from server
CPU_TIME_USER	CPU Time (local only)
CPU_TIME_PRIVILEGED	CPU Kernel Time (local only)
FETCH_COUNT	Number of records, returned by query
RESULT	0 – query executed successfully, otherwise this field contains SQLCODE of error
SQL_TEXT	Query text (with parameters)
SQL_TEXT2	Original query text(NULL if equal to SQL_TEXT)
SQL_PLAN	Query execution plan (if “Extract plans” setting is enabled)

PREPARE_TIME	Prepare time
EXECUTE_TIME	Query execution time

Indices in the log

Initially log database contains only primary key index. FBScanner Log Analyzer creates necessary indices at the first connect.

4.4.11. FBScanner Feature Matrix

#	Feature	FBScanner mode	
		Agent	Remote
	Operation Systems Support		
	Windows	X	X
	Linux, Mac OS X, Free BSD		X
	Firebird and InterBase versions supported		
	Firebird 1.0, Yaffil 1.0 (including logging)	X	X
	Firebird 1.5 (including logging)	X	X
	Firebird 2.0 (including logging)	X	X
	Firebird 2.1 (including logging)	X	X
	Firebird 2.5 (including logging + SuperClassic support)	X	X
	InterBase 6.0-2009/XE (including logging)	X	X
1	Connections		
1.1	<i>Information about established connections in the FBScanner Viewer:</i>		
	Firebird/InterBase user login	X	X
	IP-address or computer name	X	X
	Connection time and time of the latest activity	X	X
	Priority of processes (only for Classic architecture)	X	
1.2	<i>Connection management (requires logging to FBScanner Viewer with Admin rights)</i>		
	Safe disconnect of one or several connections using TCP/IP connection interruption (imitation of 10054 error)	X	X
	Changing of processes priority in Classic architecture (for example, to adjust priority of long running report or something like this. Using tags administrator can recognize connection where report is working — see below in “Tags”).	X	

	Automatic priority settings for Firebird with Classic architecture. In FBScanner configuration administrator can set up automatic correspondence: Specified IP or subnet of IPs – set priority X Specified hostname – set priority X Specified database name – set priority X Specified user login name – set priority X	X	
	Killing of Classic processes, not recommended to use, but sometimes it is helpful	X	
	Ability to restrict all connections (to perform some operations which require exclusive access)	X	X
	Filtering connections viewing using all connections parameters (except time information)	X	X
	White and black list of databases to connect	X	X
	White and black list of IP addresses (clients)	X	X
	Restriction of connections # — administrator can limit the number of connections	X	X
	Emulation of “Wrong login/password” error for denied connections	X	
	Detection of old/incorrect versions of fbclient.dll/gds32.dll	X	X
1.3	<i>Logging events related with connections</i>	X	X
	FBScanner logs unsuccessful login attempts in the <i>FBScanner.log</i> . For each unsuccessful login attempt FBScanner writes the following information: IP-address, login name, database and time of login attempt.	X	X

	<p>If connection was broken (10054 error), FBScanner determines and logs one of the 5 type of disconnects:</p> <p>Client application was closed improperly (for instance, application was closed by Task Manager)</p> <p>Connection was closed by time-out (it's possible to set forced disconnect in FBScanner to close connect by time-out too)</p> <p>Server crashed (fbserver or fb_inet_server crashed)</p> <p>Server process (fbserver or fb_inet_server) was killed from the FBScanner</p> <p>Disconnect of connections from FBScanner Viewer</p> <p>For all cases above FBScanner writes the IP-address of disconnected client(s) and the reason of disconnect. This is very useful feature to find and eliminate 10054 errors.</p>	X	X
2.	Transactions		
2.1.	<i>Transactions are shown inside appropriate connections</i>		
	Transactions' flags	X	X
	Lifetime of transactions	X	X
	Using OAT button you can find the oldest active transaction in real-time and review related connection/queries.	X	X
3.	Queries (statements)		
3.1	<i>Information about queries(statements)</i>		
	Start time	X	X
	Query text	X	X
	Transaction of the query	X	X
	Status (prepare/execute/...)	X	X
	Filtering by statement status (by default Closed statements are hidden)	X	X
	Instant CPU load indicator	X	X
	If query PREPARE or execution caused error, FBScanner writes SQLCODE to the log (for example, primary key violation)		

3.2	<i>Additional operations with queries</i>		
	<p>Ad-hoc plan extraction for queries</p> <p>Can be performed for all connections (should be set ON in FBScanner configuration utility)</p> <p>Can be turned ON/OFF for selected connection only in the FBScanner Viewer</p> <p>In both cases plans will be logged to the overall log if logging is ON.</p>	X	X
4.	Tags		
	<p>Tags allow assigning readable identifiers (names) to Connections, Queries and Transactions. You just need to add these commentaries: [source] ----</p> <pre>SELECT COUNT() FROM RDB\$DATABASE / FBSCANNER\$CON_NAME=My_application; FBSCANNER\$TR_NAME=Read_only_transaction_N1; FBSCANNER\$ST_NAME=Customers_list_query; */ ----</pre>	X	X
	<p>FBSCANNER\$CON_NAME= sets the name of connection. After the first assignment this name will be kept during the whole connection life.</p>	X	X
	<p>FBSCANNER\$TR_NAME= sets the name of transaction. After the first assignment this name will be used during the whole life of transaction</p>	X	X
	<p>FBSCANNER\$ST_NAME= sets the name of query.</p>		
	<p>Tags are showed in special column in FBScanner Viewer</p>	X	X
	<p>It's possible to filter tags by their names</p>	X	X
	<p>Tags are useful to quickly answer the following frequent questions:</p> <p>What program has launched this query? (developers need to mark with FBSCANNER\$CON_NAME tag each database connection)</p> <p>What is the transaction for this query? (developers need to use FBSCANNER\$TR_NAME tag to mark transactions)</p> <p>What is this very long query? (developer can mark long queries with readable names like "Annual report")</p>	X	X

5.	Logging		
	Logging allows intercepting all queries and writing them to the external Firebird database. FYI, logging cannot be replaced with Firebird 2.1 or InterBase system tables, because they provide only snapshots of programs.	X	X
	Connections, queries and transactions are logged	X	X
	All executed queries are logged (only prepared queries skipped)	X	X
	Queries are stored with information about their connection and transaction	X	X
	All transactions are logged, even rolled back. Transaction log record has column RESULT which shows was transaction committed or rolled back.	X	X
	If plan extraction is on, queries plans are logged too	X	X
	Automatic creation of database for logging	X	X
	Automatic creation of tables to logging in any Firebird database	X	X

Chapter 5. Database structure analysis

5.1. Overview of Firebird database structure

The first thing we have to say about the structure of Firebird database is that it represents a set of pages of strictly defined size: 4096, 8192, 16384 (previous versions of Firebird supported page sizes 1024 and 2048).

Pages can be of different types, each of which serves its certain purpose.

Pages of the same type don't go strictly one by one — they can be easily mixed, allocated in file in the order they were created by server when extending or creating databases.

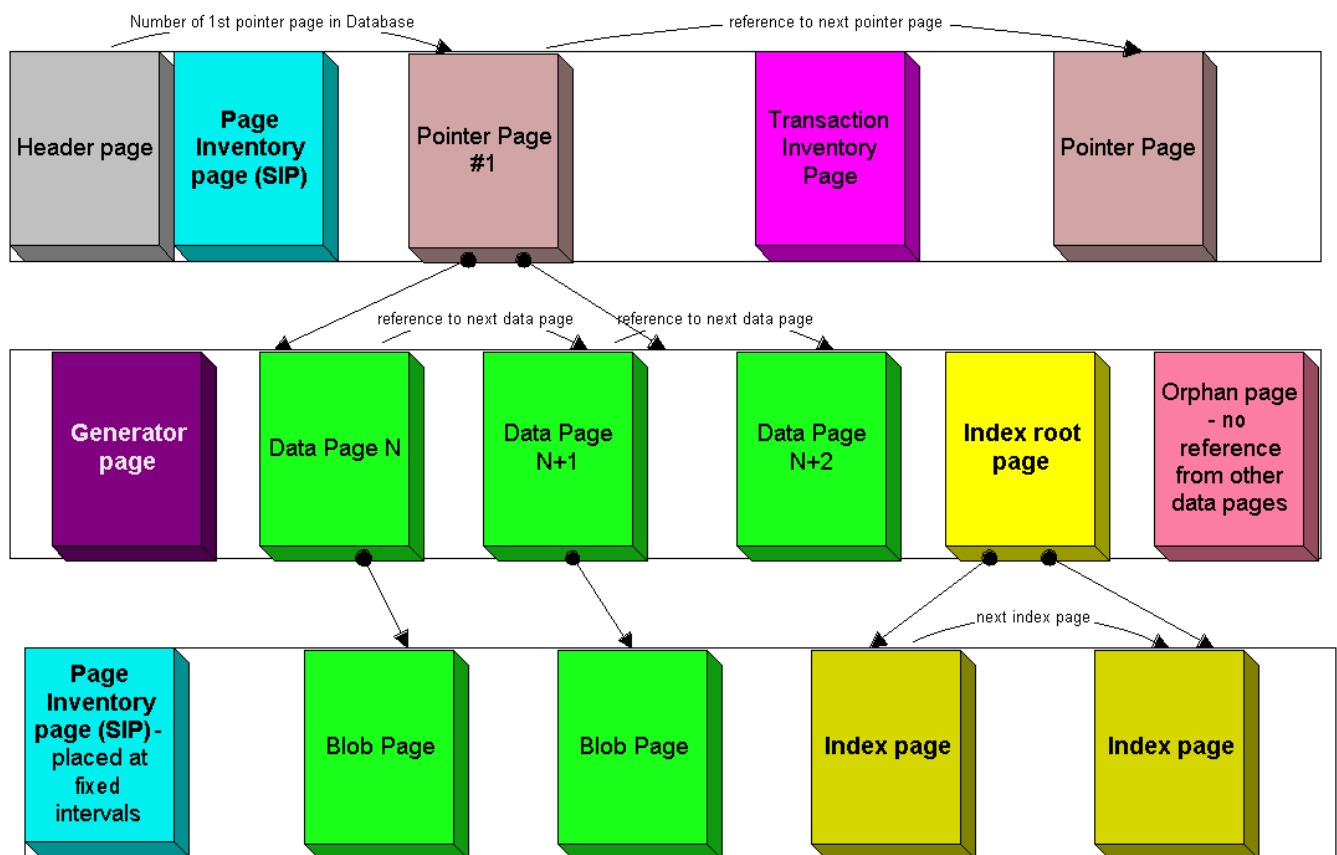


Table 4. Page types

Page Type	ID	Description
<code>pag_undefined</code>	0	Undefined – If a page has this page type, it is most likely empty
<code>pag_header</code>	1	Database header page
<code>pag_pages</code>	2	Page inventory page (or Space inventory page – SIP)
<code>pag_transactions</code>	3	Transaction inventory page (TIP)
<code>pag_pointer</code>	4	Pointer page

Page Type	ID	Description
pag_data	5	Data page
pag_root	6	Index root page
pag_index	7	Index (B-tree) page
pag_blob	8	Blob data page
pag_ids	9	Gen-ids
pag_log	10	Write ahead log information

5.2. How to analyze database structure with HQbird Database Analyst (IBAnalyst)

IBAnalyst is a tool that assists a user to analyze in detail Firebird database statistics and identify possible problems with database performance, maintenance and how an application interacts with the database. IBAnalyst graphically displays Firebird database statistics in a user-friendly way and highlights the following problems:

- tables and BLOBs fragmentation,
- record versioning,
- garbage collection,
- indices effectiveness, etc

Moreover, IBAnalyst can automatically make intelligent suggestions about improving database performance and database maintenance.

IBAnalyst can get statistics from the live production databases through Services API (recommended), or analyze text output of `gstat -a -r ...` commands. Statistics from the peak load periods can give a lot of information about actual performance problems in production databases.

Main features of IBAnalyst are listed below:

- Retrieving database statistics via Service API and from `gstat` output.
- Summary of all actual and possible problems in database
- Colored grid representation of tables, indices and table → indices, which highlights fragmented tables, poor indices and so on.
- Automatic expertise of database statistics provides recommendations and “how-to” for the following things:
 - Optimal database page size
 - Transactions state and gap between critical transactions
 - Different database flags
 - Index Depth
 - Index Key Duplicates

- Fragmented Tables
- Record Versions
- Very Big Tables
- and more...

5.2.1. How to get statistics from Firebird database in right way

Right time, right place

It sounds strange, but just taking statistics via `gstat` or Services API is not enough. Statistics must be taken at the right moment to show how applications affect data and transactions in database. Worst time to take statistics is

Right after restore

After backup (`gbak -b db.gdb`) without `-g` switch is made

After manual sweep (`gfix -sweep`)

It is also correct that during work there can be moments where database is in correct state, for example, when applications make less database load than usual (users at launch, dinner or its by specific business process times).

How to catch when there is something wrong in database?

Yes, your applications can be designed so perfect that they will always work with transactions and data correctly, not making sweep gaps, lot of active transactions, long running snapshots and so on. Usually it does not happen. At least because some developers test their applications running 2-3 simultaneous users at the same time, not more. Thus, when they set up written applications for 15 and more simultaneous users, database can behave unpredictably. Of course, multi-user mode can work Ok, because most of multi-user conflicts can be tested with 2-3 concurrently running applications. But, next, when more concurrent applications will run, garbage collection problems can come (at least). And this can be caught if you take statistics at the correct moments.

If you does not experience periodical performance problems

This can happen when your applications are designed correctly, there is low database load, or your hardware is modern and very powerful (enough to handle well current user count and data).

The most valuable information is transactions load and version accumulation. This can be seen only if you setup regular statistics saving.

The best setup is to get hourly transaction statistics. This can be done by running

```
gstat -h db.gdb>db_stat_<time>.txt
```

where

- `db.gdb` is your database name,
- `db_stat_<time>.txt` is text file where statistics will be saved,
- `<time>` — current date and time when statistics was taken.

Also you can schedule to gather database statistics with HQbird FBDataGuard, job “Database: Statistics”.

If you experience periodical performance problems

These problems usually caused by automatic sweep run. First you need to determine time period between such a performance hits. Next, divide this interval minimally to 4 (8, 16 and so on). Now information systems have lot of concurrent users, and most of performance problems with not configured server and database happens 2 or 3 timers per day.

For example, if performance problem happens each 3 hours, you need to take

```
gstat -h db.gdb
```

statistics each 30-45 minutes, and

```
gstat -a -r db.gdb -user SYSDBA -pass masterkey
```

each 1-1.5 hour. The best is when you take `gstat -a -r` statistics right before forthcoming performance hit. It will show where real garbage is and how many obsolete record versions accumulated.

What to do with this statistics

If your application explicitly uses transactions and uses them well, i.e. you know what is `read_committed` and when to use it, your snapshot transactions lasts no longer than needed, and transactions are being active minimal duration of time, you can tune sweep interval or set it off, and then only care about how many updates application(s) makes and what tables need to be less updated or cared about updates.

What does this mean, you can ask? We'll give example of some system, where performance problems happened each morning for 20-30 minutes. That was very sufficient for *morning* applications, and could not last longer.

Database admin was asked correct questions, and here is the picture:

Daily work was divided by sections — analytic works in the morning, than data is inserted and edited by usual operators, and at the end of the day special procedures started gathering data, that would be used for analytic next day (at least).

The last work on database at the end of day was lot of updates, and updates of those tables which analytic used in the morning. So, there were a lot of garbage versions, which started to be collected by application, running in the morning.

And, the answer to that problem was found simple — to run `gfix -sweep` at the end of the day.

Sweep reads all tables in database and tries to collect all garbage versions for committed and rolled back transactions. After sweeping database became clear nearly it comes after restore.

And, “morning problem” has gone.

So, you need to understand statistics with lot of other factors:

- how many concurrent users (average) work during the day
- how long is the working day (8, 12, 16, 24 hours)
- what kind of applications running at different day times, and how they affect data being used by other applications, running at the same time or next. I.e. you must understand business processes happening during the whole day and whole week.

When DBA can't do nothing

Sadly to say, these situations happen. And again, example:

Some system installed for ~15 users. Periodically performance is so bad, that DBA needs to restart server. After server restart everything works fine for some time, then performance gets bad again. Statistics showed that average daily transactions is about 75,000, and there are active transactions running from the start of day to the moment when performance getting down.

Unfortunately, applications were written with BDE and with no transactions using at all; i.e. all transaction handling was automatic and used by BDE itself. This caused some transactions to stay active for a long time, and garbage (record versions) accumulated until DBA restarted server. After restart the automatic sweep will start, and the garbage will be collected (eliminated).

All these was caused by applications, because they were tested only with 2-3 concurrent users, and when they became ~15, applications started to make very high load.

Need to say that in that configuration 70% of users were only reading data, and other 30% were inserting and updating some (!) data.

In this situation the only thing that can make performance better is to redesign transaction management in this application.

How IBAnalyst can help find problems in your Firebird database

Let's walk through the key features of IBAnalyst. When you look at your database statistics in IBAnalyst first time, things can be not clear, especially if IBAnalyst shows lot of warnings by colored red and yellow cells at Summary, Tables and Index views. Let's consider several real statistics examples.

5.2.2. Summary View

Summary contains the most important information extracted from database statistics. Usually full statistics of database contains hundreds of Kbytes and it is not easy to recognize the important information.

Below is the description of database objects and parameters that you may see in Summary. For description of visible problems (marked **red** or **yellow**) see column hints or Recommendations output.

Object or parameter	Description
Database name	Name of analyzed database.
Creation date	Database creation date. When it was created by CREATE DATABASE statement or restore (gbak -c/-r).
Statistics date	When statistics was taken — statistics file date or Services API call date (now).
Page size	Page size is the physical parameter of database. The best page size is 4096 or 8192 bytes. Other page sizes (less than 4096) marked as red. For better performance restore database from backup using 4K or 8K page size. (Note: Firebird 2.0+ can use 16K page size).
Forced Write	It shows the mode of changed pages writing: synchronized or asynchronized — appropriate setting is ON or OFF. OFF is not recommended, because server crash, power failure or other problems can cause database corruption.
Dialect	Current database dialect.
Sweep interval	Current sweep interval value. Marked yellow if it is not 0, and marked red if Sweep Gap greater than Sweep interval.
On Disk Structure	ODS. It is a database physical format. See hint to know ODS number for particular IB/FB versions
Transaction block	
Oldest transaction	Oldest interesting transaction. The oldest transaction id that was rolled back, or in limbo.
Oldest snapshot	Oldest snapshot transaction Id of transaction that was oldest active when currently oldest snapshot started.
Oldest active	Oldest active transaction Id of oldest still active transaction.
Next transaction	Newest available transaction id

Object or parameter	Description
Sweep gap (snapshot – oldest)	For ODS 10.x databases. Difference between Oldest Snapshot and Oldest Interesting transaction. If it is greater than sweep interval, and sweep interval is > 0, Firebird tries to run sweep, and it can slowdown performance.
Snapshot gap (active – oldest)	Difference between Oldest Active and Oldest transaction. Same as previous sweep gap.
TIP size	Transaction Inventory Page size, in pages and kilobytes. TIP holds transaction state for every transaction was started from database creation (or restore). It is computed as Next transaction div 4 (bytes).
Snapshot TIP Size	Size of Transaction Inventory Pages that needed for snapshot transactions. Indicates how much memory will take each snapshot transaction to check concurrent transactions state.
Active transactions	Currently active (on the moment when statistics was taken from database) transaction count (Next – Oldest Active). Maybe incorrect, because it can be one active transaction and lot of ahead transactions committed. Anyway, active transactions prevent garbage collection.
Transactions per day	Simply divides Next transaction by days' count between database creation date and date statistics taken. Shows average transaction per day, and useless if it is not production database. Transaction warnings mostly based on average transactions per day count.
Data versions percent	Percent of record versions in database. Also total records size and versions size for all tables is shown, and total index size. Row is not shown when statistics does not contain record count information (gstat -a without -r option). Note that there can be lot of other data (transaction inventory pages, empty pages and so on) in your database.
Table/Index lists (also reported in recommendations)	
Fragmented Tables	Here you can view tables (with data > 200 kilobytes) that have average fill less than 60% (File/Options/Table average fill).

Object or parameter	Description
Versioned Tables	List of tables that have Versions greater than Records, set in Options/Tables.
Tables fragmented with blobs	List of tables that have blob fields with data size less than database page size.
Massive deletes/updates	List of tables that had lot of data deleted/updated by one delete/update statement.
Very big tables	Tables that are close to technical InterBase limit (36 gigabytes per table). You will see warning beforehand problem can occur.
Deep Indices	Indices with depth more than 3 (Options/Index)
Bad Indices	Indices with big MaxDup and TotalDup values
Broken or incomplete indices	Indices with key count less than record count. This can happen when index is broken or when statistics is taken during index creation or re-activation.
Useless Indices	Indices with Unique column = 1. May be deleted or deactivated, because they are useless for index search or sort operations.
Tables with no records	List of tables with Records = 0. This can be by design (temporary tables), or they can be just forgotten by database developer.

Parameter	Value
Database info	
Database name	D:\FbData\db.fdb
Creation date	07.09.2017 17:34:04
Statistics date	11.11.2019 20:05:36
Page size	16384
Forced Write	ON
Dialect	1
OnDiskStructure	12.0
Implementation	HW=AMD/Intel/x64 little-endian OS=Windows CC=MSVC
Attributes	force write
Sweep interval	20000
Oldest transaction	69638805
Oldest snapshot	69638806
Oldest active	69638806
Next transaction	69668480
Sweep gap (active - oldest)	
TIP size	1064 pages, 17025 kilobytes
Snapshot TIP size	29675 transactions, 23 kilobytes
Active transactions	29674, 34% of daily average
Transactions per day	87523, for 796 days
Data versions percent	600% - records: 978 mb, versions: 0 mb, pages 1370 mb, indices 486 mb
Blob size	45.66 megabytes
Fragmented tables	
ATTACHMENTS	Average Fill: 37%, Records 1430, Pages 16
DNA_FREQ_PER_BREED	Average Fill: 52%, Records 2125, Pages 16
EXTERIOR	Average Fill: 54%, Records 7842, Pages 40
PROTECTED_NAMES	Average Fill: 59%, Records 4315, Pages 24
Versioned tables	
Tables fragmented with blobs	
Massive deletes/updates	
Very big tables	

Summary page shows a lot of information, but the most valuable is transactions state (*please read description of possible transactions states in IBAnalyst help, it is available by clicking F1 or in menu Help*).

At this screenshot you can see that some transaction is active for a long time, “60% of daily average”. IBAnalyst marks such transaction’s state by red, because this transaction may prevent accumulated versions to be considered as garbage by server, and so, to be garbage collected. This is a possible reason of slowness: the more versions exist for some record, the more time it will take to read it.

In order to find this long-running transaction you can use MON\$Logger module of FBScanner, or perform direct query of MON\$ tables. Then, to find out which tables were affected by long running transactions (tables with a lot of record versions) you need to go to “Tables” view of IBAnalyst.

5.2.3. Tables view

View **Tables** contains the information about all database tables. It represents important statistical information about each table. All table warnings are marked (see details below).

You can see the following columns (Columns **Records**, **RecLength**, **VerLen**, **Versions**, **Max Vers** are visible only if statistics was generated with `gstat -r` or with “Include record/rec versions” checkbox enabled):

Column	Description
Records	Record count. Marked pink if table fragmented by blob fields which data is less than database page size. Hint shows real table fragmentation and average records if there were no blob fields. Such fragmentation can cause bad performance for big table joins or natural scans.
RecLength	Average record length. Depends on record data, especially on char/varchar columns data. Min physical record length is 17 bytes (record header + all fields are null), max – as declared in table. Statistics show this data without record header count, in this case RecLength can be 0 (if nearly all records are deleted)
VerLen	Average record version length. If it is close to RecLength, almost all record is being updated. If VerLen is 40-80% and not greater of RecLength, then Versions are mostly updates. If VerLen greater than 80-90% of RecLength, than maybe Versions are mostly deletes, or update is made by char/varchar columns with new, greater data. Marked yellow if it's size is greater than specified % (Options/Record/Version size) of average record size.
Versions	Current record version count. More versions slowdown table reads. Also lot of versions means that there is no garbage collection performed or records are not read by anyone. Marked red if version count is greater than Records. (Options/Record Versions).
Max Vers	Max record versions for one particular record. Marked blue when it is equal to 1 and Versions is non-zero. It means that there were massive update/delete operation. See Options, Table, Massive deletes updates option.
Data Pages	Allocated data pages
Size, Mb	DataPages * Page Size, in megabytes. I.e. this is total table size, records + versions. Graph shows percentage of that table from the whole data size.
Idx Size, Mb	Sum of all indices size for that table. Graph shows percentage of that value to total size of all indices.

Column	Description
Slots	Count of links to data pages. Empty links are Slots-Data Pages. Doesn't affect disk space or performance.
Average Fill	Average data page fill %. Can be computed as (DataPages * Page_Size)/ Records * RecLength. Low page fill means that table is "fragmented". Frequent updates/deletes can fragment data pages. Marked red if average fill rate is less than 60% (go to Options/Average Fill to adjust it). Marked yellow if it is an artifact of high table fragmentation when it's record is too small (11-13 bytes).
Real Fill	Because we found that Average Fill, calculated by gstat, sometimes gives wrong results (at least for tables with small blobs), we placed here calculated column, that counts average fill not by data pages, but by records+versions, including record header.
20%, 40%, 60% and 100% fill	Shows page count having corresponding fill rate. Can be turned on/off in Options dialog
Total %	How big is that table plus it's indices in %, related to other tables.

Table	Records	RecLength	VerLen	Versions	Max Vers	Data Pages	Size, mb	IdxSiz...	Blobs...	Slots	Avg fill%	RealFill
SH_TYPTASK	68	80.81	0.00	0	0	4	0.03	0.00		4	73	20
SH_UNIT_TARA	61	42.00	0.00	0	0	1	0.01	0.00		1	44	44
SHLOP_TABLE	16386	18.00	0.00	0	0	117	0.91	0.00		117	60	60
SITE	40611	80.69	91.21	76142	501	2408	18.81	0.00		2554	95	62
SITEEMPL	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
SITETYPE	22	78.32	0.00	0	0	1	0.01	0.00		1	26	26
SITETYPETYPE	10	47.20	0.00	0	0	1	0.01	0.00		1	8	8
SLOT	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
SLOT_DATEEXPIRE	828	43.61	0.00	0	0	10	0.08	0.00		10	63	61
STOREDFILTER	23	62.17	0.00	0	0	1	0.01	0.00		1	75	22
TAX	2	52.00	0.00	0	0	1	0.01	0.00		1	2	2
TBINF_BUFFER2	36293	229.65	0.00	0	0	1221	9.54	0.00		1221	90	90
TBINF_TBTYPE	9	62.11	0.00	0	0	1	0.01	0.00		1	9	9
TD_COMMLOG_EXPORT	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
TD_TEMP_ORDERIMORTLOAD_R	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
TEMP_EXP_OP	2027	0.00	26.00	2027	1	16	0.13	0.00		28	93	93
TEMP_HOURCOEF	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
TEMP_MD_ORDER	12079	40.37	0.00	0	0	123	0.96	0.00		132	69	69
TEMP_OP_ART	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
TEMP_OP_FOR_EXPORT	5942	0.03	19.00	5942	1	41	0.32	0.00		41	94	95
TEMP_PT_ART	48	77.65	0.00	0	0	1	0.01	0.00		1	56	56
TEST_MDCALCORDER	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
TIMEBOARD	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
TIMEBOARD_DETAIL	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0
TMP_CHECKDOCTBL	5	82.20	0.00	0	0	1	0.01	0.00		1	6	6
TMP_PRICEON	0	0.00	0.00	0	0	0	0.00	0.00		0	0	0

At “Tables” view you can see tables and their important parameters: number of records, number of record versions, record length, maximum number of versions, etc.

You can sort this view to find the largest tables. Especially we are interested tables with many record versions – many record versions will make garbage collection for affected tables longer. Usually it is necessary to change update and delete algorithms to get rid of many record versions.

Row Versions show total versions count for particular table, and row Max Vers shows maximum versions reached by some record. For example, if you look at table SITE, there are 40611 records, total versions are 76142, but one record has 501 versions. Reading and parsing such packet from disk takes more time, so, reading this record is slower than reading others.

This picture also shows a lot of tables where data was deleted. But, because of long running transaction, server can’t delete these versions, and they still on disk, still indexed, and still being read by server when reading data.

5.2.4. Index view

View **Indices** represents all indices in your database. You can estimate the effectiveness of indices with the following parameters (problem indices are marked **red** — see smart hints for details)

Column	Description
Depth	Index depth is the page count that engine reads from disk to walk from index root to record pointer. Optimal index depth is 3 or less. When Depth is 4 and higher, it is recommended to increase database page size (backup, then restore with -page_size option). This column will be marked red if index depth is greater than 3 (Options/Index/Index Depth). More chances to exceed optimal depth have indices built on long char/varchar columns.
Keys	Index key count. Usually equals to Records. If Keys is bigger than Records and Versions count is greater than 0 it means that concrete field value was changed in those record versions. If Table.RecVersions is bigger than Keys, than this index field(s) are not changed during updates.
KeyLen	Average index key length. The less KeyLen, the more equal or similar (postfix) values (keys) stored in index.
Max Dup	Maximum duplicates count for particular key value. Some old gstat versions show no more than 32767 or 65535 — this bug is fixed in latest Firebird versions. Marked red if duplicates count is 30% of all keys. (Options/Index/Lot of key duplicates).

Column	Description
Total Dup	<p>The overall count of keys with the same values. Some old gstat versions show no more than 32767 or 65535 — this bug is fixed in latest Firebird versions.</p> <p>The closer this value to Keys count, the less effective will be searching using this index, especially when search is made using more than one index. Total Dup value can be counted as Keys minus unique keys count (index statistics is nonlinear).</p> <p>Marked yellow if $1/(\text{Keys} - \text{TotalDup})$ greater than 0.01, and red if in addition MaxDup is marked red too. This constant (0.01) is used by optimizer (see sources in <i>opt.c/opt.cpp</i>) as usable index selectivity border. Optimizer will still use that index if none other index with better selectivity exists for some condition.</p>
Uniques	<p>Count of different key values. Primary and unique key indices will show same value as in Keys column. Useful to understand how many different values stored in index — is it useful or not. Index is useless if Unique column shows 1 (marked yellow).</p>
Selectivity	<p>Information from <code>rdb\$indices.rdb\$statistics</code>, only visible if “load table/index metadata” was On. If selectivity stored in database differs from computed selectivity, yellow warning shown (less than 20% difference) or red (higher than 20% difference). Blue warning is shown when index is empty but it’s selectivity is not 0. Selectivity of inactive indices are ignored.</p>
Size, Mb	<p>Index size in megabytes. Gap show percentage of that index size related to total size of all indices.</p>
Average Fill	<p>Average index pages fill rate, in %. Marked red if average fill rate is less than 50% (go to Options/Average Index Fill to adjust it). Fragmented index results more page reads as usual, and it’s Depth can be higher. Can be fixed by alter index inactive/active, if it is not index created by primary, unique or foreign key constraints.</p>

Column	Description
Leafs	Leaf page count (pages with keys and record pointers).
20%, 40%, 60% and 100% fill	Shows page count having corresponding fill rate. Can be turned on/off in Options dialog

Database Analyst (IBAnalyst 3.0). Loaded from C:\Users\Denis\Downloads\10.11.120.10_wssterlitamak_18_20131119_15-37.iba

Statistics Reports View Options Help

Databases Summary Tables Indices Tables + Indices

Useless indices

Index	Table	Depth	Keys	Key Len	Max Dup	Total Dup	Uni...	Selectivity	Size, ...
F_UP_CORREMPLOYEEEDRIVE	OP_CORR	2	9720	0,00	9719	9719	1	1,0000000	0,06
F_OP_CORREMPLOYEEESTOSK	OP_CORR	2	9720	0,00	9719	9719	1	1,0000000	0,06
F_OP_CORRROP_ART	OP_CORR	2	9720	0,00	9719	9719	1	1,0000000	0,06
I_OP_CORR_OPARTCHILD	OP_CORR	2	9720	0,00	9719	9719	1	1,0000000	0,07
F_PLTYPE	PL_TYPE	1	17	0,00	16	16	1	1,0000000	0,01
F_PRICEWS	PRICE	1	712	0,00	711	711	1	1,0000000	0,01
F_QUESTIONEXAMINATION	QUESTION	1	22	0,00	21	21	1	1,0000000	0,01
F_REMLICAREMIND	REPLICA	1	2	4,00	1	1	1	1,0000000	0,01
F_REMLICAREMINDD	REPLICA	1	2	4,00	1	1	1	1,0000000	0,01
F_SERVERREPL	REPLICA	1	2	1,00	1	1	1	1,0000000	0,01
F_RESTCONTRCUR	RESTCONTR	2	58475	0,00	58474	58474	1	1,0000000	0,35
F_PARTY_RFS_ART	RFS_PARTY	2	12274	0,00	12273	12273	1	1,0000000	0,08
I_NUMBER	RFS_PARTY	2	12274	0,00	12273	12273	1	1,0000000	0,08
RDB\$PRIMARY169	R_DOMAIN	1	1	2,00	0	0	1	1,0000000	0,01
F_R_FIELDDOMAIN	R_FIELD	2	1836	0,00	1835	1835	1	1,0000000	0,02
F_R_ORDERSCHEDULER_ORDALG	R_ORDERSCHEDULER	2	1495	0,00	1494	1494	1	1,0000000	0,02
F_R_ORDERSCHEDULER_THROW	R_ORDERSCHEDULER	2	1495	0,00	1494	1494	1	1,0000000	0,02
F_R_ORDERSCHEDULER_WS	R_ORDERSCHEDULER	2	1495	0,00	1494	1494	1	1,0000000	0,02
F_R_QUEUEUSORT	R_QUEUE	2	1010554	0,00	1010553	1010553	1	1,0000000	6,66
F_R_QUESORTSERVER	R_QUEUEUSORT	1	9	0,00	8	8	1	1,0000000	0,01
F_R_REQUESTTYPE_RECEIPT	R_REQUESTTYPE	1	9	0,00	8	8	1	1,0000000	0,01
F_STATUSTYPE	R_STATUS	1	4	0,00	3	3	1	1,0000000	0,01
RDB\$PRIMARY269	R_STATUSTYPE	1	1	1,00	0	0	1	1,0000000	0,01
F_USERREQUESTREQUESTTYPE	R_USER_REQUEST	1	1	1,00	0	0	1	1,0000000	0,01
F_USERREQUESTUSERS	R_USER_REQUEST	1	1	3,00	0	0	1	1,0000000	0,01
RDB\$PRIMARY225	R_USER_REQUEST	1	1	7,00	0	0	1	1,0000000	0,01

Some production databases can have indices with the only key value indexed. This can happen because database was developed “to be extended in the future”, or, someone just experimented with the indices during development or tests. You can see these indices as “Useless” in IBAnalyst: I_NUMBER, etc, built on the column that has only one value for all rows. These indices are really useless, because

- Optimizer may use this index if you specify “where field = ...”. Since field contains only one value, using index will cause useless reading of index pages from disk to memory, and consume memory (and time) when server will prepare which rows to show for that query.
- Creating indices is the part of restore process. Extra indices adds extra time.

Of course, that is not all that you can find about your database in IBAnalyst. You can also find

- average number of transactions per day
- was there rollbacks or lost connections, and when
- how big (in megabytes) each table and index
- tables that have records interchanged by blobs, and thus reading only records is slower
- empty tables — just forgotten, or empty at the time when statistics was taken

- indices with lot of duplicate keys (you can consider about column value distribution)
- indices with depth 4 and greater — maybe you need to increase page size to speed up

Chapter 6. HQbird Enterprise configuration: Native Firebird replication and PerformanceEnhancements

6.1. What is HQbird Enterprise?

HQbird Enterprise is the advanced distribution of Firebird for big databases with monitoring, optimization and administration tools, it also includes the plugin for native master-slave replication and various performance improvements.

6.1.1. Compatibility

HQbird Enterprise is 100% compatible with Firebird 2.5.5+, Firebird 3.0 and Firebird 4.0 – no changes in ODS are needed. To switch to HQbird and back no backup/restore is required, just stop/start Firebird and replace binaries. The replication is possible between nodes with the same version, i.e., not possible between 3.0 and 2.5.

6.1.2. How the replication works

HQbird replication works on the logical level: it replicates DML statements (INSERT/UPDATE/DELETE, stored procedures, etc) and DDL (CREATE/ALTER TABLE, etc) changes; **no additional triggers needed**. The only requirement for the current version of replication is to have unique or primary keys for all tables that need to be replicated.

In order to use the replication, you need to install HQbird instead of Firebird, register it (with trial or with the full license) and configure replication. HQbird should be running on the master server and on all replica servers.

Below we will consider how to setup Firebird replication with HQbird Enterprise.

You can use HQbird on Windows and Linux, with Firebird 2.5, 3.0 and 4.0, 32 bit and 64 bit.

6.2. Installation

Please install HQbird Enterprise from the supplied distributive. If you have other version of Firebird installed, uninstall it first.

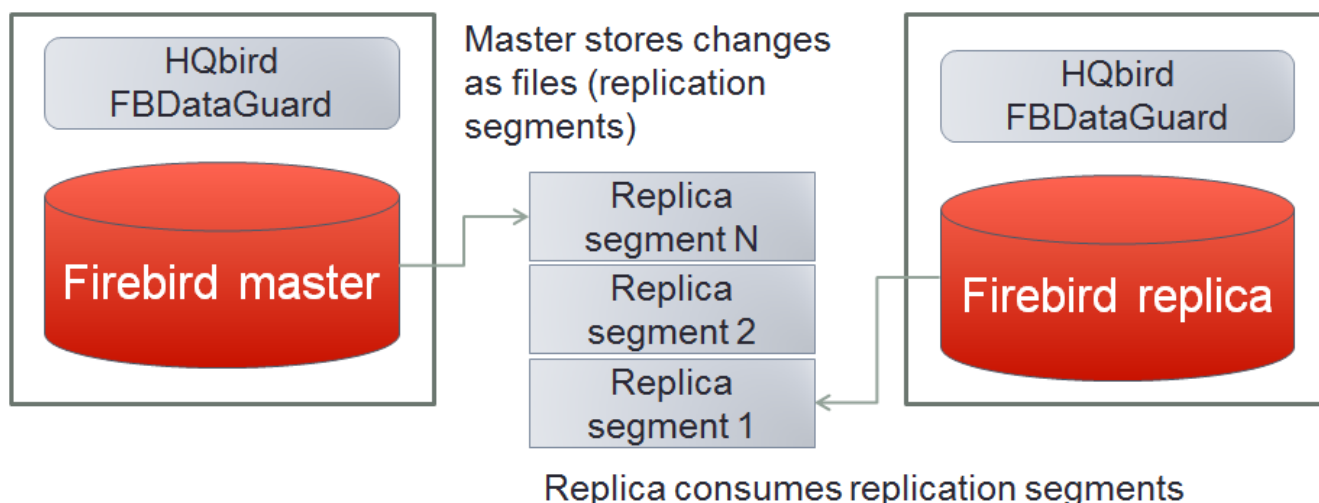
To enable replication you need to have working and registered (trial or full) copy of HQbird 2022 or higher on your master and replica servers.

Please refer to the section 2 of this guide for details of HQbird Server installation.

6.3. Asynchronous replication for Firebird

HQbird supports 2 types of replication: asynchronous and synchronous. In the case of an

asynchronous replication, the master server stores committed changes from the master database to the files (replication segments), which can be consumed asynchronously by one or more replica servers.



How the asynchronous replication works:

- Changes on the master side are journaled into the replication log files
- Journal consists of multiple segments (files)
- Replication (archived) segments are transferred to the slave and applied to the replica in the background
- Replica can be created and recreated online (without master's stop)

Important things to consider:

- Practical delay between master and replica is configurable, can be set to 15-30 seconds (default is 90 seconds)
- Delay between master and replica can grow in case of heavy load (due to the delayed processing of replication segments)
- Replica can be switched to the master (i.e., normal) mode with 1 command

Asynchronous replication is the recommended choice for HQbird Enterprise:

- it provides stability and anti-corruption protection of Firebird database,
- it can be configured quickly and easily,
- it does not require downtime to setup,
- it has online re-initialization (in HQbird 2020 and higher),
- it is suitable for distributed environments (when the replica is located in the cloud or at the remote location).

The following steps will be required to setup the asynchronous replication:

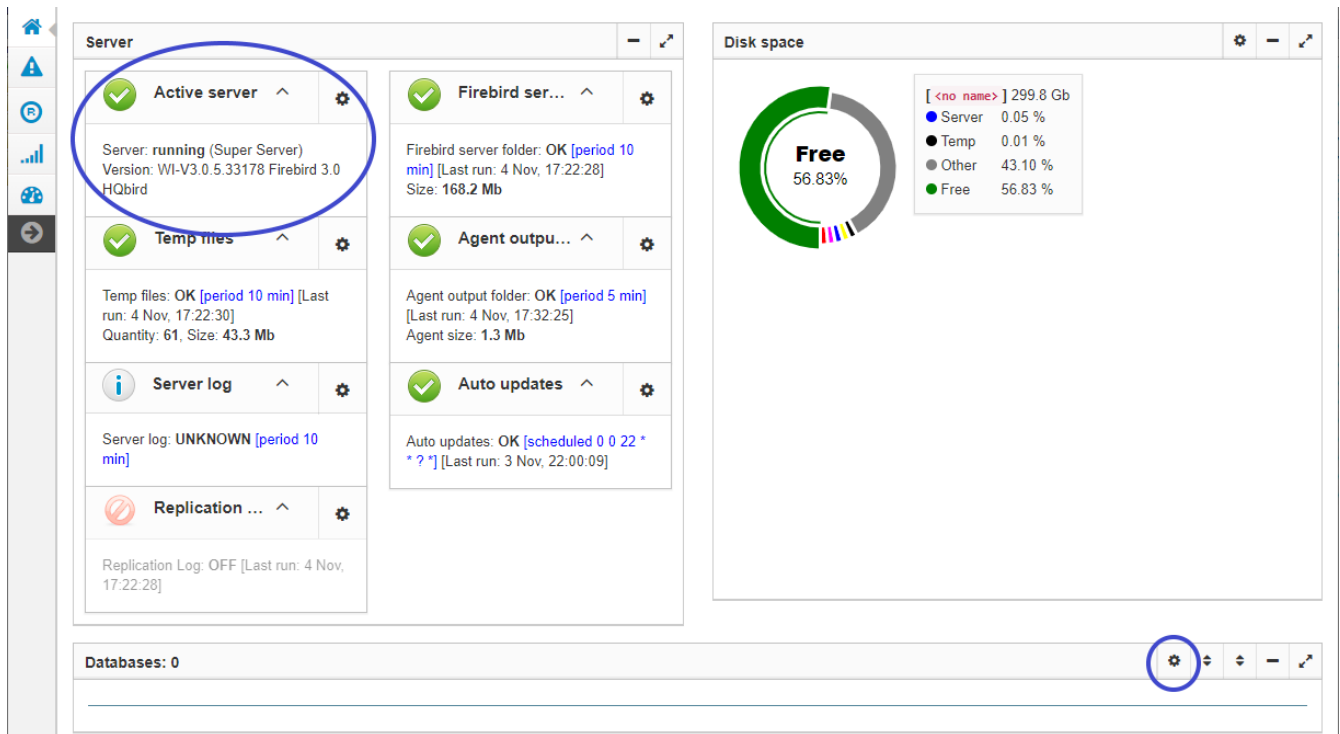
1. Configure HQbird for replication at the master
2. Create a copy of master database file

3. Setup database for replication at the replica(slave) server

6.3.1. Step 1: Configure HQbird for replication at the master

To setup replication, open HQbird FBDataGuard: run modern browser (Chrome, Firefox, etc) and open this local URL: <http://127.0.0.1:8082>(port if configurable in HQBird ini files)

Enter default name and password: **admin/strong password**. Register Firebird server, and the following picture will appear:



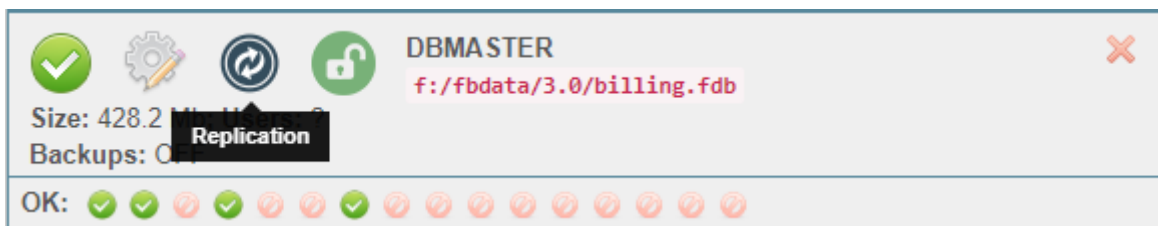
Check that you are actually connected to the correct Firebird version — in the upper left corner in “Active server” widget should be version “... Firebird 2.5 **HQbird**” or “... Firebird 3.0 **HQbird**” or “... Firebird 4.0 **HQbird**”.

After that click “Add database” in the right bottom corner and configure nick name and path to the database which will be master:



Please note that database should be registered with its explicit path, not with the alias — the replication will not work with the alias.

After the successful registration of the database click on the icon in the header of database to setup replication:



After that the main configuration dialog for master and replica databases will appear. When replication is not configured, this dialog is almost empty:

Asynchronous replication at master

Asynchronous replication writes all changes in the master database to the replication log: the set of files called “replication segments”. Replica server pulls these segments and inserts into the replica database.

Previously we have registered `H:|dbwmaster.fdb`, it is the asynchronous master database in this

example. To configure the asynchronous replication on the master side: select replication role: “Master”, then “Asynchronous”, and click “Save”.

Starting with Firebird 4.0, you need to enable publication at the database level. Clicking on the "Enable publications (and grant it for all tables)" button enables publication and adds all tables to the list for publication. The list of tables for publication can be manipulated in SQL using the following statements:

```
ALTER DATABASE INCLUDE {TABLE <table_list> | ALL} TO PUBLICATION
```

```
ALTER DATABASE EXCLUDE {TABLE <table_list> | ALL} FROM PUBLICATION
```

```
<table_list> ::= tablename [, tablename ...]
```

The screenshot shows a dialog box titled "Database replication configuration: 'billing'" with a close button (X) in the top right corner. Below the title, the Dataguard ID is displayed as "9b6e6443-de7e-428c-af75-a7b9b520f533".

The configuration options are as follows:

- Replication role:** Three radio buttons are present: "Off" (unselected), "Master" (selected), and "Replica" (unselected).
- Working mode:** Two radio buttons are present: "Synchronous" (unselected) and "Asynchronous" (selected).
- Write committed data every NN seconds:** A text input field contains the value "90".

Below the input field, there are four blue buttons stacked vertically: "more>>", "Reinitialize replica database", "See initialization status", and "Enable publications (and grant it for all tables)".

At the bottom right of the dialog, there are two buttons: "Close" and "Save".

Figure 44. Replication setup dialog for asynchronous replication

The only parameter you can change is “**Write committed data every NN seconds**”, it specifies how often we should move committed data to the archived replication segments.

By default, it is set to 90 seconds.

There are several optional parameters which you can change if you open detailed dialog with button “more>>”:

Database replication configuration: "billing"

Dataguard ID: 9b6e6443-de7e-428c-af75-a7b9b520f533

✕

Replication role Off Master Replica

Working mode Synchronous Asynchronous

Write committed data every NN seconds

Log directory 🗑

Log archive directory 🗑

Override log archive command 🗑

Optional parameters

Find and exclude tables without Primary or Unique keys

<<less

Reinitialize replica database

See initialization status

Enable publications (and grant it for all tables)

Close
Save

Let's consider all parameters in this dialog—just to give you idea what they do, **no need to change them**:


- “Log directory”—folder where operational logs will be stored. It is a system folder, completely operated by Firebird. By default, **no need to change its default value** `${db.path}.ReplLog` (db.path is where the database is located).
- “Log archive directory”—folder, where archived logs will be stored. According the default value `${db.path}.LogArch`, HQbird will create folder `DatabaseName.LogArch` in the folder with the database, so there is **no need to change this parameter**.
- The third parameter (“Override log archive command”) is optional, **leave it empty**.



Please note that replication parameters are initialized at the first connection to the database. That's why you need restart Firebird service (or all connections in case of Classic) after the replication configuration—such restart ensures that replication will start properly.

In this case, the replication log segments will be written first to `${db.path}.ReplLog` (db.path is where the database is located—in our example it will be `H:\DBWMaster.fdb.ReplLog`), and after reaching the maximum segment size, or commit, or another trigger, the default archive command will be started – it will copy archived replication segments to `${db.path}.LogArch` (in our example it will be `H:\DBWMaster.fdb.LogArch`).

After replication's start, you should be able to see replication segment files in the folder specified in “Log directory” immediately after any operation at master database:

Name	Date modified	Type	Size
 dbwmaster.fdb.log-000	05.09.2016 17:02	LOG-000 File	1 KB

The operational segments are rotated by the engine, and once each segment is completed, it is copied to archive log. Default segment size is 16Mb. Please note—you don't need to do anything with operational segments!

After the commit and/or specified timeout of committed data, you will see archived segments in the folder, specified by “Log archive directory”.

Archive replication log is essentially the chronologically ordered list of completed operational segments. These files should be imported by replica server into the replica database.

Important!



For Linux users—make sure that folder with the database is owned by firebird user. HQbird runs under “firebird” user in Linux, and the folder with the database must have permissions for “firebird” to create logs folder (`chown firebird -R /your/database/folder`).

How to copy replication segments from master server to the replicaserver?

There are 2 popular ways to copy archived segments from the master server to the replica server(s): through network share and using Cloud Backup on master and Cloud Backup Receiver on replica.

Network share

You can share the folder with archived segments as a network share. In this case, Firebird service should have enough rights to read, write and delete files on that network share. Normally Firebird and HQbird services are started under LocalSystem account, which do not have access to the network shares. Change it to some powerful account (like Domain Admin).

Cloud Backup/Cloud Backup Receiver

We recommend using HQbird FBDataGuard to send replication segments from the master server to the replica through FTP: it compresses, encrypts and uploads segments to the specified FTP server. On that server, another HQbird FBDataGuard unpacks segments and copies to the necessary folder for further consumption by the replica.

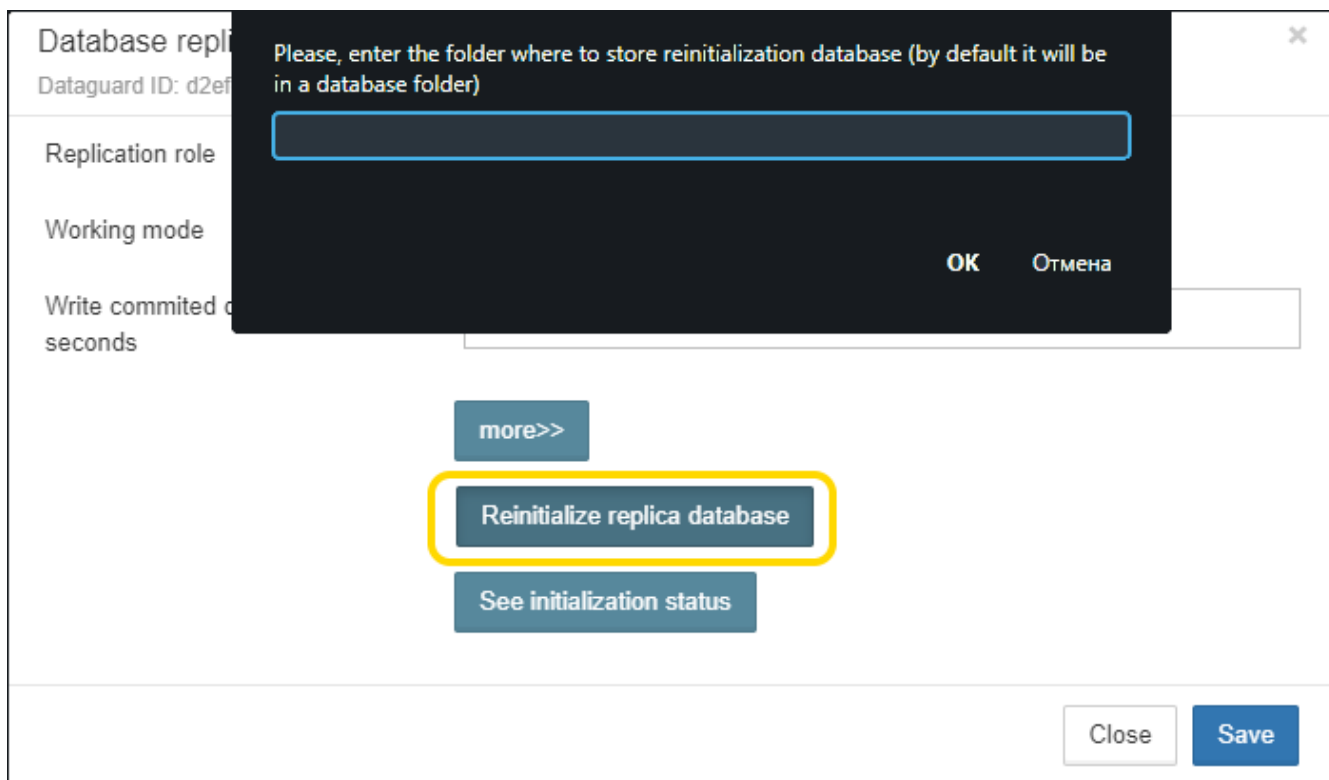


Please read about CloudBackup job for more details how to setup transfer of archived segments between master and replica(s).

6.3.2. Step 2: Create a copy of master database

To start replication we need to create an initial copy of the database file, which will be used as a target for the replication process. Let's refer to such database file as "replica".

Starting with HQbird 2018 R2, the replica will be created automatically in the folder which will you specify in the dialog after clicking on "Reinitialize replica database".



If you have enough space in the folder with the database, **just leave the path empty**, and click Ok, and replica will be created near the database. Or, you can specify other destination on the local drives with enough free space.



Important!

If there will be not enough free space (less than 105% of the database size), HQbird will not create replica copy — there will be an appropriate error message.

If you click Ok, HQbird will start the process of replica creation. There will be an appropriate message about it:

Replica initialization status ✕

Replication started 4 Nov, 18:30:00
 Source DB: f:\fbdata\3.0\billing.fdb
 Destination file: f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica
 DB GUID: 1B90FF37-9776-498D-58B0-5B3C597B1571

See initialization status

OK

In case of default action, the resulted database will be in the same folder with the database. The name of the replica will be DATABASE_NAME.EXT.DD-MMM-YYYY_NNNN.4replica — for example, *employee30.fdb.17-Apr-2018_142507.4replica*



Please note — creating of replica may take significant time in a case of the big database!

All stages of replica creation are listed as alerts in HQbird (also sent by email):

Order	Date	Severity	Source	Name	Desc
1	04/11/2019 18:30 GMT+0300	OK	DBMASTER	Replica (MASTER): reinitialization complete	Prepared replica database file: "f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica" is ready to send to replica via cloudbackup. Report file: "f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica.irreport"
2	04/11/2019 18:30 GMT+0300	OK	DBMASTER	Replica (MASTER): calculate file-hash finished	Finished calculate MD5 checksum for file "f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica.temp" in 00m 04s.638
3	04/11/2019 18:30 GMT+0300	OK	DBMASTER	Replica (MASTER): start calculate file-hash	Calculate MD5 checksum for file "f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica.temp" started at Mon Nov 04 18:30:20 MSK 2019
4	04/11/2019 18:30 GMT+0300	OK	DBMASTER	Replica (MASTER): initialization in process	Finished gfix -replica {1B90FF37-9776-498D-58B0-5B3C597B1571} "f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica.temp"
5	04/11/2019 18:30 GMT+0300	OK	DBMASTER	Replica (MASTER): initialization in process	Start gfix -replica {1B90FF37-9776-498D-58B0-5B3C597B1571} "f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica.temp"



Please make sure that replica creation process was completed successfully — check Alerts tab!

6.3.3. Step 3: Setup database for async replication at the replica(slave) server

After completing the configuration of asynchronous replication on the master server, we need to configure it for the replica database at the replica server instance.

First of all, we assume that you have successfully installed HQbird Enterprise on the replica server. We recommend to use on replica server SuperClassic for Firebird 2.5 and SuperServer for Firebird 3.0 (these are default configurations of HQbird Enterprise).

Firebird Classic Linux users: If you run Firebird on replica server in Classic mode on Linux, you need to run additional Firebird replicator process with the command `fb_smp_server -r`.

Second, the replica database should be registered in HQbird FBDataGuard. If you intend to use automatic re-initialization, you can register some small database (*employee.fdb*) with the required name, and the do re-initialization: as a result, replica database will be automatically transferred from the master server.

Third, we assume that you have managed to setup transfer of logs with Cloud Backup/Cloud Backup Receiver, or with network share.



Please note: the database should have replica database GUID before the registration! This GUID is created automatically if you have used link «Reinitialize replica database», but if you are performing manual re-initialization, don't forget to set it, otherwise will be an error about missing database GUID.

Then complete the replication setup—the only required parameter is a path to the folder with archived replication segments, and by default it is already set—HQbird will create folder with logs near the database:

Database replication configuration: "dbreplica" ✕

Dataguard ID: a57c3b53-3a02-4ed3-8f95-5b9a7e5e0dfe

Replication role Off Master Replica

Working mode Synchronous Asynchronous

Verbose

[more>>](#)

[Close](#) [Save](#)

So, no need to change anything here, just click Save.

Assuming the replica database is configured in *D:\DATABASE\DBWREPLICA.FDB*, the HQBird will create folder *D:\DATABASE\DBWREPLICA.FDB.LogArch*, and replica will import replication segment files from it.

Click “Save” and restart Firebird service (to ensure that replication parameters were applied).

After restart, the replica server will start to consume the replication segments from the folder—please note, after the import all processed segments will be deleted. Also, it will create file with the name {DATABASE-GUIDE}—Firebird stores there some internal information about replication progress.



It is not recommended to store archived replication segments from the different databases into the same folder! Always allocate the separate folder for each pair of master-replica databases!

6.4. Automatic initialization and re-initialization of replica

We recommend using Cloud Backup on the master and Cloud Backup Receiver on the replica to implement the transfer and check integrity of the replication segments through FTP. In this case, it

is also possible to implement 1-click re-initialization for the replica database.

If Cloud Backup and Cloud Backup Receiver have the following options enabled (by default), HQbird perform the re-initialization automatically, including restart of replica database:



Enable replication support

Name prefix to rename uploaded reini files:

Cancel Save

Parameter “Prefix to name uploaded reini files” should be changed if you intend to initialize several copies of the master database through the single folder – in this case set it should be unique for each database.

In case of the single database, no changes are required.

6.4.1. How re-initialization works

If Cloud Backup/Cloud Backup Receiver are configured, it is possible to perform the complete re-initialization with 1 click to “Reinitialize replica database”.

Once clicked, the master HQbird will do the following:

1. Ask you where to store copy of the database (by default it is near the master database, click Ok to store database there).
2. Master database will be copied (with nbackup)
3. The created copy of the database will be set to the replica mode
4. md5 hash-sum will be calculated for the copy
5. According the settings in Cloud Backup (Enable replication should be Enabled), master HQbird will upload database to the specified FTP

Next steps will be done by replica HQbird instance:

1. Once replica HQbird will notice the reini* files in the incoming FTP folder, Cloud Backup Receiver will start the procedure of re-initialization.
2. Processing if usual arch-segments will be stopped
3. The arrived database will be checked — md5 hash-sum will be calculated and compared with the value in the accompanied report file.
4. The existing replica database will be shutdown to disconnect all users
5. New replica database will be copied over the existing database
6. The replica server may require restart to see new replica.

Replica is back to the normal mode.

6.4.2. Troubleshooting asynchronous replication

If you have setup asynchronous replication, but it does not work, the first thing is to enable job “Replication Log” on the master and on the replica. This job parses *replication.log* files, and if there are errors, creates the appropriate alert.



Replication log monitoring

Enabled

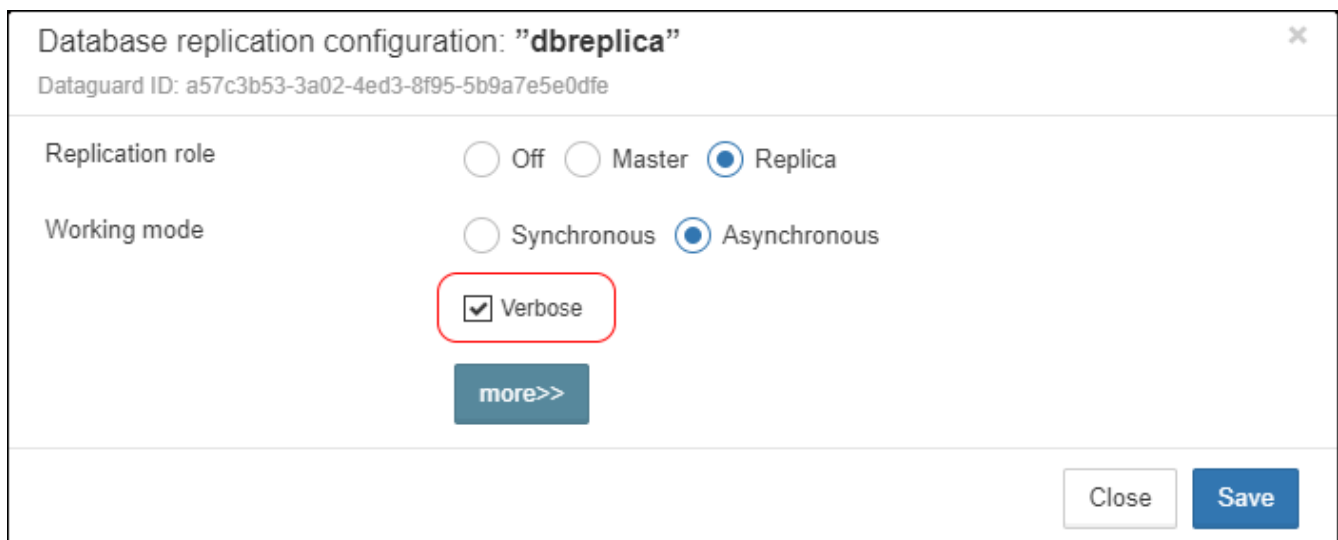
Check period, minutes: 10

Size to roll, bytes: 50000000

Date pattern for rolling: {0,date,yyyyMMdd_HH-mm-ss}

Cancel Save

Also, the good thing is to enable “Verbose” option on the replica, and restart Firebird. Verbose will make Firebird to write a lot of details about replication into the *replication.log* file (near *firebird.log*).



Database replication configuration: "dbreplica"

Dataguard ID: a57c3b53-3a02-4ed3-8f95-5b9a7e5e0dfe

Replication role Off Master Replica

Working mode Synchronous Asynchronous

Verbose

more>>

Close Save

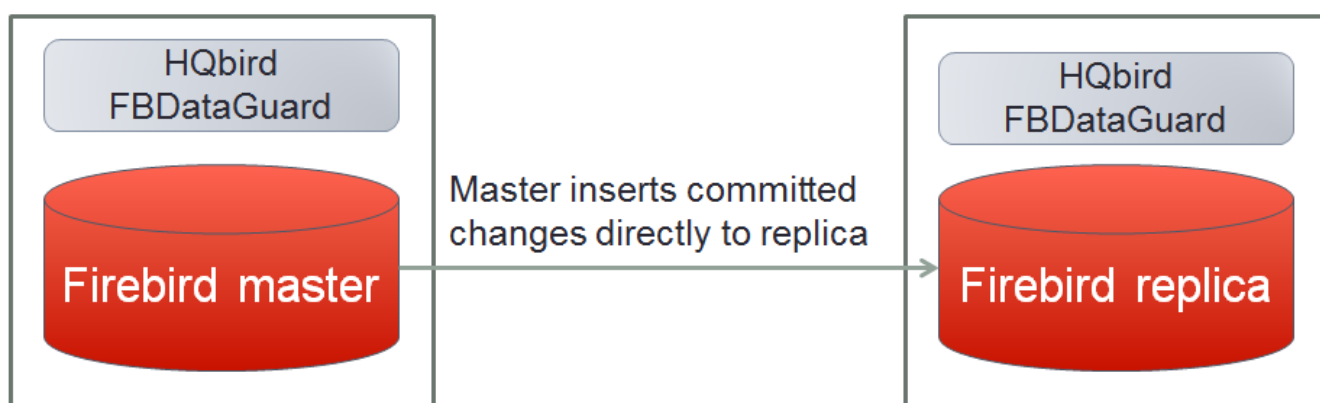
Usually the text of the error is self-explanatory, but since there are some popular questions which occur regularly, we decide to create the table with the list of main problems with asynchronous replication and ways to resolve it.

Problem	Possible reasons and how to resolve
Master part of replication was configured, but folders for operational or archived segments (<i>\${dbpath}.LogRepl</i> or <i>\${dbpath}.LogArch</i>) are not created	<p>HQbird creates these folders automatically, but it requires permissions.</p> <p>On Windows: these folders should be on local drives, or HQbird and Firebird services must run with “Run As” with the powerful account (Domain Admin?).</p> <p>On Linux: folders must have permissions for “firebird” user.</p>
Master part of replication was configured; folders for ReplLog and LogArch were created, but nothing appear there. <i>Replication.log</i> is empty.	Firebird does not see the replication configuration. Restart Firebird service (all connections in case of Classic) to make read the new configuration.
Master part of replication was configured; there are files <i>databasename.log-000</i> in ReplLog folder, but no files in LogArch. Also, could be errors about insufficient space or out of space in <i>replication.log</i>	<p>It means that there is no permission for Firebird to access the LogArch folder and create replication segment files (<i>databasename-logarch.000XXX</i>) there.</p> <p>If LogArch folder on the network share or mounted drive, make sure that Firebird has rights (full access)to access it.</p>
“Verbose” option on replica is enabled, but <i>replication.log</i> is empty or nor created.	Sometimes Firebird cannot create <i>replication.log</i> or even write to already created file. Try to create it manually and apply necessary permissions to it (especially on Linux). Verbose output should be written to the <i>replication.log</i> every 60 seconds even if there is no segments to import.
Master part of replication is Ok, but replica does not consume replication segments. <i>Replication.log</i> file is empty.	Replica did not read the new replication configuration. Restart Firebird.
Master part of replication is Ok, but replica does not consume replication segments. <i>Replication.log</i> contains errors about permissions.	Replica does not have enough permissions to read from the LogArch folder. Set necessary permissions or run replica under powerful account.
Replica has errors in <i>replication.log</i> “`Segment NNN is missing`”	Check is there such segment on the replica side, and if it is on the master size. If segment has size = 0 on replica, copy it manually or use “Perform fresh backup” checkmark in Cloud Backup.

Problem	Possible reasons and how to resolve
Replica has errors in <i>replication.log</i> about wrong foreign keys and stopped consume segments	It means that replica copy is desynchronized, so some records do not have the appropriate values in referenced tables for the specified Foreign Key. Replica should be reinitialized. If you see this errors often, please contact IBSurgeon support.

6.5. Synchronous replication for Firebird

In case of synchronous replication, master server directly inserts committed changes of the master database to one or more replicas databases:



The main features of the synchronous replication are the following:

- Changes are buffered per transaction, transferred in batches, synchronized at commit
- Practical delay is below 1 second
- Follows the master priority of locking
- Replication errors can either interrupt operations or just detach replica
- Replica is available for read-only queries (with caveats)
- Automatic fail-over can be implemented (with HQbird Cluster Manager)

Issues to be considered

- Additional CPU and I/O load on the replica side
- Requires direct and permanent network connection from master to replica(s), 1+Gbps recommended
- Replica can be recreated online, re-initialization of synchronous replication requires stop of master

When to use synchronous replication:

- Custom fail-over cluster solutions with 3+ nodes (especially for web applications)
- Scale performance by moving reads to the separate replica server (report servers, data marts or read-only web representation)

- In combination with asynchronous replication for performance scaling

6.5.1. Steps to setup synchronous replication

1. Stop Firebird
2. Create a copy of master database file, switch it to replica mode and copy it to the replica server(s)
3. Setup replica server(s) and database(s) for replication with HQbird FBDataGuard
4. Start replica server(s) — before master server!
5. Setup master server and master database for replication with HQBird FBDataGuard
6. Start master server

As you can see, the downtime required for initialization the synchronous replication is bigger than downtime to configure asynchronous replication, because replica database must be online before master's start.

6.5.2. Synchronous replication at master and replica

Synchronous replication is designed to write changes from the master database directly to the replica database. The big advantage of synchronous replication that replication delay can be very small, but the disadvantage is that in the case of the lost connection between master and replica servers there will be gaps in transmitted data.

The screenshot shows a configuration window titled "Database replication configuration: 'billing'". It includes a Dataguard ID and three main sections: "Replication role" with radio buttons for Off, Master (selected), and Replica; "Working mode" with radio buttons for Synchronous (selected) and Asynchronous; and "Connection to replica" with a text input field containing "sysdba:masterkey@replicaserver:/data/test2.fdb". Below the input field are buttons for "more>>" and "Enable publications (and grant it for all tables)". At the bottom right are "Close" and "Save" buttons.

In this example, the synchronous replica database is on the remote server with IP address **replica server** and path `/data/test2.fdb`.

No setup is necessary for synchronous replication on the replica server, except `gfix -replica {master-guid}` for the replica database to switch it to the replica mode.

6.5.3. Replication parameters for testing synchronous replication

In the case of testing synchronous replication of HQbird Enterprise on the production system, we recommend setting parameter *disable_on_error* to true.

Database replication configuration: "billing" ✕

Dataguard ID: 9b6e6443-de7e-428c-af75-a7b9b520f533

Replication role Off Master Replica

Working mode Synchronous Asynchronous

Connection to replica

Optional parameters

Find and exclude tables without Primary or Unique keys

<<less

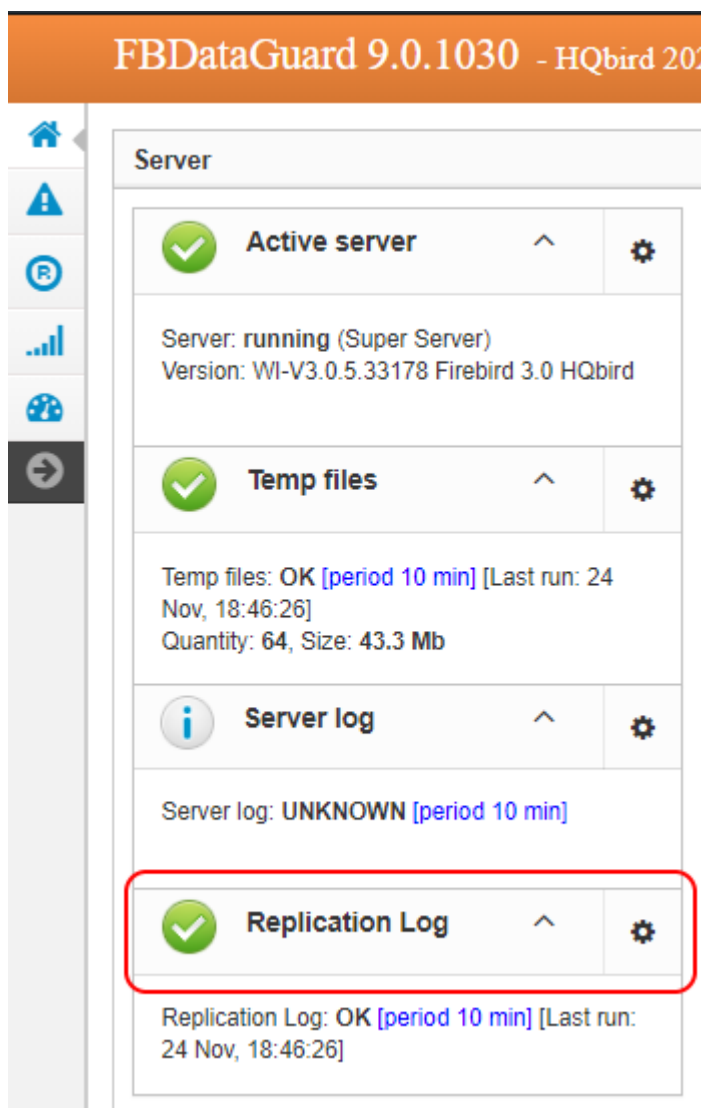
Enable publications (and grant it for all tables)

Close Save

It will switch off replication in case of replication error, and the master server will continue to work without replication.

To reinitialize replication the replication log should be analyzed and all initialization steps should be done again.

Also, please enable job "Replication log" in HQbird FBDataGuard to monitor replication log for errors and warnings:



6.6. How to manually create replica of the database?

Of course, it is always possible to create replica with the simple copy process: stop Firebird on master, copy database file, complete setup of replication on the replica, then start Firebird. However, HQbird supports online replica creation – see details below.

If, for some reason, you cannot use the automatic replica creation (which is available since v. 2018 R2), you can create replica copy of the master database manually.

Starting with HQbird 2018, it is possible to create replica file without stopping the master server, with nbackup. It is easy for asynchronous replication, and it also makes possible to create additional replicas online — i.e., without stopping a master.

6.6.1. Creating copy online (with nbackup)

Let's consider how to create replica for asynchronous replication using nbackup:

1. apply nbackup lock

```
nbackup -l database_path_name -user SYSDBA -pass masterkey
```

2. copy locked database file to create a replica

```
copy database_path_name replica_path_name
```

3. unlock master database

```
nbackup -n database_path_name -user SYSDBA -pass masterkey
```

4. Fixup replica database

```
nbackup -f replica_path_name_name
```

5. Switch database to replica mode

for Firebird 2.5 and 3.0

```
gfix replica_path_name -replica {DATABASEGUID} -user SYSDBA -pass masterkey
```

for Firebird 4.0

```
gfix replica_path_name -replica <replica_mode> -user SYSDBA -pass masterkey  
  
<replica_mode> ::= read_only | read_write
```

6.6.2. What is {DATABASEGUID}?

Database GUID is the unique identifier of a master database. To find out {DATABASEGUID}, run command `gstat -h`:

```

C:\HQbird\Firebird25\bin>gstat -h H:\DBWMASTER.FDB

Database "H:\DBWMASTER.FDB"
Database header page information:
  Flags                0
  Checksum             12345
  Generation           42984
  Page size            16384
  ODS version          11.2
  Oldest transaction   42600
  Oldest active        42601
  Oldest snapshot      42601
  Next transaction     42602
  Bumped transaction   1
  Sequence number      0
  Next attachment ID   1255
  Implementation ID    26
  Shadow count         0
  Page buffers         0
  Next header page     0
  Database dialect     3
  Creation date        Apr 2, 2014 0:17:50
  Attributes           force write

Variable header data:
  Database backup GUID: {AABAA4E7-D5D2-4E3D-BDB7-7594F48C9A04}
  Database GUID: {44206E92-025B-4E2C-A1B3-135783860326}
  Replication sequence: 2
*END*

```

To switch database to the replica mode run the following command:

```
gfix disk:\path\mydatabase.fdb -replica {guid} -user SYSDBA -pass masterkey
```



If you don't see Database GUID in `gstat -h` output, connect to the master database using Firebird binaries from HQbird distribution (with `isql` or any other application), and run `gstat -h` again.

6.6.3. How to set replica database to the master mode

To switch database to the normal (master) mode run the same command with the empty `{}` instead of database GUID:

for Firebird 2.5 and 3.0

```
gfix disk:\path\mydatabase.fdb -replica {} -user SYSDBA -pass masterkey
```


for Firebird 4.0

```
gfix replica_path_name -replica none -user SYSDBA -pass masterkey
```

6.7. How to distinguish master database from replica

6.7.1. Using gstat -h

If you run `gstat -h database_name`, the output will contain the keyword “replica” in Attributes section for database configured as replica:

```
Database "D:\030.FDB"
Gstat execution time Mon Nov 26 17:47:07 2018

Database header page information:
Flags                0
Generation           187842
System Change Number 15
Page size            8192
ODS version          12.0
Oldest transaction   173630
Oldest active        185440
Oldest snapshot      185440
Next transaction     185441
Sequence number      0
Next attachment ID   24975
Implementation        HW=AMD/Intel/x64 little-endian OS=Windows CC=MSVC
Shadow count         0
Page buffers         0
Next header page     0
Database dialect     3
Creation date        Jan 11, 2017 15:12:20
Attributes            replica

Variable header data:
Database backup GUID: {37E7918F-5478-43CF-E3B2-D80B0E7D3F63}
Sweep interval:      0
Database GUID:       {BBBD2881-ACDE-4636-CEB2-7EE31AF66CC3}
Replication master GUID: {BBBD2881-ACDE-4636-CEB2-7EE31AF66CC3}
*END*
Gstat completion time Mon Nov 26 17:47:07 2018
```

For master database there is no special marks in Attributes.

6.7.2. With SQL query to the context variable

In Firebird 2.5 and 3.0, there is a context variable REPLICAS in the SYSTEM area that contains

information about database status:

```
SQL> select RDB$GET_CONTEXT('SYSTEM', 'REPLICA') from rdb$database;
```

```
RDB$GET_CONTEXT
```

```
=====
FALSE
```

In Firebird 4.0 use another context variable REPLICA_MODE:

```
SQL> select RDB$GET_CONTEXT('SYSTEM', 'REPLICA_MODE') from rdb$database;
```

```
RDB$GET_CONTEXT
```

```
=====
READ-ONLY
```

Also in Firebird 4.0 you can use the MON\$DATABASE monitoring table:

```
SQL> SELECT MON$REPLICA_MODE FROM MON$DATABASE;
```

```
MON$REPLICA_MODE
```

```
=====
1
```

Database replica mode:

- 0 - not a replica
- 1 - read-only replica
- 2 - read-write replica

6.8. Optional parameters for replication

It is possible to specify several additional parameters for fine tuning of the replication process. These parameters can be specified in the “Optional parameters” of replication setup dialog.

1. Size of the local buffer used to accumulate replication events that can be deferred until the transaction commit/rollback. The bigger this value the less network round-trips between master and slave hosts are performed. However, it costs longer replication “checkpoints” (time to synchronize the original database with its replica).

```
buffer_size = 1048576
```

2. If enabled, any error during replication causes the master to stop replicating changes and continue working normally. Otherwise (the default behavior), the master reports an error.

```
disable_on_error = false
```

3. If enabled, replicated records are RLE-compressed before transmission and decompressed on the slave side. It reduces the traffic and (indirectly) a number of round-trips at the cost of extra CPU cycles on both sides.

```
compress_records = false
```

4. If enabled, conflicting records in the target database are modified to match records in the master database. In particular:
 - if there's an insert and the target record exists, it gets updated;
 - if there's an update and the target record does not exist, it gets inserted;
 - if there's a delete and the target record does not exist, it gets ignored.

```
master_priority = false
```

1. Pattern (regular expression) that defines what tables must be included into replication. By default, all tables are replicated.

```
include_filter
```

2. Pattern (regular expression) that defines what tables must be excluded from replication. By default, all tables are replicated.

```
exclude_filter
```

3. If enabled, tables without primary key (or unique index) excluded from replication. By default, all tables are replicated.

```
exclude_without_pk = false
```

4. Program (complete command line with arguments) that is executed when the current replication session notices a critical error. This command is executed once per every failed replication session. Please note that the program is executed synchronously and the server is waiting for its completion before continuing its operations.

```
alert_command
```

5. Prefix for replication log file names. It will be automatically suffixed with an ordinal sequential number. If not specified, database filename (without path) is used as a prefix.

```
log_file_prefix
```

6. Maximum allowed size for a single replication segment. It must at least double the specified *buffer_size*.

```
log_segment_size = 16777216
```

7. Maximum allowed number of full replication segments. Once this limit is reached, the replication process is delayed for *log_archive_timeout* seconds (see below) to allow the archiving to catch up. If any of the full segments is not archived and marked for reuse during the timeout, the replication fails with an error.

Zero means an unlimited number of segments pending archiving.

```
log_segment_count = 8
```

Chapter 7. Performance enhancements

7.1. Pool of external connections

HQbird 2018 supports a pool of external connections for Firebird 2.5 and (from 2018R3) in Firebird 3. Firebird 4.0 standard build supports external connection pooling out of the box. This pool allows running parallel EXECUTE ON EXTERNAL statements to external Firebird databases.



Please note — this pool is allocated per Firebird instance.

The feature is managed in the *firebird.conf*:

```
# =====
# Settings of External Connections Pool
# =====

# Set the maximum number of inactive (idle) external connections to retain at
# the pool. Valid values are between 0 and 1000.
# If set to zero, pool is disabled,
# i.e. external connection is destroyed immediately after the use.
#
# Type: integer
#
#ExtConnPoolSize = 0

# Set the time before destroying inactive external connection, seconds.
# Valid values are between 1 and 86400.
#
# Type: integer
#
#ExtConnPoolLifeTime = 7200
```

From the application point of view, no additional steps are required to use or do not use—it is enabled or disabled in the server configuration, and absolutely seamless for the applications.

The following commands exist to manage pool:

- changes the pool size

```
ALTER EXTERNAL CONNECTIONS POOL SET SIZE N
```

Example — this command sets the size of a pool to 190 connections.

```
ALTER EXTERNAL CONNECTIONS POOL SET SIZE 190
```

- changes the lifetime of the pooled connection

```
ALTER EXTERNAL CONNECTIONS POOL
SET LIFETIME N {SECOND | MINUTE | HOUR}
```

Example — this command limits the lifetime of a connection in the pool to 1 hour.

```
ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME 1 HOUR
```

- clear all pooled connections

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL
```

- clear the oldest connection in the pool

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST
```

To get information about pool status, new context variables were introduced. The following example demonstrates their usage

```
SELECT
  CAST(RDB$GET_CONTEXT('SYSTEM', 'EXT_CONN_POOL_SIZE') AS INT) AS POOL_SIZE,
  CAST(RDB$GET_CONTEXT('SYSTEM', 'EXT_CONN_POOL_IDLE_COUNT') AS INT) AS POOL_IDLE,
  CAST(RDB$GET_CONTEXT('SYSTEM', 'EXT_CONN_POOL_ACTIVE_COUNT') AS INT) AS POOL_ACTIVE,
  CAST(RDB$GET_CONTEXT('SYSTEM', 'EXT_CONN_POOL_LIFETIME') AS INT) AS POOL_LIFETIME
FROM RDB$DATABASE;
```

7.2. Cached prepared statements

HQbird 2018 has the feature to improve the performance of Firebird (version 3+ only!) engine in case of the many frequent and fast SQL queries: server-side cache of prepared SQL statements.

This feature can be enabled in *firebird.conf* with the parameter `DSQLCacheSize`:

```
# Size of DSQL statements cache.
# Maximum number of statements to cache.
# Use with care as it is per-attachment and could lead to big memory usage.
# Value of zero disables caching.
# Per-database configurable.
# Type: integer
#DSQLCacheSize = 0
```

The number specifies how many recent queries for each database connection to cache.

To apply the new value, Firebird restart is required.

By default, cache of prepared statements is 0, it means OFF. We recommend to carefully enable it: start with values like 4, 8, 16, to find the best performance effect.



Please note: enabling cache of prepared statements increases the memory usage.

7.3. TempSpaceLogThreshold: monitoring of big sorting queries and BLOBs

HQbird Enterprise has a new parameter in *firebird.conf* in Firebird 2.5, Firebird 3.0 and Firebird 4.0:

```
TempSpaceLogThreshold=1000000000 #bytes
```

When Firebird sees this parameter, it starts to log to *firebird.log* queries which produce large sortings: queries with GROUP BY, ORDER BY, etc.

When such query creates the sorting file which exceeds the specified threshold, the following message will appear in *firebird.log*:

```
SRV-DB1 Wed Nov 28 21:55:36 2018
  Temporary space of type "sort" has exceeded threshold of 1000000000 bytes.
  Total size: 10716980736, cached: 1455423488 bytes, on disk: 9263120384 bytes.
  Query: select count(*) from (select lpad(' ',1000,uuid_to_char(gen_uuid())) s
    from rdb$types a,rdb$types b, rdb$types c order by 1)
```

Total size — the total size of sorting file

Cached — the part of sorting which had fit into temporary space (specified by TempCacheLimit parameter)

On disk — the part of sorting which was stored to the temporary file, which can be cached in the OS memory, or stored on disk (in the folder specified by TempDirectories parameter, or in the default temp folder)

For very big BLOBs the following message will appear in the *firebird.log*

```
SRV-DB1 Tue Nov 27 17:35:39 2018
  Temporary space of type "blob" has exceeded threshold of 5000000000 bytes.
  Total size: 500377437, cached: 0 bytes, on disk: 501219328 bytes.
```

Use TempSpaceLogThreshold to find the non-optimized queries with big sortings and big BLOBs. In Firebird 3.0, it also will report large hash sortings.

If you encounter such queries, optimize them either with redesign of SQL query itself, or try to enable parameter SortDataStorageThreshold.

7.4. SortDataStorageThreshold: REFETCH instead SORT for wide record sets

HQbird Enterprise supports the new REFETCH optimization method. The standard build of Firebird 4.0 supports this optimization algorithm out of the box.

HQbird Enterprise has a new parameter `SortDataStorageThreshold` in *firebird.conf* (Firebird 3.0+):

```
SortDataStorageThreshold=16384 # bytes
```

If the size of the record, returned by SQL query, will be more than specified threshold, Firebird will use the different approach for sorting record sets: REFETCH instead of SORT.

For example, we have the following query

```
select tdetl.name_detl
       ,tmain.name_main
       ,tdetl.long_description
from tdetl
join tmain on tdetl.pid=tmain.id
order by tdetl.name_detl
```

with the following execution plan:

```
Select Expression
  -> Sort (record length: 32860, key length: 36)
    -> Nested Loop Join (inner)
      -> Table "TMAIN" Full Scan
      -> Filter
        -> Table "TDETL" Access By ID
          -> Bitmap
            -> Index "FK_TABLE1_1" Range Scan (full match)
```

In this case, the size of each record to be sorted is 32860+36 bytes. It can lead to the very big sort files, which will be written to the disk, and the overall query can slow.

With parameter `SortDataStorageThreshold=16384`, Firebird will use plan REFETCH, where only key is sorted, and data are read from the database:

```
Select Expression
  -> Refetch
    -> Sort (record length: 76, key length: 36)
      -> Nested Loop Join (inner)
```

This approach can significantly (2-5 times) speed up queries with very wide sorted record sets

(usually, heavy reports).



Please note!

It is not recommended to set `SortDataStorageThreshold` less than 2048 bytes.

7.5. Multi-thread sweep, backup, restore

In HQbird 2020+, the possibility of multi-threaded execution of sweep, backup and restore has appeared, which speeds up their work from 2x to 6 times (depending on the specific database). Multi-threaded operations work in HQbird Firebird 2.5 and 3.0 (starting from builds 2.5.9.27143 and 3.0.5.3.31717 respectively), in any architectures — Classic, SuperClassic, SuperServer.

To enable multi-threaded execution, the `gfix` and `gbak` command-line utilities have the `-par n` option, where *n* is the number of threads that will be involved in a particular operation. In practice, choosing the number *n* should be correlated with the number of available processor cores.

For example

- `gfix -sweep database -par 8 ...`
- `gbak -b database backup -par 8 ...`
- `gbak -c backup database -par 8 ...`

Also, to control the number of threads and set their default number in *firebird.conf*, two new parameters are introduced that affect only sweep and restore, but not backup:

```
# =====
# Settings for parallel work
# =====
# Limit number of parallel workers for the single task. Per-process.
# Valid values are from 1 (no parallelism) to 64. All other values
# silently ignored and default value of 1 is used.
MaxParallelWorkers = 64
```

Example: if you set `MaxParallelWorkers = 10`, then you can

- run `gfix -sweep database -par 10`
- run `gfix -sweep database -par 5` and `gbak -c -par 5 ...`

That is, no more than 10 threads will be used in total. In case of exceeding (for example, if you set 6 threads for sweep and 6 threads for restore), for a process that exceeds the limit, the message “No enough free worker attachments” will be displayed).

Thus, to enable the multi-threaded capabilities of sweep and restore, you must set the `MaxParallelWorkers` parameter in *firebird.conf*

```
MaxParallelWorkers = 64
```

and then restart Firebird.

The `ParallelWorkers` sets the number of threads used by `sweep` and `restore` by default if the `-par n` option is not specified.

```
# Default number of parallel workers for the single task. Per-process.
# Valid values are from 1 (no parallelism) to MaxParallelWorkers (above).
# Values less than 1 is silently ignored and default value of 1 is used.
#
ParallelWorkers = 1
```

For example, if `ParallelWorkers = 8`, then starting

```
gfix -sweep
```

without the `-par n` option will use 8 threads to execute `sweep` in parallel.



For `restore`, filling tables from backup is always performed in one thread, and only creating indexes is parallelized. Thus, the acceleration for `restore` depends on the number of indexes in the database and their size. Also, the `ParallelWorkers` parameter automatically affects the creation of indexes performed by the `CREATE INDEX` and `ALTER INDEX ... ACTIVE` operations.

As mentioned above, these options do not affect backup. The multi-threading of backup is regulated only by the `-par n` parameter in the command line:

- `gbak -b -par 6 ...`
- `gbak -b -par 8 -se ...`

If the database is in shutdown single state, when only 1 connection is allowed to the database, then in version 2.5 both sweep and backup with `-par 2` or more will produce an error several seconds after starting:

- sweep — connection lost to database
- backup — ERROR: database ... shutdown (via xnet protocol, a line with this message will not be displayed in the backup log)



This is due to the fact that for these operations an appropriate number of database connections is required, more than 1.

In 3.0, only backup will throw an error “ERROR: database ... shutdown”, sweep will work.

Multi-threaded restore, Firebird 2.5, 3.0 and 4.0, creates the database in shutdown multi mode, so such errors do not occur. However, there is a risk of connecting other applications from SYSDBA or the owner to the database in the restore process.

Notes



- The new parameters in *firebird.conf* only affect sweep and restore, to simplify administration and eliminate ambiguity, it is recommended that you always explicitly specify the `-par n` parameter for `gfix` and `gbak` if you need to perform multi-threaded sweep, restore, and backup operations. For example, if you set `ParallelWorkers = 4` and do not specify `-par n`, then sweep and restore will use 4 threads by default, and backup will use 1 thread, because it does not use the values from *firebird.conf* neither locally nor with `-se`.
- The performance improvement does not necessarily depend on the number of processor cores and their compliance with the set value `-par n`. It depends on the number of cores, the Firebird architecture, and the disk subsystem performance (IOPS). Therefore, the optimal value `-par n` for your system must be selected experimentally.

7.6. BLOB_APPEND function

Regular operator `||` (concatenation) with BLOB arguments creates temporary BLOB per every pair of args with BLOB. This could lead to the excessive memory consumption and growth of database file. The `BLOB_APPEND` function is designed to concatenate BLOBs without creating intermediate BLOBs.

In order to achieve this, the result BLOB is left open for writing instead of been closed immediately after it is filled with data. I.e. such blob could be appended as many times as required. Engine marks such blob with new internal flag `BLB_close_on_read` and closes it automatically when necessary.

Available in: DSQL, PSQL.

Syntax:

```
BLOB_APPEND(<blob> [, <value1>, ... <valueN>]
```

Table 5. Parameters of *BLOB_APPEND* function

Parameter	Description
blob	BLOB or NULL.
value	Any type of value.

Return type: BLOB, temporary, not closed (i.e. open for writing), marked by flag `BLB_close_on_read`.

Input Arguments:

- The first argument is BLOB or NULL. The following options are possible:
 - NULL: creates new temporary blob, not closed, with flag `BLB_close_on_read`
 - permanent BLOB (from table) or temporary already closed BLOB: will create a new empty unclosed BLOB with the flag `BLB_close_on_read` and the contents of the first BLOB will be added to it
 - temporary unclosed BLOB with the `BLB_close_on_read` flag: it will be used further
- other arguments can be of any type. The following behavior is defined for them:
 - NULL ignored
 - non-BLOBs are converted to string (as usual) and appended to the content of the result
 - BLOBs, if necessary, are transliterated to the character set of the first argument and their contents are appended to the result

The `BLOB_APPEND` function returns a temporary unclosed BLOB with the ``BLB_close_on_read`` flag. This is either a new BLOB or the same as in the first argument. Thus, a series of operations like `blob = BLOB_APPEND (blob, ...)` will result in the creation of at most one BLOB (unless you try to add a BLOB to itself). This BLOB will be automatically closed by the engine when the client tries to read it, assign it to a table, or use it in other expressions that require reading the content.



Testing a BLOB for NULL value using the `IS [NOT] NULL` operator does not read it, and therefore a temporary BLOB with the ``BLB_close_on_read`` flag will not be closed during such test.

```
execute block
returns (b blob sub_type text)
as
begin
  -- will create a new temporary not closed BLOB
  -- and will write to it the string from the 2nd argument
  b = blob_append(null, 'Hello ');
  -- adds two strings to the temporary BLOB without closing it
  b = blob_append(b, 'World', '!');
  -- comparing a BLOB with a string will close it, because for this you need to read
  the BLOB
  if (b = 'Hello World!') then
  begin
    -- ...
  end
  -- will create a temporary closed BLOB by adding a string to it
  b = b || 'Close';
  suspend;
end
```



Use the `LIST` and `BLOB_APPEND` functions to concatenate BLOBs. This will save memory consumption, disk I/O, and prevent database growth due to the creation of many temporary BLOBs when using concatenation operators.

Let's say you need to build JSON on the server side. We have a PSQL package JSON_UTILS with a set of functions for converting primitive data types to JSON notation. Then the JSON building using the BLOB_APPEND function will look like this:

```
EXECUTE BLOCK
RETURNS (
    JSON_STR BLOB SUB_TYPE TEXT CHARACTER SET UTF8)
AS
DECLARE JSON_M BLOB SUB_TYPE TEXT CHARACTER SET UTF8;
BEGIN
FOR
    SELECT
        HORSE.CODE_HORSE,
        HORSE.NAME,
        HORSE.BIRTHDAY
    FROM HORSE
    WHERE HORSE.CODE_DEPARTURE = 15
    FETCH FIRST 1000 ROW ONLY
    AS CURSOR C
DO
BEGIN
    SELECT
        LIST(
            '{' ||
            JSON_UTILS.NUMERIC_PAIR('age', MEASURE.AGE) ||
            ',' ||
            JSON_UTILS.NUMERIC_PAIR('height', MEASURE.HEIGHT_HORSE) ||
            ',' ||
            JSON_UTILS.NUMERIC_PAIR('length', MEASURE.LENGTH_HORSE) ||
            ',' ||
            JSON_UTILS.NUMERIC_PAIR('chestaround', MEASURE.CHESTAROUND) ||
            ',' ||
            JSON_UTILS.NUMERIC_PAIR('wristaround', MEASURE.WRISTAROUND) ||
            ',' ||
            JSON_UTILS.NUMERIC_PAIR('weight', MEASURE.WEIGHT_HORSE) ||
            '}'
        ) AS JSON_M
    FROM MEASURE
    WHERE MEASURE.CODE_HORSE = :C.CODE_HORSE
    INTO JSON_M;

    JSON_STR = BLOB_APPEND(
        JSON_STR,
        IIF(JSON_STR IS NULL, '[', ',' || ascii_char(13)),
        '{',
        JSON_UTILS.INTEGER_PAIR('code_horse', C.CODE_HORSE),
        ',',
        JSON_UTILS.STRING_PAIR('name', C.NAME),
        ',',
```

```

JSON_UTILS.TIMESTAMP_PAIR('birthday', C.BIRTHDAY),
',',
JSON_UTILS.STRING_VALUE('measures') || ':[', JSON_M, ']',
'}'
);
END
JSON_STR = BLOB_APPEND(JSON_STR, ']');
SUSPEND;
END

```

A similar example using the usual concatenation operator `||` is an order of magnitude slower and does 1000 times more disk writes.

7.7. Transform LEFT joins into INNER

HQbird Enterprise allow transform LEFT joins into INNER ones if the WHERE condition violates the outer join rules.

Example:

```

SELECT *
FROM T1 LEFT JOIN T2 ON T1.ID = T2.ID
WHERE T2.FIELD1 = 0

```

In this case the condition `T2.FIELD1 = 0` effectively removes all the "fake NULL" rows of T2, so the result is the same as for the INNER JOIN. However, the optimizer is forced to use the `T1 → T2` join order while `T2 → T1` could also be considered. It makes sense to detect this case during join processing and internally replace LEFT with INNER before optimization starts.

This is primarily intended to improve "ad hoc" and machine-generated (e.g. ORM) queries.

This optimization will not be enabled if a NULL value is checked, for example

```

SELECT *
FROM T1 LEFT JOIN T2 ON T1.ID = T2.ID
WHERE T2.ID IS NULL

```



or

```

SELECT *
FROM T1 LEFT JOIN T2 ON T1.ID = T2.ID
WHERE T2.ID IS NOT NULL

```

Chapter 8. Encryption support

Since HQbird 2020, HQbird Enterprise includes the encryption plugin and provides support to work with encryption databases in web interface. This feature is supported only in HQBird Enterprise with Firebird 3.0.

**Please note!**

The encryption plugin requires the separate license file, it is sent in the purchase email.

**Please note!**

The encryption plugin can be purchased separately and used with community version of Firebird: <https://ib-aid.com/download-demo-firebird-encryption-plugin>

8.1. OpenSSL files

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)

HQbird Enterprise for Windows already includes necessary binary files from OpenSSL 1.1, in order to use encryption features on Linux it is necessary to install **OpenSSL 1.1**.

8.1.1. How to encrypt and decrypt Firebird database

In this short guide below, we will demonstrate the key features of the encryption: how to encrypt your Firebird database on the server, how to implement an encrypted client connection, and perform backup/restore of the encrypted database. The demo of FEFP is fully functional, with the only exception — it is limited until December 30, 2019.

Demo package with client applications examples

Please download demo package with client applications examples from <https://ib-aid.com/download/crypt/CryptTest.zip>

Stage 1 — Initial encryption of the database

At the point, we suppose that you have some database to be encrypted. Put unencrypted database to some path, for example, into `c:\temp\employee30\employee.fdb`

We suppose that all actions are made with 64-bit version of the Firebird 3.0.3+, in case of 32-bit version simply use the files from WinSrv32bit_ServerPart folder.

1. Create the following alias in databases.conf


```
crypt = C:\Temp\EMPLOYEE30\EMPLOYEE30.FDB
{
  KeyHolderPlugin = KeyHolder
}
```

Also, you can declare KeyHolder plugin for all databases at the server, for this add the following parameter to *firebird.conf*:

```
KeyHolderPlugin = KeyHolder
```

or, simply copy *firebird.conf* at step 2 (see below).

2. Check that the following files to server/plugins from the folder WinSrv64Bit_ServerPart\plugins

- *DbCrypt.dll*
- *DbCrypt.conf*
- *KeyHolder.dll*
- *KeyHolder.conf*— this is the text file with keys, it is only for developer's usage, it should not be sent to end users!

3. Put the following files into Firebird root from the folder WinSrv64Bit_ServerPart

- *fbcrypt.dll*
- *libcrypto-1_1-x64.dll*
- *libssl-1_1-x64.dll*
- *gbak.exe*
- *firebird.msg*
- *firebird.conf* (optional, can be used as an example)

4. Connect to the unencrypted database with isql and encrypt the database:

```

isql localhost:C:\Temp\EMPLOYEE30\EMPLOYEE30.FDB -user SYSDBA -pass masterkey
SQL>alter database encrypt with dbcrypt key red;
SQL> show database;
Database: localhost:C:\Temp\EMPLOYEE30\EMPLOYEE30.FDB
      Owner: ADMINISTRATOR
PAGE_SIZE 8192
Number of DB pages allocated = 326
Number of DB pages used = 301
Number of DB pages free = 25
Sweep interval = 20000
Forced Writes are OFF
Transaction - oldest = 2881
Transaction - oldest active = 2905
Transaction - oldest snapshot = 2905
Transaction - Next = 2909
ODS = 12.0
Database encrypted
Default Character set: NONE

```

Let's consider the encryption command:

```
alter database encrypt with dbcrypt key red;
```

Here, dbcrypt is the name of the encryption plugin, and red is the name of the key to being used. Keys are defined in *KeyHolder.conf* file.

Please note — on Linux it is necessary to use quotes and case-sensitive plugin name:

```
alter database encrypt with "DbCrypt" key Red;
```

After that, the database is encrypted with server-side authentication: the keys are located in the file *KeyHolder.conf*.

At the figure below you can see files we have on the server to enable the encryption, and what we need on the client side:

On the server's side:	On the client's side (demo - CryptTest.exe) - 32bit
Mandatory files:	Mandatory files:
<i>plugins/dbcrypt.dll</i>	<i>fbclient.dll</i>
<i>plugins/keyholder.dll</i>	<i>fbcrypt.dll</i>
<i>DbCrypt.conf</i>	<i>libcrypto-1_1.dll</i>
<i>libssl-1_1-x64.dll</i>	Optional files:
<i>libcrypto-1_1-x64.dll</i>	<i>firebird.conf</i>

On the server's side:	On the client's side (demo - CryptTest.exe) - 32bit
<i>fbcrypt.dll</i>	
Files for gbak with encryption:	
<i>gbak.exe</i>	
<i>firebird.msg</i>	
Optional files:	
<i>plugins/KeyHolder.conf</i> (for initial encryption in development mode)	
<i>firebird.conf</i> (contains parameter to set encryption plugin)	

Stage 2 — Connect to the encrypted database with the client application

After the initial encryption, we suppose that the database will be copied to the customer environment, where access to it will be done only through the authorized application.

To imitate such environment, we need to remove (or simply rename) the file with keys (*KeyHolder.conf*) from the folder *plugins*.

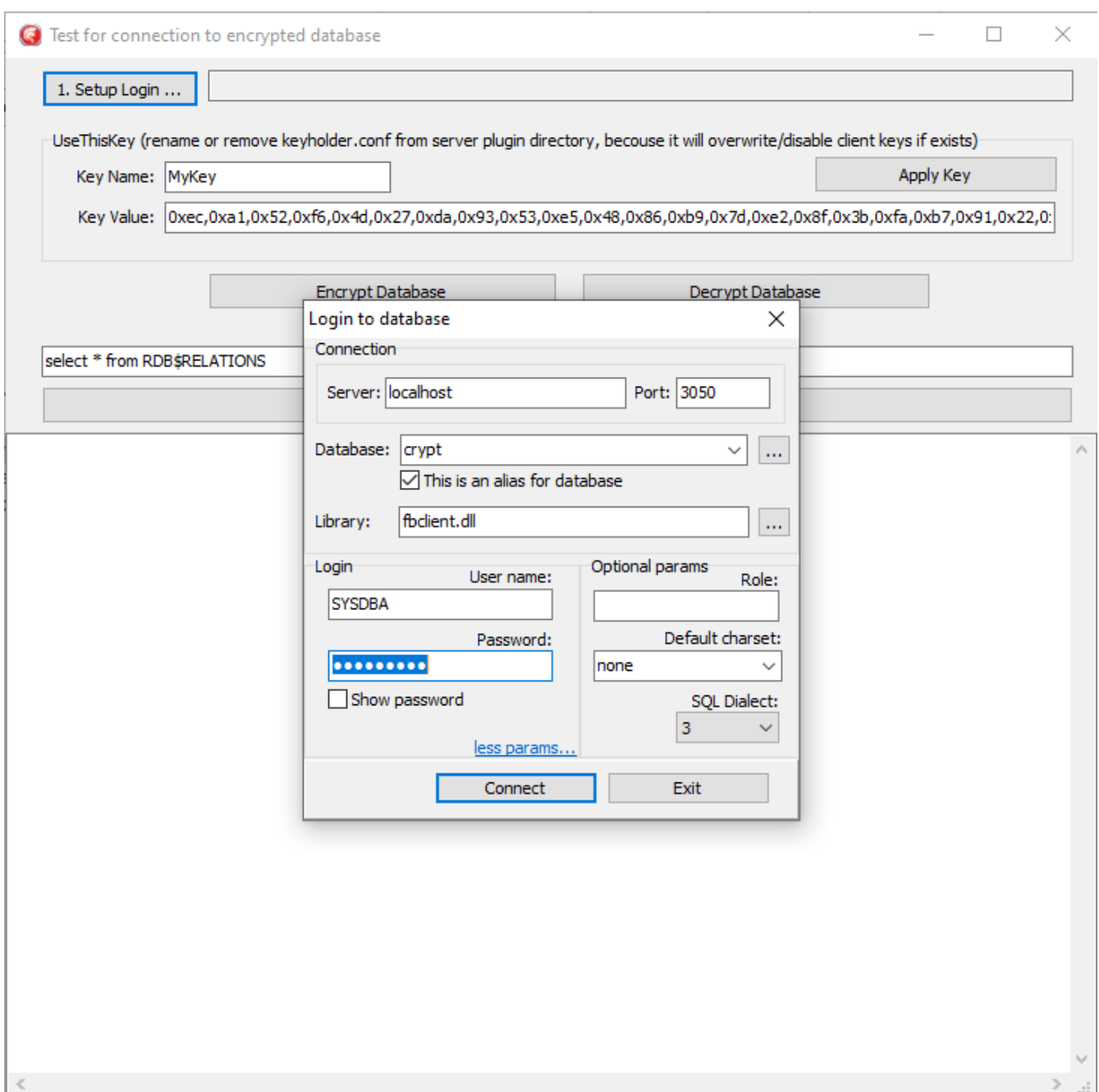
Without *KeyHolder.conf*, the encryption plugin will require receiving the key from the connected application. The example of such application is included in the archive with demo plugin — there is a compiled version and full sources for it on Delphi XE8.

The code to initialize encrypted connection is very simple — before the usual connection, several calls should be done to send an appropriate key. After that, the client application works with Firebird as usual.

Run the demo application to test the work with the encrypted database, it is in the folder *CryptTest|EnhancedCryptTestClient|Win32|Debug*.

Do the following steps:

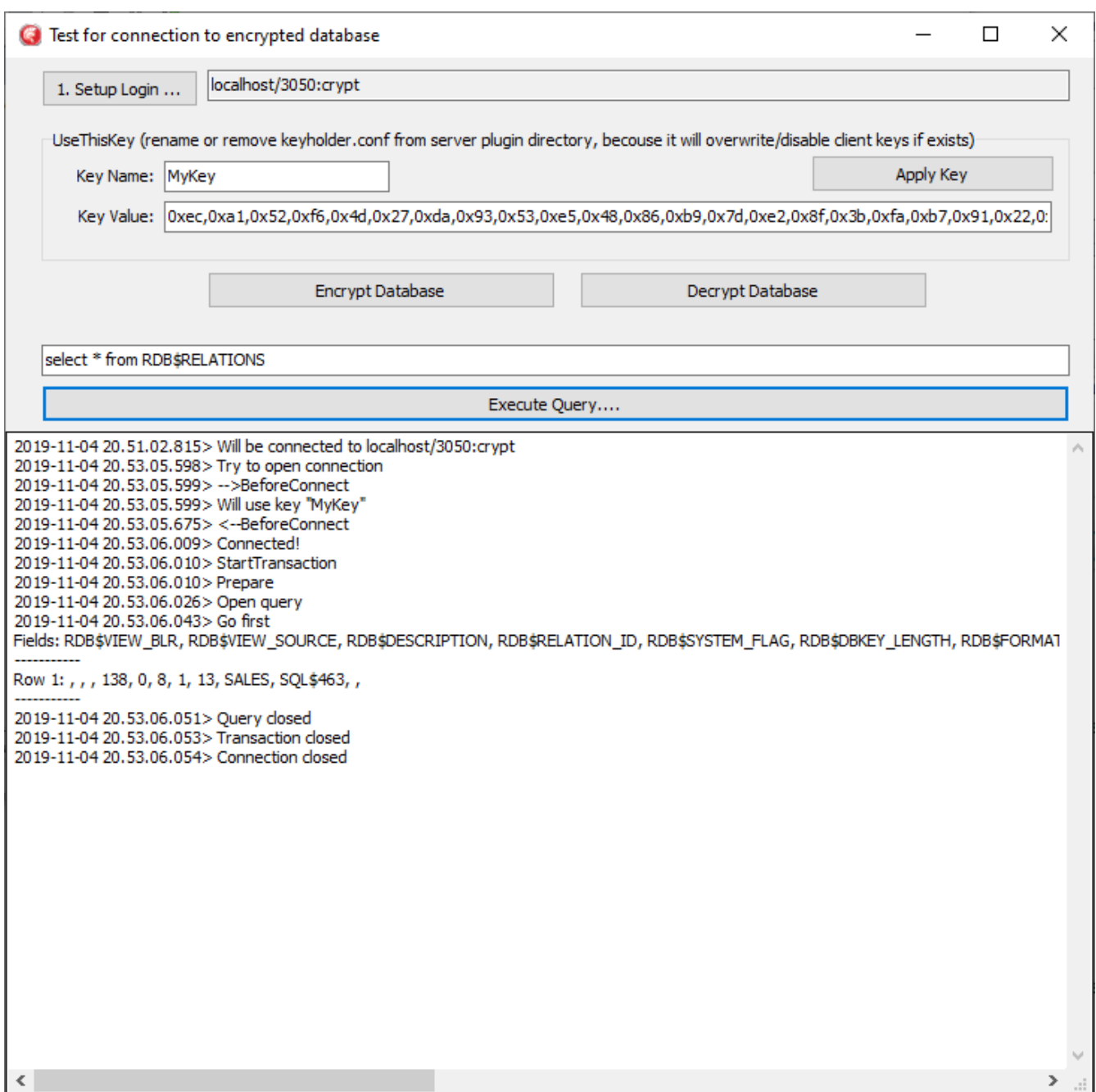
1. Specify database path or alias in “1. Setup Login”. This database will be used in the next steps.
2. Specify the key name and value to be used. If you have previously used key RED, set Key Name = RED and copy the key value from the *KeyHolder.conf* file.
3. You can encrypt and decrypt database with the specified key. Please note — encryption takes time, and it requires to have an active connection to the database
4. Click Execute query to test the connection to the encrypted database with the simple



Please note!

The test application can connect to the encrypted database only through TCP/IP, xnet is not supported.

In the example of the client application, all database operations (connection, transaction start, transaction's commit, query start, etc) are made in the very straightforward way to demonstrate all steps of the operation against the encrypted database. You can use this code as an example for the implementing encryption in your applications.



Step 3: backup and restore of the encrypted database

The full verified backup with *gbak.exe* is the primary backup method for Firebird databases. The standard Firebird distribution includes command line tool *gbak.exe* to perform it, however, it will not work with the encrypted database in the production mode (without keys on the server). After the encryption, only authorized applications can access an encrypted database, and standard *gbak* is not an authorized application.

We all know how important backup and restore for the database health and performance, so, in order to perform backup and restore for the encrypted databases, we have developed *gbak.exe* with the encryption support, and included it into the FEPE.

It is important to say, that this *gbak.exe* produces the encrypted backup file: it encrypts the backup with the same key as for the database encryption.

If you run *gbak.exe* from the plugin files with the switch *-?*, you will see the new parameters of

`gbak.exe`, which are used to work with the encrypted databases:

```
-KEYFILE      name of a file with DB and backup crypt key(s)
-KEYNAME      name of a key to be used for encryption
-KEY          key value in "0x5A," notation
```

Let's consider how to use `gbak.exe` with encrypted databases and backups.

Backup encrypted Firebird database

To backup an encrypted Firebird database, `gbak.exe` must provide the key for the server. This key will be used to connect and read the database and to encrypt the backup file.

There are 2 ways to supply the key for `gbak.exe`: store key in the key file or explicitly put it in the command line:

Example 1. Example of backup with the encryption key in the key file

```
gbak.exe -b -KEYFILE h:\Firebird\Firebird-3.0.3.32900-0_Win32\examplekeyfile.txt
  -KEYNAME RED localhost:h:\employee_30.fdb h:\testenc4.fbk -user SYSDBA
  -pass masterkey
```

Here, in the parameter `-KEYFILE` we specify the location of the files with keys, and in `-KEYNAME` — the name of the key being used. Please note, that the file `examplekeyfile.txt` has the same structure as `KeyHolder.conf`.

If you apply backup with encryption (`gbak -b -keyfile ... -keyname ...`) on the unencrypted database, the backup will be encrypted.

Example 2. Example of backup with the explicit key

```
gbak -b -KEY 0xec,0xa1,0x52,0xf6,0x4d,0x27,0xda,0x93,0x53,0xe5,0x48,0x86,0xb9,
  0x7d,0xe2,0x8f,0x3b,0xfa,0xb7,0x91,0x22,0x5b,0x59,0x15,0x82,0x35,0xf5,0x30,
  0x1f,0x04,0xdc,0x75, -keyname RED localhost:h:\employee30\employee30.fdb
  h:\testenc303.fbk -user SYSDBA -pass masterkey
```

Here, we specify the key value in the parameter `-KEY`, and the name of the key in the parameter `-KEYNAME`. It is necessary to specify key name even if we supply the explicit key value.

Restore the backup to the encrypted Firebird database

The `gbak` can also restore from the backup files to the encrypted databases. The approach is the same: we need to provide the key name and key value to restore the backup file.

See below examples of the restore commands:

Example of restore with the encryption key in the keyfile

```
gbak -c -v -keyfile h:\Firebird\Firebird-3.0.3.32900-0_Win32\examplekeyfile.txt
-keyname white h:\testenc4.fbk localhost:h:\employeeenc4.fdb -user SYSDBA
-pass masterkey
```

Example 3. Example of restore with the explicit key

```
gbak -c -v -key 0xec,0xa1,0x52,0xf6,0x4d,0x27,0xda,0x93,0x53,0xe5,0x48,0x86,
0xb9,0x7d,0xe2,0x8f,0x3b,0xfa,0xb7,0x91,0x22,0x5b,0x59,0x15,0x82,0x35,0xf5,
0x30,0x1f,0x04,0xdc,0x75, -keyname RED h:\testenc4.fbk
localhost:h:\employeeenc4.fdb -user SYSDBA -pass masterkey
```

If you restore from an unencrypted backup file with encryption keys (`gbak -c -keyfile ... -keyname ...`), the restored database will be encrypted.

Chapter 9. Authentication plugin for Execute Statement On External

Firebird has a convenient mechanism of cross-database queries: Execute Statement On External (ESOE). For example, the typical ESOE can look like this:

```
EXECUTE STATEMENT 'SELECT * FROM RDB$DATABASE'
ON EXTERNAL 'server:db1' AS USER 'MYUSER' PASSWORD 'mypassword'
```

As you can see, the statement contains username and password in the open form, which is not secure: for example, if ESOE is called from the stored procedure code, connected users can see the password.

The HQbird Enterprise includes an authentication plugin for ESOE allows to establish trusted relationships between Firebird servers and perform the authentication of ESOE without a password:

```
EXECUTE STATEMENT 'SELECT * FROM RDB$DATABASE'
ON EXTERNAL 'server:db1' AS USER 'MYUSER';
```

Let's consider how to install and configure HQbird Authentication plugin for ESOE.

9.1. Installation of authentication plugin for ESOE

9.1.1. Authentication plugin files

- Windows
 - *plugins\cluster.dll*
 - *clusterkeygen.exe*
- Linux
 - *plugins\libcluster.so*
 - *bin\ClusterKeygen* # executable



Please note!

The plugin files are already included into HQbird Enterprise, so you don't need to copy them. There are different files for 32 and 64 bit versions of Firebird.

9.1.2. Configuration

In *firebird.conf*

First of all, it is necessary to add plugin name (Cluster) to the AuthServer and AuthClient parameters

in *firebird.conf* on all servers which will trust each other:

```
AuthServer = Srp, Legacy_Auth, Cluster
AuthClient = Srp, Srp256, Legacy_Auth, Cluster
```

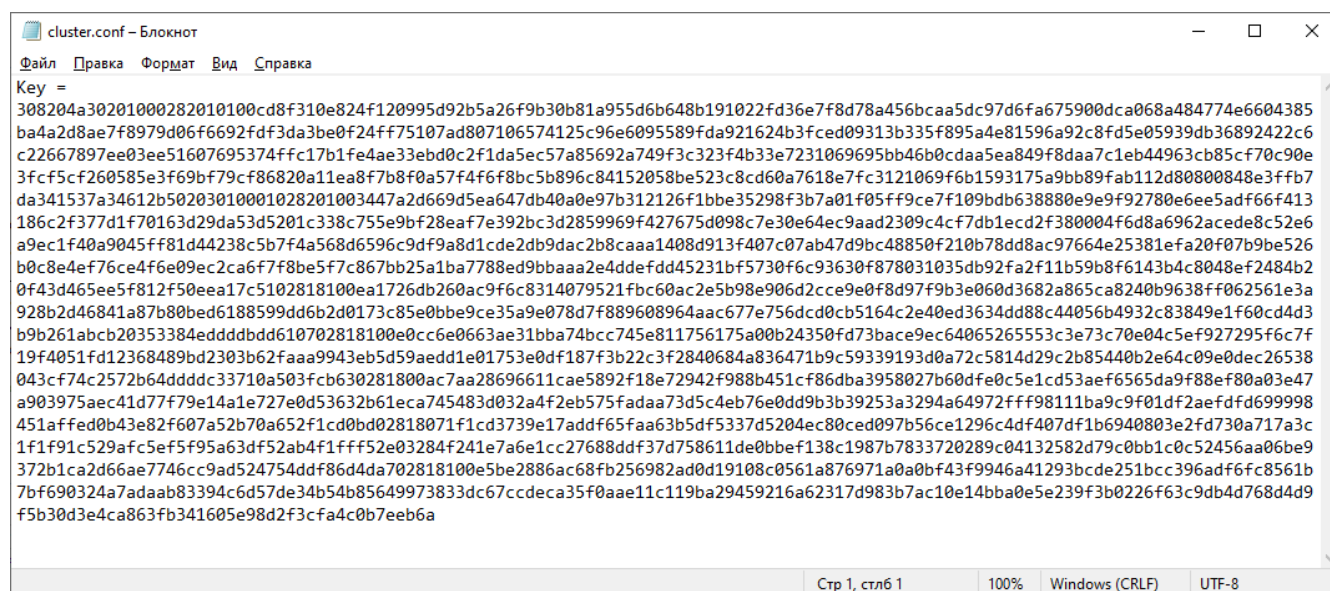
Keyfile

Then, it is necessary to generate keyfile for the plugin. This key should be placed to the all Firebird servers which should trust to each other.

In order to generate keyfile *cluster.conf* run the following command:

```
C:\HQBird\Firebird30>clusterkeygen.exe > cluster.conf
```

As a result, the key file *cluster.conf* will be generated. It contains 2048 digits key, something like this:



```
cluster.conf - Блокнот
Файл  Правка  Формат  Вид  Справка
Key =
308204a30201000282010100cd8f310e824f120995d92b5a26f9b30b81a955d6b648b191022fd36e7f8d78a456bcaa5dc97d6fa675900dca068a484774e6604385
ba4a2d8ae7f8979d06f6692fd3da3be0f24ff75107ad807106574125c96e6095589fda921624b3fced09313b335f895a4e81596a92c8fd5e05939db36892422c6
c22667897ee03ee51607695374ffc17b1fe4ae33ebd0c2f1da5ec57a85692a749f3c323f4b33e7231069695bb46b0cdaa5ea849f8daa7c1eb44963cb85cf70c90e
3fcf5c260585e3f69bf79cf86820a11ea8f7b8f0a57f46f8bc5b896c84152058be523c8cd60a7618e7fc3121069f6b1593175a9bb89fab112d80800848e3ffb7
da341537a34612b50203010001028201003447a2d669d5ea647db40a0e97b312126f1bbe35298f3b7a01f05ff9ce7f109bdb638880e9e9f92780e6ee5adf66f413
186c2f377d1f70163d29da53d5201c338c755e9bf28eaf7e392bc3d2859969f427675d098c7e30e64ec9aad2309c4cf7db1ecd2f380004f6d8a6962accede8c52e6
a9ec1f40a90455f81d44238c5b7f4a568d6596c9df9a8d1cde2db9dac2b8caaa1408d913f407c07ab47d9bc48850f210b78dd8ac97664e25381efa20f07b9be526
b0c8e4ef76ce4f6e09ec2ca6f7f8be5f7c867bb25a1ba7788ed9bbaaa2e4ddeffd45231bf5730f6c93630f878031035db92fa2f11b59b8f6143b4c8048ef2484b2
0f43d465ae5f812f50eea17c5102818100ea1726db260ac9f6c8314079521fbc60ac2e5b98e906d2cce9e0f8d97f9b3e060d3682a865ca8240b9638ff062561e3a
928b2d46841a87b08bed6188599dd6b2d0173c85e0bbe9ce35a9e078d7f889608964aac677e756dcd0cb5164c2e40ed3634dd88c44056b4932c3849e1f60cd4d3
b9b261abcb20353384eddddbdd610702818100e0cc6e0663ae31bba74bcc745e811756175a00b24350fd73bace9ec64065265553c3e73c70e04c5ef927295f6c7f
19f4051fd12368489bd2303b62faaa9943eb5d59aedd1e01753e0df187f3b22c3f2840684a836471b9c5939193d0a72c5814d29c2b85440b2e64c09e0dec26538
043cf74c2572b64d4ddc33710a503fcb630281800ac7aa28696611cae5892f18e72942f988b451cf86dba3958027b60df0c5e1cd53aef6565da9f88ef80a03e47
a903975aec41d77f79e14a1e727e0d53632b61eca745483d0324f2eb575fadaa73d5c4eb76e0dd9b3b39253a3294a64972ffff98111ba9c9f01df2aefdf699998
451affed0b43e82f607a52b70a652f1cd0bd02818071f1cd3739e17addf65faa63b5df5337d5204ec80ced097b56ce1296c4df407df1b6940803e2fd730a717a3c
1f1f91c529af5ef5f95a63df52ab4f1fff52e03284f241e7a6e1cc27688dd37d758611de0bbeff138c1987b7833720289c04132582d79c0bb1c0c52456aa06be9
372b1ca2d66ae7746cc9ad524754ddf86d4da702818100e5be2886ac68fb256982ad0d19108c0561a876971a0a0bf43f9946a41293bcde251bcc396ad6f6c8561b
7bf690324a7adaab83394c6d57de34b54b85649973833dc67ccdeca35f0aae11c119ba2945921a6e2317d983b7ac10e14bba0e5e239f3b0226f63c9db4d768d4d9
f5b30d3e4ca863fb341605e98d2f3cfa4c0b7eeb6a
```

Then, we need to copy the key file to the all Firebird servers with trusted relationships to the *plugins* folder. Key, created on Windows, can be used on Linux, and vice versa.



Please note!

The keyfile name should be exactly *cluster.conf*. It should be located in *plugins* folder of Firebird.

Mapping

In order to use authentication plugin inside the particular database, it is necessary to create mapping between users of cluster plugin and regular Firebird users.

For example, if we run the following ESOE

```
EXECUTE STATEMENT 'SELECT * FROM RDB$DATABASE'
ON EXTERNAL 'server:db1' AS USER 'MYUSER';
```

we need to map user MYUSER to the actual user in the destination database db1. Let's assume we have a user MYUSER2 in the destination database, in this case we need to create the following command in the destination database db1:

```
SQL> CREATE MAPPING usr_mapping_cluster1 USING PLUGIN CLUSTER
CON> FROM USER MYUSER TO user MYUSER2;
SQL> commit;
```

As a result, the mapping `usr_mapping_cluster1` will be created in db1, to map user MYUSER to MYUSER2.

Please note!

The both users should exist, even if they have the same name. Otherwise there will be the following error:



Execute statement error at attach:

```
335544472: Your user name and password are not defined. Ask your
database
administrator to set up a Firebird login.
```

You can create as many mapping as you need. Existing mapping can be found in the table `RDB$AUTH_MAPPING` with the following query:

```
SQL> select rdb$map_name, rdb$map_from, rdb$map_to from RDB$AUTH_MAPPING
CON> where RDB$MAP_PLUGIN = 'CLUSTER';
```

RDB\$MAP_NAME	RDB\$MAP_FROM	RDB\$MAP_TO
USR_MAPPING_CLUSTER1	MYUSER	MYUSER2

Global mappings

It is possible to create mapping between users for all databases on the server—in this case the following command should be used:

```
CREATE GLOBAL MAPPING global_usr_mapping_cluster1 USING PLUGIN CLUSTER
FROM USER MYUSER TO user MYUSER2;
```

In this case, mappings will be stored in security database, to see them use the following query:

```
SQL> select SEC$MAP_NAME, SEC$MAP_USING, SEC$MAP_FROM, SEC$MAP_TO
CON> from SEC$GLOBAL_AUTH_MAPPING where SEC$MAP_PLUGIN ='CLUSTER';
```

SEC\$MAP_NAME	SEC\$MAP_USING	SEC\$MAP_FROM	SEC\$MAP_TO
GLOBAL_USR_MAPPING_CLUSTER1	P	MYUSER	MYUSER2

Role mappings

In order to map user to the role in the destination database, it is necessary to create 2 mappings:

```
CREATE MAPPING USR_CLUSTER9 USING PLUGIN CLUSTER
FROM USER MUSER TO ROLE RDB$ADMIN;
```

```
CREATE MAPPING USR_CLUSTER_X USING PLUGIN CLUSTER
FROM ANY USER TO USER MYUSER;
```

9.1.3. How to test

The following query can be used to test the work of the authentication plugin for ESOE:

```
execute block
returns (
  CUSER varchar(255),
  CCONNECT bigint,
  CROLE varchar(31))
as
begin
  execute statement
    'select CURRENT_USER, CURRENT_CONNECTION, CURRENT_ROLE FROM RDB$DATABASE'
  on external 'server:db1'
  into :CUSER, :CCONNECT, :CROLE;
suspend;
end
```

As a result, this query will return username, connection id and user role from the destination database db1.

Chapter 10. RSA-UDR — security functions to sign documents and verify signatures

HQbird Enterprise includes RSA-UDR which contains the set of useful security functions. These functions can be useful to protect documents (stored as varchars, BLOBS and even external files) with digital signature.



Important

This RSA-UDR is not required if using Firebird 4.0, as these functions are built-in.



Please note!

To use RSA-UDR functions, you need to register them in your database with an appropriate SQL script. The script is available in *plugin/UDR/crypto.sql*

Let's consider functions from this library and examples their usage.

Function Name	Description
BIN2HEX	Convert binary representation to HEX
BIN2HEXB	Convert BLOB binary representation to HEX
CRC32	Calculate checksum
CRC32B	Calculate BLOB checksum
DECODE_BASE64	Decode to Base64
DECODE_BASE64B	Decode BLOB to Base64
ENCODE_BASE64	Encode to Base64
ENCODE_BASE64B	Encode BLOB to Base64
HEX2BIN	Convert HEX to binary
HEX2BINB	Convert NEX BLOB to binary
MD5	Calculate MD5 hash sum
MD5B	Calculate BLOB MD5 hash sum
RSA_PUBLIC_KEY	Generate public key
RSA_PRIVATE_KEY	Generate private key
RSA_SIGN	Sign object
RSA_VERIFY	Verify the signature
SHA1	Calculate SHA
SHA1B	Calculate SHA for BLOB
SHA256	Calculate SHA256
SHA256B	Calculate SHA256 for BLOB

Let's consider the simplified example how to use these functions to sign some document and verify the signature.

For simplicity, we will put the document, private key, public key, digest (hash sum of the document) and signature will be in the same table, in the single column:

```
SQL> show table TBL;

DOC BLOB segment 80, subtype BINARY Nullable
DIGEST VARCHAR(32) CHARACTER SET OCTETS Nullable
SALTLEN INTEGER Nullable
PRIVATE_KEY VARCHAR(2048) CHARACTER SET OCTETS Nullable
SIGN VARCHAR(1024) CHARACTER SET OCTETS Nullable
PUBLIC_KEY VARCHAR(512) CHARACTER SET OCTETS Nullable
BAD_SIGN VARCHAR(1024) CHARACTER SET OCTETS Nullable
```

Then, let's go through the example:

```
-- First, we will connect to the database:

C:\HQbird\Firebird30>isql localhost:c:\temp\rsatest.fdb -user SYSDBA -pass masterkey
Database: localhost:c:\temp\rsatest.fdb, User: SYSDBA

-- and then we will check that functions are registered
SQL> show functions;
Global functions

Function Name Invalid Dependency, Type
=====
RSA_PRIVATE_KEY
RSA_PUBLIC_KEY
RSA_SIGN
RSA_VERIFY
SHA256

-- clean the test table
SQL>delete from tbl
SQL>commit;

-- generate private key and write
-- it into table TBL (normally, the private will be kept in the - secret place)

SQL>insert into tbl(PRIVATE_KEY) values(rsa_private_key(1024));

-- generate public key
SQL>update tbl set PUBLIC_KEY = rsa_public_key(PRIVATE_KEY);

-- create BLOB document
```

```

SQL>update TBL set DOC='testtesttest';

-- and calculate its digest
SQL>update tbl set digest = sha256(doc);

-- sign document and remember its signature
SQL>update tbl set sign = rsa_sign(digest, PRIVATE_KEY, 8);

-- check the signature
SQL> select RSA_VERIFY(SIGN, DIGEST, PUBLIC_KEY, SALTLEN) from tbl;

RSA_VERIFY
=====
<true>
-- as you can see, signature is valid

-- change the document (BLOB)
SQL> update TBL set DOC='testtesttest222';

-- recalculate its digest
SQL> update tbl set digest = sha256(doc);

-- check signature
SQL> select rsa_verify(sign, digest, PUBLIC_KEY, 8) from tbl;

RSA_VERIFY
=====
<false>
-- we can see that protected document was changed

```

Examples of BIN2HEX and HEX2BIN functions

```

SQL> set list;
SQL> select bin2hex('Test string') from rdb$database;

BIN2HEX 5465737420737472696E67

SQL> select cast (hex2bin('5465737420737472696E67') as varchar(32))
CON> from rdb$database;

CAST      Test string

```

10.1. How to use RSA-UDR security and conversion functions

In general, RSA-UDR functions allow to seal electronic documents of all types (DOC, PDF, XML, JPG, PNG, etc), and then detect unauthorized changes.

Conversion functions make easy BIN → HEX and HEX → BIN conversions, as well as Base64 encoding and decoding.

Chapter 11. SPLIT-UDR — procedures to splitting lines by separator

HQbird Enterprise includes SPLIT-UDR which contains the set of useful stored procedures. These procedures can be useful to splitting string (stored as varchars or BLOBS) by separator.



Please note!

To use SPLIT-UDR functions, you need to register them in your database with an appropriate SQL script. The script is available in *plugin/UDR/split-udr.sql*

For convenience, procedures for splitting text by separator are packed in the SPLIT_UTILS package.

Let's consider procedures from this library and examples their usage.

Procedure Name	Description
SPLIT_BOOLEAN	Splits the string at the delimiter and returns a set of BOOLEAN values.
SPLIT_SMALLINT	Splits the string at the delimiter and returns a set of SMALLINT values.
SPLIT_INT	Splits the string at the delimiter and returns a set of INTEGER values.
SPLIT_BIGINT	Splits the string at the delimiter and returns a set of BIGINT values.
SPLIT_STR	Splits the string at the delimiter and returns a set of VARCHAR(8191) values.
SPLIT_DATE	Splits the string at the delimiter and returns a set of DATE values.
SPLIT_TIME	Splits the string at the delimiter and returns a set of TIME values.
SPLIT_TIMESTAMP	Splits the string at the delimiter and returns a set of TIMESTAMP values.
SPLIT_DOUBLE	Splits the string at the delimiter and returns a set of DOUBLE PRECISION values.

The first argument of these procedures is a BLOB with subtype TEXT, the second is a string of type VARCHAR (10). The output type depends on the name of the procedure.

Let's look at simple examples of how to use these procedures.


```
SQL> select cast(out_str as varchar(10)) from SPLIT_UTILS.split_str('abc##defg##aa',
'##');
```

```
CAST
```

```
=====
```

```
abc
defg
aa
```

```
SQL> select out_int from SPLIT_UTILS.split_int('1,2,3,4,5', ',');
```

```
OUT_INT
```

```
=====
```

```
1
2
3
4
5
```

In addition, SplitUDR also contains procedures for splitting text into tokens (the text is split into several separators). These procedures are declared as follows:

```
CREATE OR ALTER PROCEDURE SPLIT_WORDS (
  IN_TXT          BLOB SUB_TYPE TEXT CHARACTER SET UTF8,
  IN_SEPARATORS  VARCHAR(50) CHARACTER SET UTF8 DEFAULT NULL)
RETURNS (
  WORD VARCHAR(8191) CHARACTER SET UTF8)
EXTERNAL NAME 'splitudr!strtok' ENGINE UDR;
```

```
CREATE OR ALTER PROCEDURE SPLIT_WORDS_S (
  IN_TXT          VARCHAR(8191) CHARACTER SET UTF8,
  IN_SEPARATORS  VARCHAR(50) CHARACTER SET UTF8 DEFAULT NULL)
RETURNS (
  WORD VARCHAR(8191) CHARACTER SET UTF8)
EXTERNAL NAME 'splitudr!strtok_s' ENGINE UDR;
```

Input parameters:

- IN_TXT - input text of type BLOB SUB_TYPE TEXT or VARCHAR (8191)
- IN_SEPARATORS - a list of separators (a string with separator symbols), if not specified, then separators are used " \n\r\t,?!:;\/<>[]{}@#\$\$%^&*~+=``~`"

Example:

```
SELECT  
  w.WORD  
FROM DOCS  
LEFT JOIN SPLIT_WORDS(DOCS.CONTENT) w  
WHERE DOCS.DOC_ID = 4
```

Chapter 12. OCR-UDR — function to recognizing text from images

HQbird Enterprise includes OCR-UDR which contains function to recognizing text from images.

UDR OCR is based on the free text recognition library Tesseract OCR 4.1, released under the Apache License, Version 2.0.

To register ocr-udr, you need to execute the following script:

```
CREATE OR ALTER FUNCTION GET_OCR_TEXT (
    PIX_DATA BLOB SUB_TYPE BINARY,
    PPI      SMALLINT = NULL)
RETURNS BLOB SUB_TYPE TEXT
EXTERNAL NAME 'ocrudr!getOcrText!eng' ENGINE UDR;

COMMENT ON FUNCTION GET_OCR_TEXT IS
'Recognizing text from images';

COMMENT ON PARAMETER GET_OCR_TEXT.PIX_DATA IS
'Image';

COMMENT ON PARAMETER GET_OCR_TEXT.PPI IS
'Pix Per Inch. Some image formats do not store this information. For scanned
pages are usually 200-300 ppi. The minimum value is 70.';
```

Pay attention to the EXTERNAL NAME in it after the module name and entry point with "!" the name of the dictionary is indicated.



Where is eng a dictionary from tessdata catalog, which is used for recognition. Several dictionaries can be listed here, for example rus+eng. If absent, then the dictionary eng is used.

The set contains a limited number of dictionaries. A complete list of dictionaries is available at: https://github.com/tesseract-ocr/tessdata_best (high recognition quality) or https://github.com/tesseract-ocr/tessdata_fast (high recognition speed).

The library supports text recognition from images in PNG, JPEG, GIF formats.

12.1. Example of using OCR-UDR

Suppose you have a DOCS table that stores scans of documents, defined as follows:

```
CREATE TABLE DOCS (  
  ID          BIGINT GENERATED BY DEFAULT AS IDENTITY,  
  NAME       VARCHAR(127) NOT NULL,  
  PIC        BLOB SUB_TYPE BINARY,  
  TXT        BLOB SUB_TYPE TEXT,  
  PROCESSED  BOOLEAN default FALSE NOT NULL,  
  CONSTRAINT PK_DOCS PRIMARY KEY (ID)  
);
```

Images for recognition are saved in the PIC field, the recognized text in the TXT field.

To recognize text from a single image, you can use the query:

```
SELECT GET_OCR_TEXT(pic, 200) as txt  
FROM docs  
WHERE docs.id = 2;
```

The next query recognizes text from images and saves it to the TXT field.

```
update docs  
UPDATE DOCS  
SET TXT = GET_OCR_TEXT(PIC, 200),  
    PROCESSED = TRUE  
WHERE PROCESSED IS FALSE;
```

Chapter 13. LK-JSON-UDR — building and parsing JSON

HQbird Enterprise includes UDR library `udr-lkJSON` for building and parsing JSON on the server side. The library is distributed open source under the MIT license and is free to use. It is written in Free Pascal. Source code is available at <https://github.com/mnf71/udr-lkJSON>

13.1. Install UDR lkJSON

To use the UDR `lkJSON` library, you need to register it in your database. To do this, run one of the scripts `plugin/UDR/udrJSON.sql` or `plugin/UDR/udrJSON-utf8.sql`, depending on the encoding of your database (the first will work for any single-byte encoding).

After installing the UDR, it can be verified using the <https://github.com/mnf71/udr-lkJSON/blob/main/verify.sql> script.

The script calls a function to parse the JSON and build it back into a string. If the original JSON is the same as the newly assembled JSON, then everything is in order. In reality, the strings will not completely match, since the JSON is assembled without taking into account the beautiful formatting. But the content must be identical.

Verification occurs for two sets (procedure + function):

- `js$func.ParseText` — parsing JSON given as BLOB. `js$func.GenerateText` — JSON assembly with BLOB return.
- `js$func.ParseString` — parsing JSON given as VARCHAR(N). `js$func.GenerateString` — JSON assembly returning VARCHAR(N).

13.2. How it works?

The `udr-lkJSON` library is based on the free `lkJSON` library for generating and parsing JSON. Therefore, in order to have a good idea of how to work with UDR-`lkJSON`, it is advisable to familiarize yourself with the `lkjson` library (see <https://sourceforge.net/projects/lkjson/>).

When parsing JSON, some elements can be simple types that exist in Firebird (INTEGER, DOUBLE PRECISION, VARCHAR (N), BOOLEAN), and some complex ones are objects and arrays. Complex objects are returned as a pointer to an internal object from the `lkJSON` library. The pointer maps to the `TY$POINTER` domain. This domain is defined as follows:

```
CREATE DOMAIN TY$POINTER AS
CHAR(8) CHARACTER SET OCTETS;
```

In addition, if NULL is encountered in JSON, then it will not be returned to simple types! You will have to recognize this value separately. This is because the UDR-`lkJSON` library simply copies the methods of the `lkJSON` library classes into PSQL packages. And as you know, simple types in Pascal do not have a separate state for NULL.

13.3. Description of PSQL packages from UDR-lkJSON

13.3.1. JS\$BASE package

The header of this package looks like this:

```
CREATE OR ALTER PACKAGE JS$BASE
AS
BEGIN
  /* TlkJSONbase = class
  TlkJSONtypes =
  (jsBase, jsNumber, jsString, jsBoolean, jsNull, jsList, jsObject);
  0      1      2      3      4      5      6
  */
  FUNCTION Dispose(Self TY$POINTER) RETURNS SMALLINT; /* 0 - succes */

  FUNCTION Field(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE /* = Idx */)
  RETURNS TY$POINTER;

  FUNCTION Count_(Self TY$POINTER) RETURNS INTEGER;
  FUNCTION Child(Self TY$POINTER, Idx INTEGER, Obj TY$POINTER = NULL /* Get */)
  RETURNS TY$POINTER;

  FUNCTION Value_(Self TY$POINTER, Val VARCHAR(32765) CHARACTER SET NONE = NULL /* Get
  */) RETURNS VARCHAR(32765) CHARACTER SET NONE;
  FUNCTION WideValue_(Self TY$POINTER, WVal BLOB SUB_TYPE TEXT = NULL /* Get */)
  RETURNS BLOB SUB_TYPE TEXT;

  FUNCTION SelfType(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
  SMALLINT;
  FUNCTION SelfTypeName(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
  VARCHAR(32) CHARACTER SET NONE;
END
```

As you can see from the comment, this package is a blueprint for the TlkJSONbase class. It contains basic functionality for working with JSON.

The Dispose function is designed to release a pointer to a JSON object. Pointers to be forcibly freed are the result of parsing or JSON generation. You should not call it on intermediate objects when parsing or assembling JSON. It is only required for the top-level object.

The Field function returns a pointer to an object field. The first parameter is a pointer to the object, the second is the field name. If the field does not exist, then the function will return a null pointer (This is not NULL, but x'0000000000000000').

The Count_ function returns the number of items in a list or fields in an object. A pointer to an object or a list is specified as a parameter.

The Child function returns or sets the value for the element at index Idx in the Self object or list. If

the `Obj` parameter is not specified, then it returns a pointer to the element from the `Idx` indices. If `Obj` is specified, then sets its value to the element with indices `Idx`. Note `Obj` is a pointer to one of the `TlkJSONbase` descendants.

The `Value_` function returns or sets in the form of a JSON string (`VARCHAR`) the value for the object specified in the `Self` parameter. If the `Val` parameter is not specified, then the value is returned; otherwise, it is set.

The `WideValue_` function returns or sets as a JSON string (`BLOB SUB_TYPE TEXT`) the value for the object specified in the `Self` parameter. If the `Val` parameter is not specified, then the value is returned; otherwise, it is set.

The `SelfType` function returns the type of the object for the pointer specified in the `Self` parameter. The object type is returned as a number, where

- 0 — `jsBase`
- 1 — `jsNumber`
- 2 — `jsString`
- 3 — `jsBoolean`
- 4 — `jsNull`
- 5 — `jsList`
- 6 — `jsObject`

If the `Self` parameter is not specified, it will return 0.

The `SelfTypeName` function returns the object type for the pointer specified in the `Self` parameter. The object type is returned as a string. If the `Self` parameter is not specified, it will return `'jsBase'`.

13.3.2. JS\$BOOL package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE JS$BOOL
AS
BEGIN
  /* TlkJSONbase = class
     TlkJSONboolean = class(TlkJSONbase)
  */
  FUNCTION Value_(Self TY$POINTER, Bool BOOLEAN = NULL /* Get */) RETURNS BOOLEAN;

  FUNCTION Generate(Self TY$POINTER = NULL /* NULL - class function */, Bool BOOLEAN =
TRUE) RETURNS TY$POINTER;

  FUNCTION SelfType(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
SMALLINT;
  FUNCTION SelfTypeName(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
VARCHAR(32) CHARACTER SET NONE;
END

```

As you can see from the comment, this package is a blueprint for the `TlkJSONboolean` class. It is designed to work with the `BOOLEAN` datatype.

The `Value_` function returns or sets to a boolean value for the object specified in the `Self` parameter. If the `Bool` parameter is not specified, then the value will be returned, if specified — set. Note that `NULL` is not returned and cannot be set by this method, there is a separate `JS$NULL` package for this.

The `Generate` function returns a pointer to a new `TlkJSONboolean` object, which is a Boolean value in JSON. The `Self` parameter is a pointer to the JSON object on the basis of which the `TlkJSONboolean` object is created. The boolean value is specified in the `Bool` parameter.

The `SelfType` function returns the type of the object for the pointer specified in the `Self` parameter. The object type is returned as a number. If the `Self` parameter is not specified, it will return 3.

The `SelfTypeName` function returns the object type for the pointer specified in the `Self` parameter. The object type is returned as a string. If the `Self` parameter is not specified, it will return `'jsBoolean'`.

13.3.3. JS\$CUSTLIST package

The header of this package looks like this:


```

CREATE OR ALTER PACKAGE JS$CUSTLIST
AS
BEGIN
  /* TlkJSONbase = class
     TlkJSONcustomlist = class(TlkJSONbase)
  */
  PROCEDURE ForEach
    (Self TY$POINTER) RETURNS (Idx Integer, Name VARCHAR(128) CHARACTER SET NONE, Obj
TY$POINTER /* js$Base */);

  FUNCTION Field(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE /* = Idx */)
RETURNS TY$POINTER;
  FUNCTION Count_(Self TY$POINTER) RETURNS INTEGER;
  FUNCTION Child(Self TY$POINTER, Idx INTEGER, Obj TY$POINTER = NULL /* Get */)
RETURNS TY$POINTER;

  FUNCTION GetBoolean(Self TY$POINTER, Idx INTEGER) RETURNS BOOLEAN;
  FUNCTION GetDouble(Self TY$POINTER, Idx INTEGER) RETURNS DOUBLE PRECISION;
  FUNCTION GetInteger(Self TY$POINTER, Idx INTEGER) RETURNS INTEGER;
  FUNCTION GetString(Self TY$POINTER, Idx INTEGER) RETURNS VARCHAR(32765) CHARACTER
SET NONE;
  FUNCTION GetWideString(Self TY$POINTER, Idx INTEGER) RETURNS BLOB SUB_TYPE TEXT;
END

```

As you can see from the comment, this package is a blueprint for the `TlkJSONcustomlist` class. This type is basic when working with objects and lists. All procedures and functions of this package can be used as JSON of the object type, and JSON of the list type.

The `ForEach` procedure retrieves each list item or each object field from the JSON pointer specified in `Self`. The following values are returned:

- `Idx` — the index of the list item or the number of the field in the object. Starts at 0.
- `Name` — the name of the next field, if `Self` is an object. Or the index of the list item, starting at 0, if `Self` is a list.
- `Obj` is a pointer to the next element of the list or object field.

The `Field` function returns a pointer to a field by its name from the object specified in `Self`. Instead of a field name, you can specify the item number in the list or the field number. Numbering starts from 0.

The `Count_` function returns the number of items in a list or fields in an object specified in the `Self` parameter.

The `Child` function returns or sets the value for the element at index `Idx` in the `Self` object or list. Indexing starts from 0. If the `Obj` parameter is not specified, then it returns a pointer to the element from the `Idx` indices. If `Obj` is specified, then sets its value to the element with indices `Idx`. Note `Obj` is a pointer to one of the `TlkJSONbase` descendants.

The `GetBoolean` function returns the boolean value of an object field or array element with index

Idx. Indexing starts at 0.

The `GetDouble` function returns the floating point value of an object field or array element with index `Idx`. Indexing starts at 0.

The `GetInteger` function returns the integer value of an object field or array element with index `Idx`. Indexing starts at 0.

The `GetString` function returns the character value (`VARCHAR`) of an object field or array element with index `Idx`. Indexing starts at 0.

The `GetWideString` function returns the `BLOB SUB_TYPE TEXT` of an object field or array element with index `Idx`. Indexing starts at 0.



The functions `GetBoolean`, `GetDouble`, `GetInteger`, `GetString`, `GetWideString` cannot return `NULL`. There is a separate set of functions for handling `NULL` values in the `JS$NULL` package.

13.3.4. JS\$FUNC package

The header of this package looks like this:

```
CREATE OR ALTER PACKAGE JS$FUNC
AS
BEGIN
    FUNCTION ParseText(Text BLOB SUB_TYPE TEXT, Conv BOOLEAN = FALSE) RETURNS TY$
    POINTER;
    FUNCTION ParseString(String VARCHAR(32765) CHARACTER SET NONE, Conv BOOLEAN = FALSE)
    RETURNS TY$POINTER;

    FUNCTION GenerateText(Obj TY$POINTER, Conv BOOLEAN = FALSE) RETURNS BLOB SUB_TYPE
    TEXT;
    FUNCTION GenerateString(Obj TY$POINTER, Conv BOOLEAN = FALSE) RETURNS VARCHAR(32765)
    CHARACTER SET NONE;

    FUNCTION ReadableText(Obj TY$POINTER, Level INTEGER = 0, Conv BOOLEAN = FALSE)
    RETURNS BLOB SUB_TYPE TEXT;
END
```

This package contains a set of functions for parsing JSON or converting JSON to string.

The `ParseText` function parses JSON specified as a string of `BLOB SUB_TYPE TEXT` type in the `Text` parameter. If you pass `TRUE` in the `Conv` parameter, then the JSON text of the string will be converted from UTF8 encoding to general.

The `ParseString` function parses the JSON specified as a `VARCHAR (N)` string in the `String` parameter. If you pass `TRUE` in the `Conv` parameter, then the JSON text of the string will be converted from UTF8 encoding to general.

The `GenerateText` function returns JSON as a `BLOB SUB_TYPE TEXT` string. If `TRUE` is passed in the `Conv` parameter, then the text returned by this function will be converted to UTF8.

The `GenerateString` function returns JSON as a `VARCHAR (N)` string. If `TRUE` is passed in the `Conv` parameter, then the text returned by this function will be converted to UTF8.

The `ReadableText` function returns JSON as a human-readable string of type `BLOB SUB_TYPE TEXT`. The `Level` parameter sets the number of indents for the first level. This is required if the generated string is part of another JSON. If `TRUE` is passed in the `Conv` parameter, then the text returned by this function will be converted to UTF8.



Use of the `Conv` parameter set to `TRUE` is left for compatibility with the original `lkJSON` library. There is no special need for it, since external services independently know how to convert the source string into the format required for the DBMS and vice versa.

13.3.5. JS\$LIST package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE JS$LIST
AS
BEGIN
  /* TlkJSONbase = class
  TlkJSONcustomlist = class(TlkJSONbase)
  TlkJSONlist = class(TlkJSONcustomlist)
  */
  PROCEDURE ForEach
    (Self TY$POINTER) RETURNS (Idx Integer, Name VARCHAR(128) CHARACTER SET NONE, Obj
TY$POINTER /* js$Base */);

  FUNCTION Add_(Self TY$POINTER, Obj TY$POINTER) RETURNS INTEGER;
  FUNCTION AddBoolean(Self TY$POINTER, Bool BOOLEAN) RETURNS INTEGER;
  FUNCTION AddDouble(Self TY$POINTER, Db1 DOUBLE PRECISION) RETURNS INTEGER;
  FUNCTION AddInteger(Self TY$POINTER, Int_ INTEGER) RETURNS INTEGER;
  FUNCTION AddString(Self TY$POINTER, Str VARCHAR(32765) CHARACTER SET NONE) RETURNS
INTEGER;
  FUNCTION AddWideString(Self TY$POINTER, WStr BLOB SUB_TYPE TEXT) RETURNS INTEGER;

  FUNCTION Delete_(Self TY$POINTER, Idx Integer) RETURNS SMALLINT;
  FUNCTION IndexOfObject(Self TY$POINTER, Obj TY$POINTER) RETURNS INTEGER;
  FUNCTION Field(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE /* = Idx */)
RETURNS TY$POINTER;

  FUNCTION Count_(Self TY$POINTER) RETURNS INTEGER;
  FUNCTION Child(Self TY$POINTER, Idx INTEGER, Obj TY$POINTER = NULL /* Get */)
RETURNS TY$POINTER;

  FUNCTION Generate(Self TY$POINTER = NULL /* NULL - class function */) RETURNS TY
$POINTER;

  FUNCTION SelfType(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
SMALLINT;
  FUNCTION SelfTypeName(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
VARCHAR(32) CHARACTER SET NONE;
END

```

As you can see from the comment, this package is a blueprint for the `TlkJSONlist` class. It is designed to work with a list.

The `ForEach` procedure retrieves each list item or each object field from the JSON pointer specified in `Self`. The following values are returned:

- `Idx` — the index of the list item or the number of the field in the object. Starts at 0.
- `Name` — the name of the next field, if `Self` is an object. Or the index of the list item, starting at 0, if `Self` is a list.
- `Obj` is a pointer to the next element of the list or object field.

The `Add_` function adds a new item to the end of the list, the pointer to which is specified in the `Self`

parameter. The element to add is specified in the `Obj` parameter, which must be a pointer to one of the `TlkJSONbase` descendants. The function returns the index of the newly added element.

The `AddBoolean` function adds a new boolean element to the end of the list pointed to by the `Self` parameter. The function returns the index of the newly added element.

The `AddDouble` function adds a new element of real type to the end of the list, the pointer to which is specified in the `Self` parameter. The function returns the index of the newly added element.

The `AddInteger` function adds a new integer element to the end of the list pointed to by the `Self` parameter. The function returns the index of the newly added element.

The `AddString` function adds a new element of string type (`VARCHAR (N)`) to the end of the list pointed to by the `Self` parameter. The function returns the index of the newly added element.

The `AddWideString` function adds a new `BLOB SUB_TYPE TEXT` to the end of the list pointed to by the `Self` parameter. The function returns the index of the newly added element.

The `Delete_` function removes an element from the list with index `Idx`. The function returns 0.

The `IndexOfObject` function returns the index of an item in a list. The pointer to the list is specified in the `Self` parameter. The `Obj` parameter specifies a pointer to the element whose index is being defined.

The `Field` function returns a pointer to a field by its name from the object specified in `Self`. Instead of a field name, you can specify the item number in the list or the field number. Numbering starts from 0.

The `Count_` function returns the number of items in a list or fields in an object specified in the `Self` parameter.

The `Child` function returns or sets the value for the element at index `Idx` in the `Self` object or list. Indexing starts from 0. If the `Obj` parameter is not specified, then it returns a pointer to the element from the `Idx` indices. If `Obj` is specified, then sets its value to the element with indices `Idx`. Note `Obj` is a pointer to one of the `TlkJSONbase` descendants.

The `Generate` function returns a pointer to a new `TlkJSONlist` object, which is an empty list. The `Self` parameter is a pointer to the JSON object on the basis of which the `TlkJSONlist` is created.

The `SelfType` function returns the type of the object for the pointer specified in the `Self` parameter. The object type is returned as a number. If the `Self` parameter is not specified, it will return 5.

13.3.6. JS\$METH package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE JS$METH
AS
BEGIN
  /* TlkJSONbase = class
     TlkJSONobjectmethod = class(TlkJSONbase)
  */
  FUNCTION MethodObjValue(Self TY$POINTER) RETURNS TY$POINTER;
  FUNCTION MethodName(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE = NULL /*
Get */) RETURNS VARCHAR(128) CHARACTER SET NONE;
  FUNCTION MethodGenerate(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE, Obj
TY$POINTER /* js$Base */)
  RETURNS TY$POINTER /* js$Meth */;
END

```

As you can see from the comment, this package is a blueprint for the TlkJSONobjectmethod class. It is a key-value pair.

The MethodObjValue function returns a pointer to the value from the key-value pair specified in the Self parameter.

The MethodName function returns or sets the key name for the key-value pair specified in the Self parameter. If the Name parameter is not specified, then returns the name of the key, if specified, then sets the new name of the key.

The MethodGenerate function creates a new key-value pair and returns a pointer to it. The Name parameter specifies the name of the key, and the Obj parameter specifies a pointer to the key value.

13.3.7. JS\$NULL package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE JS$NULL
AS
BEGIN
  /* TlkJSONbase = class
     TlkJSONnull = class(TlkJSONbase)
  */
  FUNCTION Value_(Self TY$POINTER) RETURNS SMALLINT;

  FUNCTION Generate(Self TY$POINTER = NULL /* NULL - class function */) RETURNS TY
$POINTER;

  FUNCTION SelfType(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
SMALLINT;
  FUNCTION SelfTypeName(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
VARCHAR(32) CHARACTER SET NONE;
END

```

As you can see from the comment, this package is a blueprint for the `TlkJSONnull` class. It is designed to handle NULL values.

`Value_` returns 0 if the value of the object in `Self` is null (`jsNull`), and 1 otherwise.

The `Generate` function returns a pointer to a new `TlkJSONnull` object, which is null. The `Self` parameter is a pointer to the JSON object on the basis of which `TlkJSONnull` is created.

The `SelfType` function returns the type of the object for the pointer specified in the `Self` parameter. The object type is returned as a number. If the `Self` parameter is not specified, it will return 4.

The `SelfTypeName` function returns the object type for the pointer specified in the `Self` parameter. The object type is returned as a string. If the `Self` parameter is not specified, it will return `'jsNull'`.

13.3.8. JS\$NUM package

The header of this package looks like this:

```
CREATE OR ALTER PACKAGE JS$NUM
AS
BEGIN
  /* TlkJSONbase = class
     TlkJSONnumber = class(TlkJSONbase)
  */
  FUNCTION Value_(Self TY$POINTER, Num DOUBLE PRECISION = NULL /* Get */) RETURNS
DOUBLE PRECISION;

  FUNCTION Generate(Self TY$POINTER = NULL /* NULL - class function */, Num DOUBLE
PRECISION = 0) RETURNS TY$POINTER;

  FUNCTION SelfType(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
SMALLINT;
  FUNCTION SelfTypeName(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
VARCHAR(32) CHARACTER SET NONE;
END
```

As you can see from the comment, this package is a blueprint for the `TlkJSONnumber` class. It is designed to handle numeric values.

The `Value_` function returns or sets to a value of a numeric type for the object specified in the `Self` parameter. If the `Num` parameter is not specified, then the value will be returned, if specified — set. Note that NULL is not returned and cannot be set by this method, there is a separate `JS$NULL` package for this.

The `Generate` function returns a pointer to a `TlkJSONnumber` object, which is a JSON numeric value. The `Self` parameter is a pointer to the JSON object on the basis of which the `TlkJSONnumber` object is created. The `Num` parameter is a numeric value.

The `SelfType` function returns the type of the object for the pointer specified in the `Self` parameter. The object type is returned as a number. If the `Self` parameter is not specified, it will return 1.

The `SelfTypeName` function returns the object type for the pointer specified in the `Self` parameter. The object type is returned as a string. If the `Self` parameter is not specified, it will return `'jsNumber'`.

13.3.9. JS\$OBJ package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE JS$OBJ
AS
BEGIN
  /* TlkJSONbase = class
     TlkJSONcustomlist = class(TlkJSONbase)
     TlkJSONobject = class(TlkJSONcustomlist)
  */
  FUNCTION New_(UseHash BOOLEAN = TRUE) RETURNS TY$POINTER;
  FUNCTION Dispose(Self TY$POINTER) RETURNS SMALLINT; /* 0 - succes */

  PROCEDURE ForEach(Self TY$POINTER) RETURNS (Idx INTEGER, Name VARCHAR(128)
  CHARACTER SET NONE, Obj TY$POINTER /* js$Meth */);

  FUNCTION Add_(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE, Obj TY$POINTER)
  RETURNS INTEGER;
  FUNCTION AddBoolean(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE, Bool
  BOOLEAN) RETURNS INTEGER;
  FUNCTION AddDouble(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE, Db1 DOUBLE
  PRECISION) RETURNS INTEGER;
  FUNCTION AddInteger(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE, Int_
  INTEGER) RETURNS INTEGER;
  FUNCTION AddString(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE, Str
  VARCHAR(32765) CHARACTER SET NONE) RETURNS INTEGER;
  FUNCTION AddWideString(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE, WStr
  BLOB SUB_TYPE TEXT) RETURNS INTEGER;

  FUNCTION Delete_(Self TY$POINTER, Idx Integer) RETURNS SMALLINT;
  FUNCTION IndexOfName(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE) RETURNS
  INTEGER;
  FUNCTION IndexOfObject(Self TY$POINTER, Obj TY$POINTER) RETURNS INTEGER;
  FUNCTION Field(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE /* = Idx */,
  Obj TY$POINTER = NULL /* Get */) RETURNS TY$POINTER;

  FUNCTION Count_(Self TY$POINTER) RETURNS INTEGER;
  FUNCTION Child(Self TY$POINTER, Idx INTEGER, Obj TY$POINTER = NULL /* Get */)
  RETURNS TY$POINTER;

  FUNCTION Generate(Self TY$POINTER = NULL /* NULL - class function */, UseHash
  BOOLEAN = TRUE) RETURNS TY$POINTER;

  FUNCTION SelfType(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
  SMALLINT;

```



```

FUNCTION SelfTypeName(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
VARCHAR(32) CHARACTER SET NONE;

FUNCTION FieldByIndex(Self TY$POINTER, Idx INTEGER, Obj TY$POINTER = NULL /* Get */)
RETURNS TY$POINTER;
FUNCTION NameOf(Self TY$POINTER, Idx INTEGER) RETURNS VARCHAR(128) CHARACTER SET
NONE;

FUNCTION GetBoolean(Self TY$POINTER, Idx INTEGER) RETURNS BOOLEAN;
FUNCTION GetDouble(Self TY$POINTER, Idx INTEGER) RETURNS DOUBLE PRECISION;
FUNCTION GetInteger(Self TY$POINTER, Idx INTEGER) RETURNS INTEGER;
FUNCTION GetString(Self TY$POINTER, Idx INTEGER) RETURNS VARCHAR(32765) CHARACTER
SET NONE;
FUNCTION GetWideString(Self TY$POINTER, Idx INTEGER) RETURNS BLOB SUB_TYPE TEXT;

FUNCTION GetBooleanByName(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE)
RETURNS BOOLEAN;
FUNCTION GetDoubleByName(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE)
RETURNS DOUBLE PRECISION;
FUNCTION GetIntegerByName(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE)
RETURNS INTEGER;
FUNCTION GetStringByName(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE)
RETURNS VARCHAR(32765) CHARACTER SET NONE;
FUNCTION GetWideStringByName(Self TY$POINTER, Name VARCHAR(128) CHARACTER SET NONE)
RETURNS BLOB SUB_TYPE TEXT;
END

```

As you can see from the comment, this package is a blueprint for the `TlkJSONObject` class. It is designed to handle object values.

The `New_` function creates and returns a pointer to a new empty object. If `UseHash` is set to `TRUE` (the default value), then the `HASH` table will be used to search for fields within the object, otherwise the search will be performed by simple iteration.

The `Dispose` function is designed to release a pointer to a JSON object. Pointers to be forcibly freed are the result of parsing or JSON generation. You should not call it on intermediate objects when parsing or assembling JSON. It is only required for the top-level object.

The `ForEach` procedure retrieves each object field from the JSON pointer specified in `Self`. The following values are returned:

- `Idx` — the index of the list item or the number of the field in the object. Starts at 0.
- `Name` — the name of the next field, if `Self` is an object. Or the index of the list item, starting at 0, if `Self` is a list.
- `Obj` is a pointer to a key-value pair (to handle such a pair, you must use the `JS$METH` package).

The `Add_` function adds a new field to the object, the pointer to which is specified in the `Self` parameter. The element to add is specified in the `Obj` parameter, which must be a pointer to one of the `TlkJSONbase` descendants. The field name is specified in the `Name` parameter. The function

returns the index of the newly added field.

The `AddBoolean` function adds a new boolean field to the object pointed to by the `Self` parameter. The field name is specified in the `Name` parameter. The field value is specified in the `Bool` parameter. The function returns the index of the newly added field.

The `AddDouble` function adds a new field of real type to the object, the pointer to which is specified in the `Self` parameter. The field name is specified in the `Name` parameter. The field value is specified in the `Dbl` parameter. The function returns the index of the newly added field.

The `AddInteger` function adds a new integer field to the object pointed to by the `Self` parameter. The field name is specified in the `Name` parameter. The field value is specified in the `Int_` parameter. The function returns the index of the newly added field.

The `AddString` function adds a new field of string type (`VARCHAR (N)`) to the object pointed to by the `Self` parameter. The field name is specified in the `Name` parameter. The field value is specified in the `Int_` parameter. The function returns the index of the newly added field.

The `AddWideString` function adds a new `BLOB SUB_TYPE TEXT` field to the object pointed to by the `Self` parameter. The field name is specified in the `Name` parameter. The field value is specified in the `Int_` parameter. The function returns the index of the newly added field.

The `Delete_` function removes a field from the object with the `Idx` index. The function returns 0.

The `IndexOfName` function returns the index of a field by its name. A pointer to an object is specified in the `Self` parameter. The `Obj` parameter specifies a pointer to the element whose index is being defined.

The `IndexOfObject` function returns the index of a field value in an object. A pointer to an object is specified in the `Self` parameter. The `Obj` parameter specifies a pointer to the values of the field whose index is being determined.

The `Field` function returns or sets the value of a field by its name. A pointer to an object is specified in the `Self` parameter. The field name is specified in the `Name` parameter. Instead of a field name, you can specify the item number in the list or the field number. Numbering starts from 0. If a value other than `NULL` is specified in the `Obj` parameter, then the new value will be written in the field, otherwise the function will return a pointer to the field value.

The `Count_` function returns the number of fields in the object specified in the `Self` parameter.

The `Child` function returns or sets the value for the element at index `Idx` in the `Self` object. Indexing starts from 0. If the `Obj` parameter is not specified, then it returns a pointer to the element from the `Idx` indices. If `Obj` is specified, then sets its value to the element with indices `Idx`. Note `Obj` is a pointer to one of the `TlkJSONbase` descendants.

The `Generate` function returns a pointer to a `TlkJSONobject`, which is a JSON object. If `UseHash` is set to `TRUE` (the default value), then the `HASH` table will be used to search for fields within the object, otherwise the search will be performed by simple iteration. In the `Self` parameter, a pointer to the object is passed on the basis of which a new object of the `TlkJSONobject` type is created.

The `SelfType` function returns the object type for the pointer specified in the `Self` parameter. The

object type is returned as a number. If the `Self` parameter is not specified, it will return 6.

The `SelfTypeName` function returns the object type for the pointer specified in the `Self` parameter. The object type is returned as a string. If the `Self` parameter is not specified, it will return `'jsObject'`.

The `FieldByIndex` function returns or sets the property as a key-value pair at the specified `Idx` index. A pointer to an object is specified in the `Self` parameter. You must use the `JS$METH` package to handle the key-value pair. If a value other than `NULL` is specified in the `Obj` parameter, then the new value will be written to the field at the specified index, otherwise the function will return a pointer to the field.

The `NameOf` function returns the name of the field at its index specified in the `Idx` parameter. A pointer to an object is specified in the `Self` parameter.

The `GetBoolean` function returns the boolean value of the object field with the `Idx` index. Indexing starts at 0.

The `GetDouble` function returns the floating point value of the field of the object with the `Idx` index. Indexing starts at 0.

The `GetInteger` function returns the integer value of the object field with the `Idx` index. Indexing starts at 0.

The `GetString` function returns the character value (`VARCHAR`) of the object field with index `Idx`. Indexing starts at 0.

The `GetWideString` function returns the `BLOB SUB_TYPE TEXT` of the object field with the `Idx` index. Indexing starts at 0.

The `GetBooleanByName` function returns the boolean value of an object field by its name `Name`.

The `GetDoubleByName` function returns the floating point value of an object field by its name `Name`.

The `GetIntegerByName` function returns the integer value of the object field by its name `Name`.

The `GetStringByName` function returns the character value (`VARCHAR`) of an object field by its name `Name`.

The `GetWideStringByName` function returns the `BLOB SUB_TYPE TEXT` of an object field by its `Name`.

13.3.10. JS\$PTR package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE JS$PTR
AS
BEGIN
  FUNCTION New_
    (UsePtr CHAR(3) CHARACTER SET NONE /* Tra - Transaction, Att - Attachment */,
    UseHash BOOLEAN = TRUE)
    RETURNS TY$POINTER;
  FUNCTION Dispose(UsePtr CHAR(3) CHARACTER SET NONE) RETURNS SMALLINT;

  FUNCTION Tra RETURNS TY$POINTER;
  FUNCTION Att RETURNS TY$POINTER;

  FUNCTION isNull(jsPtr TY$POINTER) RETURNS BOOLEAN;
END

```

This package helps keep track of pointers that occur when creating JSON objects.

The `New_` function creates and returns a pointer to a new empty object. If the value 'Tra' is passed to the `UsePtr` parameter, then the pointer will be attached to the transaction, and upon its completion it will be automatically deleted. If the 'Att' value is passed to the `UsePtr` parameter, then the pointer will be attached to the connection, and when it is closed, it will be automatically deleted. If `UseHash` is set to `TRUE` (the default value), then the `HASH` table will be used to search for fields within the object, otherwise the search will be performed by simple iteration.

The `Dispose` function removes a pointer to a JSON object bound to a transaction or connection. If the 'Tra' value is passed to the `UsePtr` parameter, the pointer associated with the transaction will be deleted. If the 'Att' value is passed to the `UsePtr` parameter, then the pointer bound to the connection will be deleted.

The `Tra` function returns the pointer associated with the transaction.

The `Att` function returns the pointer attached to the connection.

The `isNull` function checks if the pointer is not null (with a null address). A null pointer returns the functions `js$func.ParseText` and `js$func.ParseString` in case of incorrect JSON input, access to a nonexistent field through the `Field` method, and more. This function can be used to detect such errors.

13.3.11. JS\$STR package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE JS$STR
AS
BEGIN
  /* TlkJSONbase = class
  TlkJSONstring = class(TlkJSONbase)
  */
  FUNCTION Value_(Self TY$POINTER, Str VARCHAR(32765) CHARACTER SET NONE = NULL /* Get
  */) RETURNS VARCHAR(32765) CHARACTER SET NONE;
  FUNCTION WideValue_(Self TY$POINTER, WStr BLOB SUB_TYPE TEXT = NULL /* Get */)
  RETURNS BLOB SUB_TYPE TEXT;

  FUNCTION Generate(Self TY$POINTER = NULL /* NULL - class function */, Str VARCHAR
  (32765) CHARACTER SET NONE = '') RETURNS TY$POINTER;
  FUNCTION WideGenerate(Self TY$POINTER = NULL /* NULL - class function */, WStr BLOB
  SUB_TYPE TEXT = '') RETURNS TY$POINTER;

  FUNCTION SelfType(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
  SMALLINT;
  FUNCTION SelfTypeName(Self TY$POINTER = NULL /* NULL - class function */) RETURNS
  VARCHAR(32) CHARACTER SET NONE;
END

```

As you can see from the comment, this package is a blueprint for the TlkJSONstring class. It is designed to handle string values.

The Value_ function returns or sets to a value of a string type (VARCHAR (N)) for the object specified in the Self parameter. If the Str parameter is not specified, then the value will be returned, if specified — set. Note that NULL is not returned and cannot be set by this method, there is a separate JS\$NULL package for this.

The WideValue_ function returns or sets to a value of the BLOB SUB_TYPE TEXT type for the object specified in the Self parameter. If the Str parameter is not specified, then the value will be returned, if specified — set. Note that NULL is not returned and cannot be set by this method, there is a separate JS\$NULL package for this.

The Generate function returns a pointer to a `TlkJSONstring` object, which is a JSON string value. The Self parameter is a pointer to a JSON object on the basis of which a new TlkJSONstring object is created. The string value is specified in the Str parameter.

The WideGenerate function returns a pointer to the TlkJSONstring object, which is a JSON string value. The Self parameter is a pointer to a JSON object for which a long string value (BLOB SUB_TYPE TEXT) is set in the Str parameter. The value of the Self parameter will be returned by the function if it is non-NULL, otherwise it will return a pointer to a new TlkJSONstring object.

The SelfType function returns the object type for the pointer specified in the Self parameter. The object type is returned as a number. If the Self parameter is not specified, it will return 2.

The SelfTypeName function returns the object type for the pointer specified in the Self parameter. The object type is returned as a string. If the Self parameter is not specified, it will return

'jsString'.

13.4. Examples

13.4.1. Building JSON

Let's take the employee database as an example.



The examples use a modified employee database converted to UTF8 encoding.

The MAKE_JSON_DEPARTMENT_TREE function displays a list of departments in JSON format in a hierarchical format.

```
CREATE OR ALTER FUNCTION MAKE_JSON_DEPARTMENT_TREE
RETURNS BLOB SUB_TYPE TEXT
AS
  DECLARE VARIABLE JSON_TEXT BLOB SUB_TYPE TEXT;
  DECLARE VARIABLE JSON          TY$POINTER;
  DECLARE VARIABLE JSON_SUB_DEPS TY$POINTER;
BEGIN
  JSON = JS$OBJ.NEW_();
  FOR
    WITH RECURSIVE R
    AS (SELECT
      :JSON AS JSON,
      CAST(NULL AS TY$POINTER) AS PARENT_JSON,
      D.DEPT_NO,
      D.DEPARTMENT,
      D.HEAD_DEPT,
      D.MNGR_NO,
      D.BUDGET,
      D.LOCATION,
      D.PHONE_NO
    FROM DEPARTMENT D
    WHERE D.HEAD_DEPT IS NULL
    UNION ALL
    SELECT
      JS$OBJ.NEW_() AS JSON,
      R.JSON,
      D.DEPT_NO,
      D.DEPARTMENT,
      D.HEAD_DEPT,
      D.MNGR_NO,
      D.BUDGET,
      D.LOCATION,
      D.PHONE_NO
    FROM DEPARTMENT D
    JOIN R
      ON D.HEAD_DEPT = R.DEPT_NO)
```

```

SELECT
    JSON,
    PARENT_JSON,
    DEPT_NO,
    DEPARTMENT,
    HEAD_DEPT,
    MNGR_NO,
    BUDGET,
    LOCATION,
    PHONE_NO
FROM R AS CURSOR C_DEP
DO
BEGIN
    -- for each departure, fill in the value of the JSON object fields
    JS$OBJ.ADDSTRING(C_DEP.JSON, 'dept_no', C_DEP.DEPT_NO);
    JS$OBJ.ADDSTRING(C_DEP.JSON, 'department', C_DEP.DEPARTMENT);
    IF (C_DEP.HEAD_DEPT IS NOT NULL) THEN
        JS$OBJ.ADDSTRING(C_DEP.JSON, 'head_dept', C_DEP.HEAD_DEPT);
    ELSE
        JS$OBJ.ADD_(C_DEP.JSON, 'head_dept', JS$NULL.GENERATE());
    IF (C_DEP.MNGR_NO IS NOT NULL) THEN
        JS$OBJ.ADDINTEGER(C_DEP.JSON, 'mngr_no', C_DEP.MNGR_NO);
    ELSE
        JS$OBJ.ADD_(C_DEP.JSON, 'mngr_no', JS$NULL.GENERATE());
    -- here ADDSTRING is probably better, since it is guaranteed to preserve the
    precision of the number
    JS$OBJ.ADDDOUBLE(C_DEP.JSON, 'budget', C_DEP.BUDGET);
    JS$OBJ.ADDSTRING(C_DEP.JSON, 'location', C_DEP.LOCATION);
    JS$OBJ.ADDSTRING(C_DEP.JSON, 'phone_no', C_DEP.PHONE_NO);
    -- add a list to each departure in which the subordinate departures will be
    entered.
    JS$OBJ.ADD_(C_DEP.JSON, 'departments', JS$LIST.GENERATE());
    IF (C_DEP.PARENT_JSON IS NOT NULL) THEN
        BEGIN
            -- where there are departments, there is also an object of the parent JSON
            object,
            -- we get a field with a list from it
            JSON_SUB_DEPS = JS$OBJ.FIELD(C_DEP.PARENT_JSON, 'departments');
            -- and add the current departure to it
            JS$LIST.ADD_(JSON_SUB_DEPS, C_DEP.JSON);
        END
    END
    -- generate JSON as text
    JSON_TEXT = JS$FUNC.READABLETEXT(JSON);
    -- don't forget to release the pointer
    JS$OBJ.DISPOSE(JSON);
    RETURN JSON_TEXT;
WHEN ANY DO
BEGIN
    -- if there was an error, release the pointer anyway
    JS$OBJ.DISPOSE(JSON);

```

```
EXCEPTION;
END
END
```

Here's a trick: at the very top level of the recursive statement, a pointer to a previously created JSON root object is used. In the recursive part of the query, we output a JSON object for the parent departure PARENT_JSON and a JSON object for the current departure PARENT_JSON. Thus, we always know in which JSON object to add the departure.

Then we loop through the cursor and add field values for the current departure at each iteration. Note that in order to add a NULL value, you have to use the JS\$NULL.GENERATE() call. If you don't, then when you call JS\$OBJ.ADDSTRING (C_DEP.JSON, 'head_dept', C_DEP.HEAD_DEPT) when C_DEP.HEAD_DEPT is NULL, the head_dept field will simply not be added.

Also, for each department, you need to add a JSON list to which subordinate departments will be added.

If the JSON object of the parent unit is not NULL, then we get the list added for it differently using the JS\$OBJ.FIELD function and add the current JSON object to it.

Further, the JSON of the object of the highest level, you can generate the text, after which we no longer need the object itself and we need to clear the pointer allocated for it using the JS\$OBJ.DISPOSE function.

Pay attention to the WHEN ANY DO exception handling block. It is required, because even when it happens, we need to free the pointer to avoid a memory leak.

13.4.2. Parse JSON

Parsing JSON is somewhat more difficult than collecting it. The fact is that you need to take into account that incorrect JSON may be received at the input, not only by itself, but also with a structure that does not correspond to your logic.

Suppose you have a JSON that contains a list of people with their characteristics.

This JSON looks like this:

```
[
  {"id": 1, "name": "John"},
  {"id": 2, "name": null}
]
```

The following procedure returns a list of people from this JSON:

```
create exception e_custom_error 'custom error';

set term ^;

CREATE OR ALTER PROCEDURE PARSE_PEOPLES_JSON (
```



```

JSON_STR BLOB SUB_TYPE TEXT)
RETURNS (
  ID INTEGER,
  NAME VARCHAR(120))
AS
declare variable json TY$POINTER;
declare variable jsonId TY$POINTER;
declare variable jsonName TY$POINTER;
begin
  json = js$func.parsetext(json_str);
  -- If JSON incorrect js$func.parsetext will not throw an exception,
  -- but return a null pointer, so you need to handle this case yourself
  if (js$ptr.isNull(json)) then
    exception e_custom_error 'invalid json';
  -- Again, functions from this library do not check the correctness of element types
  -- and do not return an understandable error. We need to check if the type we are
processing.
  -- Otherwise, when calling js$list.foreach, an "Access violation" will occur
  if (js$base.SelfTypeName(json) != 'jsList') then
    exception e_custom_error 'Invalid JSON format. The top level of the JSON item must
be a list.';
  for
    select Obj
    from js$list.foreach(:json)
    as cursor c
  do
    begin
      -- Checking that the array element is an object,
      -- Otherwise, when calling js$obj.GetIntegerByName, an "Access violation" will
occur
      if (js$base.SelfTypeName(c.Obj) != 'jsObject') then
        exception e_custom_error 'Element of list is not object';
      -- js$obj.GetIntegerByName does not check for the existence of an element with the
given name
      -- it will simply return 0 if it is missing. Therefore, such a check must be done
independently.
      -- And js$obj.Field will return a null pointer.
      if (js$obj.indexofname(c.Obj, 'id') < 0) then
        exception e_custom_error 'Field "id" not found in object';
      jsonId = js$obj.Field(c.Obj, 'id');
      if (js$base.selftypename(jsonId) = 'jsNull') then
        id = null;
      else if (js$base.selftypename(jsonId) = 'jsNumber') then
        id = js$obj.GetIntegerByName(c.Obj, 'id');
      else
        exception e_custom_error 'Field "id" is not number';

      if (js$obj.indexofname(c.Obj, 'name') < 0) then
        exception e_custom_error 'Field "name" not found in object';
      jsonName = js$obj.Field(c.Obj, 'name');
      if (js$str.selftypename(jsonName) = 'jsNull') then

```

```

        name = null;
    else
        name = js$str.value_(jsonName);
    suspend;
end
js$base.dispose(json);
when any do
begin
    js$base.dispose(json);
    exception;
end
end^

set term ;^

```

Run the following query to check if it is correct

```

select id, name
from parse_peoples_json( ' [{"id": 1, "name": "John"}, {"id": 2, "name": null}] ' )

```

Let's take a closer look at the JSON parsing script. The first feature is that the `js$func.parsetext` function will not throw an exception if any other string is input instead of JSON. It will just return a null pointer. But, this is not NULL as you thought, but a pointer with the content `x'0000000000000000'`. Therefore, after executing this function, you need to check what was returned to you, otherwise the calls of the subsequent functions will return an "Access violation" error.

Next, it is important to check what type of JSON object was returned. If an object or any other type appears in the input instead of a list, then the `js$list.foreach` call will cause an "Access violation". The same will happen if you call any other function that expects a pointer to a different type that is not intended for it.

The next feature is that you should always check for the presence of fields (object properties). If there is no field with the specified name, then in some cases an incorrect value may be returned (as in the case of `js$obj.GetIntegerByName`), in others it will lead to a type conversion error.

Note that functions like `js$obj.GetIntegerByName` or `js$obj.GetSrtingByName` cannot return NULL. To recognize a null value, you need to check the field type with the `js$base.selftypename` function.

As with the JSON assembly, remember to free the top-level JSON pointer and also do this in the WHEN ANY DO exception handling block.

Below is an example of parsing JSON that was collected by the `MAKE_JSON_DEPARTMENT_TREE` function in the example above. The text of the example contains comments explaining the principle of parsing.

```

SET TERM ^ ;

CREATE OR ALTER PACKAGE JSON_PARSE_DEPS

```

```

AS
BEGIN
  PROCEDURE PARSE_DEPARTMENT_TREE (
    JSON_TEXT BLOB SUB_TYPE TEXT)
  RETURNS (
    DEPT_NO    CHAR(3),
    DEPARTMENT VARCHAR(25),
    HEAD_DEPT  CHAR(3),
    MNGR_NO    SMALLINT,
    BUDGET     DECIMAL(18,2),
    LOCATION   VARCHAR(15),
    PHONE_NO   VARCHAR(20));
END^

```

```

RECREATE PACKAGE BODY JSON_PARSE_DEPS

```

```

AS
BEGIN
  PROCEDURE GET_DEPARTMENT_INFO (
    JSON TY$POINTER)
  RETURNS (
    DEPT_NO    CHAR(3),
    DEPARTMENT VARCHAR(25),
    HEAD_DEPT  CHAR(3),
    MNGR_NO    SMALLINT,
    BUDGET     DECIMAL(18,2),
    LOCATION   VARCHAR(15),
    PHONE_NO   VARCHAR(20),
    JSON_LIST  TY$POINTER);

```

```

  PROCEDURE PARSE_DEPARTMENT_TREE (
    JSON_TEXT BLOB SUB_TYPE TEXT)
  RETURNS (
    DEPT_NO    CHAR(3),
    DEPARTMENT VARCHAR(25),
    HEAD_DEPT  CHAR(3),
    MNGR_NO    SMALLINT,
    BUDGET     DECIMAL(18,2),
    LOCATION   VARCHAR(15),
    PHONE_NO   VARCHAR(20))

```

```

AS
  DECLARE VARIABLE JSON    TY$POINTER;
BEGIN
  JSON = JS$FUNC.PARSETEXT(JSON_TEXT);
  -- If JSON is incorrect js$func.parsetext will not throw an exception,
  -- but simply return a null pointer, so you need to handle this case
  -- yourself.
  IF (JS$PTR.ISNULL(JSON)) THEN
    EXCEPTION E_CUSTOM_ERROR 'invalid json';
  FOR
    SELECT
      INFO.DEPT_NO,

```

```

        INFO.DEPARTMENT,
        INFO.HEAD_DEPT,
        INFO.MNGR_NO,
        INFO.BUDGET,
        INFO.LOCATION,
        INFO.PHONE_NO
    FROM JSON_PARSE_DEPS.GET_DEPARTMENT_INFO(:JSON) INFO
    INTO
        :DEPT_NO,
        :DEPARTMENT,
        :HEAD_DEPT,
        :MNGR_NO,
        :BUDGET,
        :LOCATION,
        :PHONE_NO
    DO
        SUSPEND;
    JS$OBJ.DISPOSE(JSON);
    WHEN ANY DO
    BEGIN
        JS$OBJ.DISPOSE(JSON);
    EXCEPTION;
    END
END

PROCEDURE GET_DEPARTMENT_INFO (
    JSON TY$POINTER)
    RETURNS (
        DEPT_NO    CHAR(3),
        DEPARTMENT VARCHAR(25),
        HEAD_DEPT  CHAR(3),
        MNGR_NO    SMALLINT,
        BUDGET     DECIMAL(18,2),
        LOCATION   VARCHAR(15),
        PHONE_NO   VARCHAR(20),
        JSON_LIST  TY$POINTER)
    AS
    BEGIN
        IF (JS$OBJ.INDEXOFNAME(JSON, 'dept_no') < 0) THEN
            EXCEPTION E_CUSTOM_ERROR 'field "dept_no" not found';
        DEPT_NO = JS$OBJ.GETSTRINGBYNAME(JSON, 'dept_no');
        IF (JS$OBJ.INDEXOFNAME(JSON, 'department') < 0) THEN
            EXCEPTION E_CUSTOM_ERROR 'field "department" not found';
        DEPARTMENT = JS$OBJ.GETSTRINGBYNAME(JSON, 'department');
        IF (JS$OBJ.INDEXOFNAME(JSON, 'head_dept') < 0) THEN
            EXCEPTION E_CUSTOM_ERROR 'field "head_dept" not found';
        IF (JS$BASE.SELFTYPENAME(JS$OBJ.FIELD(JSON, 'head_dept')) = 'jsNull') THEN
            HEAD_DEPT = NULL;
        ELSE
            HEAD_DEPT = JS$OBJ.GETSTRINGBYNAME(JSON, 'head_dept');
        IF (JS$OBJ.INDEXOFNAME(JSON, 'mng_r_no') < 0) THEN

```

```

EXCEPTION E_CUSTOM_ERROR 'field "mngr_no" not found';
IF (JS$BASE.SELFTYPENAME(JS$OBJ.FIELD(JSON, 'mngr_no')) = 'jsNull') THEN
  MNGR_NO = NULL;
ELSE
  MNGR_NO = JS$OBJ.GETINTEGERBYNAME(JSON, 'mngr_no');
IF (JS$OBJ.INDEXOFNAME(JSON, 'budget') < 0) THEN
  EXCEPTION E_CUSTOM_ERROR 'field "budget" not found';
BUDGET = JS$OBJ.GETDOUBLEBYNAME(JSON, 'budget');
IF (JS$OBJ.INDEXOFNAME(JSON, 'location') < 0) THEN
  EXCEPTION E_CUSTOM_ERROR 'field "location" not found';
LOCATION = JS$OBJ.GETSTRINGBYNAME(JSON, 'location');
IF (JS$OBJ.INDEXOFNAME(JSON, 'phone_no') < 0) THEN
  EXCEPTION E_CUSTOM_ERROR 'field "phone_no" not found';
PHONE_NO = JS$OBJ.GETSTRINGBYNAME(JSON, 'phone_no');
IF (JS$OBJ.INDEXOFNAME(JSON, 'departments') >= 0) THEN
BEGIN
  -- get a list of child departures
  JSON_LIST = JS$OBJ.FIELD(JSON, 'departments');
  IF (JS$BASE.SELFTYPENAME(JSON_LIST) != 'jsList') THEN
    EXCEPTION E_CUSTOM_ERROR 'Invalid JSON format. Field "departments" must be
list';
  SUSPEND;
  -- This list is traversed and the procedure for retrieving information about
each
  -- departure is recursively called for it.
FOR
  SELECT
    INFO.DEPT_NO,
    INFO.DEPARTMENT,
    INFO.HEAD_DEPT,
    INFO.MNGR_NO,
    INFO.BUDGET,
    INFO.LOCATION,
    INFO.PHONE_NO,
    INFO.JSON_LIST
  FROM JS$LIST.FOREACH(:JSON_LIST) L
  LEFT JOIN JSON_PARSE_DEPS.GET_DEPARTMENT_INFO(L.OBJ) INFO
  ON TRUE
  INTO
    :DEPT_NO,
    :DEPARTMENT,
    :HEAD_DEPT,
    :MNGR_NO,
    :BUDGET,
    :LOCATION,
    :PHONE_NO,
    :JSON_LIST
DO
  SUSPEND;
END
ELSE

```

```
EXCEPTION E_CUSTOM_ERROR 'Invalid JSON format. Field "departments" not found' ||  
DEPT_NO;  
    END  
END  
^  
  
SET TERM ; ^
```

Chapter 14. NANODBC-UDR — working with ODBC Data Sources

Starting from version 2.5, Firebird DBMS has the ability to work with external data through the `EXECUTE STATEMENT .. ON EXTERNAL DATA SOURCE` statement. Unfortunately, working with external data sources is limited only to Firebird databases.

To be able to work with other DBMS, UDR `nanodbc` was developed. Fully open source library licensed under the MIT license and free to use. It is written in C++. Source code is available at <https://github.com/mnf71/udr-nanodbc>

The library is based on a thin C++ wrapper around the native C ODBC API <https://github.com/nanodbc/nanodbc>

14.1. Install UDR `nanodbc`

To be able to use UDR `nanodbc`, it must be registered in your database. To do this, you need to execute the `plugin/UDR/nanodbc_install.sql` script.

14.2. How it works?

UDR `nanodbc` is based on the free library `nanodbc`, therefore, for a complete understanding, we recommend that you study the API of this library in its source codes and documentation (see <https://github.com/mnf71/udr-nanodbc>).

When working with library objects, so-called descriptors (pointers to `nanodbc` objects) are used. Pointers are described by a domain defined as:

```
CREATE DOMAIN TY$POINTER AS
CHAR(8) CHARACTER SET OCTETS;
```

in Firebird 4.0 it can be described in the following way

```
CREATE DOMAIN TY$POINTER AS BINARY(8);
```

After finishing work with the object, the pointer to it must be released using the `release_()` functions, which are located in the corresponding PSQL packages. Which package to use depends on the type of object you want to free the pointer to.

In HQbird, it is impossible to create a function that does not return a result, therefore, for C++ functions with a void return type, UDR functions return the type described by the `TY$NANO_BLANK` domain. It makes no sense to analyze the result of such functions. Domain `TY$NANO_BLANK` is described as:

```
CREATE DOMAIN TY$NANO_BLANK AS SMALLINT;
```

Before you start working with the UDR, you need to initialize the nanodbc library. This is done by calling the `nano$udr.initialize()` function. And upon completion of the work, call finalization function `nano$udr.finalize()`. It is recommended to call `nano$udr.initialize()` function in the ON CONNECT trigger, and `nano$udr.finalize()` function in the ON DISCONNECT trigger.

14.3. Description of PSQL packages from UDR-nanodbc

14.3.1. NANO\$UDR package

The header of this package looks like this:

```
CREATE OR ALTER PACKAGE NANO$UDR
AS
BEGIN

    FUNCTION initialize RETURNS TY$NANO_BLANK;
    FUNCTION finalize RETURNS TY$NANO_BLANK;
    FUNCTION expunge RETURNS TY$NANO_BLANK;

    FUNCTION locale(
        set_locale VARCHAR(20) CHARACTER SET NONE DEFAULT NULL /* NULL - Get */
    ) RETURNS CHARACTER SET NONE VARCHAR(20);

    FUNCTION error_message RETURNS VARCHAR(512) CHARACTER SET UTF8;

END
```

The NANO\$UDR package contains functions for initializing and finalizing UDRs.

The `initialize()` function initializes the nanodbc UDR. It is recommended to call this function in the ON CONNECT trigger. It must be called before the first call to any other function from the nanodbc UDR.

The `finalize()` function terminates the nanodbc UDR. After calling it, it is impossible to work with UDR nanodbc. When called, the function automatically releases all previously allocated resources. It is recommended to call this function in the ON DISCONNECT trigger.

The `expunge()` function automatically releases all previously allocated resources (connections, transactions, prepared statements, cursors).

The `locale()` function returns or sets the default encoding for connections. If the `set_locale` parameter is specified, then a new encoding will be set, otherwise the function will return the value of the current encoding. This is necessary to transform transmitted and received strings, before and after exchanging with an ODBC source. The default encoding is cp1251.

If initially the connection to the database is established with UTF8 encoding, then you can set utf8, according to the names of iconv. If the encoding is NONE, then it is better to convert to your language encoding using the convert_[var]char() functions.

The error_message() function returns the text of the last error.

14.3.2. NANO\$CONN package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE NANO$CONN
AS
BEGIN

  /* Note:
    CHARACTER SET UTF8
    attr VARCHAR(512) CHARACTER SET UTF8 DEFAULT NULL
    ...
    CHARACTER SET WIN1251
    attr VARCHAR(2048) CHARACTER SET WIN1251 DEFAULT NULL
  */

  FUNCTION connection(
    attr VARCHAR(512) CHARACTER SET UTF8 DEFAULT NULL,
    user_ VARCHAR(63) CHARACTER SET UTF8 DEFAULT NULL,
    pass VARCHAR(63) CHARACTER SET UTF8 DEFAULT NULL,
    timeout INTEGER NOT NULL DEFAULT 0
  ) RETURNS TY$POINTER;

  FUNCTION valid(conn TY$POINTER NOT NULL) RETURNS BOOLEAN;

  FUNCTION release_(conn TY$POINTER NOT NULL) RETURNS TY$POINTER;
  FUNCTION expunge(conn ty$pointer NOT NULL) RETURNS TY$NANO_BLANK;

  FUNCTION allocate(conn ty$pointer NOT NULL) RETURNS TY$NANO_BLANK;
  FUNCTION deallocate(conn ty$pointer NOT NULL) RETURNS TY$NANO_BLANK;

  FUNCTION txn_read_uncommitted RETURNS SMALLINT;
  FUNCTION txn_read_committed RETURNS SMALLINT;
  FUNCTION txn_repeatable_read RETURNS SMALLINT;
  FUNCTION txn_serializable RETURNS SMALLINT;

  FUNCTION isolation_level(
    conn TY$POINTER NOT NULL,
    level_ SMALLINT DEFAULT NULL /* NULL - get usage */
  ) RETURNS SMALLINT;

  FUNCTION connect_(
    conn TY$POINTER NOT NULL,
    attr VARCHAR(512) CHARACTER SET UTF8 NOT NULL,

```

```

    user_ VARCHAR(63) CHARACTER SET UTF8 DEFAULT NULL,
    pass VARCHAR(63) CHARACTER SET UTF8 DEFAULT NULL,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION connected(conn TY$POINTER NOT NULL) RETURNS BOOLEAN;

FUNCTION disconnect_(conn ty$pointer NOT NULL) RETURNS TY$NANO_BLANK;

FUNCTION transactions(conn TY$POINTER NOT NULL) RETURNS INTEGER;

FUNCTION get_info(conn TY$POINTER NOT NULL, info_type SMALLINT NOT NULL)
    RETURNS VARCHAR(256) CHARACTER SET UTF8;

FUNCTION dbms_name(conn ty$pointer NOT NULL) RETURNS VARCHAR(128) CHARACTER SET
UTF8;
FUNCTION dbms_version(conn ty$pointer NOT NULL) RETURNS VARCHAR(128) CHARACTER SET
UTF8;
FUNCTION driver_name(conn TY$POINTER NOT NULL) RETURNS VARCHAR(128) CHARACTER SET
UTF8;
FUNCTION database_name(conn TY$POINTER NOT NULL) RETURNS VARCHAR(128) CHARACTER SET
UTF8;
FUNCTION catalog_name(conn TY$POINTER NOT NULL) RETURNS VARCHAR(128) CHARACTER SET
UTF8;

END

```

The NANO\$CONN package contains functions for setting up an ODBC data source and getting some connection information.

The `connection()` function establishes a connection to an ODBC data source. If more than one parameter is not specified, the function will return a pointer to the "connection" object. The actual connection to the ODBC data source can be made later using the `connect_()` function. Function parameters:

- `attr` specifies the connection string or the so-called DSN;
- `user_` specifies the username;
- `pass` sets the password;
- `timeout` specifies the idle timeout.

The `valid()` function returns whether the connection object pointer is valid.

The `release_()` function releases the connection pointer and all associated resources (transactions, prepared statements, cursors).

The `expunge()` function releases all resources associated with the connection (transactions, prepared statements, cursors).

The `allocate()` function allows you to allocate descriptors on demand for setting the environment

and ODBC attributes prior to establishing a connection to the database. Typically, the user does not need to make this call explicitly.

The `deallocate()` function frees the connection handles.

The `txn_read_uncommitted()` function returns the numeric constant required to set the transaction isolation level to `READ UNCOMMITTED`.

The `txn_read_committed()` function returns the numeric constant required to set the transaction isolation level to `READ COMMITTED`.

The `txn_repeatable_read()` function returns a numeric constant required to set the isolation level of the `REPEATABLE READ` transaction.

The `txn_serializable()` function returns the numeric constant required to set the transaction isolation level to `SERIALIZABLE`.

The `isolation_level()` function sets the isolation level for new transactions. Function parameters:

- `conn` - pointer to the connection object;
- `attr` specifies the connection string or the so-called DSN;
- `user_` specifies the username;
- `pass` sets the password;
- `timeout` specifies the idle timeout.

The `connected()` function returns whether a database connection has been established for the given pointer to the connection object.

The `disconnect_()` function disconnects from the database. A pointer to the connection object is passed as a parameter.

The `transactions()` function returns the number of active transactions for a given connection.

The `get_info()` function returns various information about the driver or data source. This low-level function is the ODBC analogue of the `SQLGetInfo` function. It is not recommended use it directly. Function parameters:

- `conn` - pointer to the connection object;
- `info_type` - the type of information returned. Numeric constants with return types can be found at <https://github.com/microsoft/ODBC-Specification/blob/master/Windows/inc/sql.h>

The `dbms_name()` function returns the name of the DBMS to which the connection was made.

The `dbms_version()` function returns the version of the DBMS to which the connection was made.

The `driver_name()` function returns the name of the driver.

The `database_name()` function returns the name of the database to which the connection was made.

The `catalog_name()` function returns the name of the database catalog to which the connection was

made.

14.3.3. NANO\$TNX package

The header of this package looks like this:

```
CREATE OR ALTER PACKAGE NANO$TNX
AS
BEGIN

    FUNCTION transaction_(conn TY$POINTER NOT NULL) RETURNS TY$POINTER;

    FUNCTION valid(tnx TY$POINTER NOT NULL) RETURNS BOOLEAN;

    FUNCTION release_(tnx ty$pointer NOT NULL) RETURNS TY$POINTER;

    FUNCTION connection(tnx TY$POINTER NOT NULL) RETURNS TY$POINTER;

    FUNCTION commit_(tnx TY$POINTER NOT NULL) RETURNS TY$NANO_BLANK;

    FUNCTION rollback_(tnx TY$POINTER NOT NULL) RETURNS TY$NANO_BLANK;

END
```

The NANO\$TNX package contains functions for explicitly managing transactions.

The `transaction_()` function disables the automatic confirmation of the transaction and starts a new transaction with the isolation level specified in the `NANO$CONN.isolation_level()` function. The function returns a pointer to a new transaction.

The `valid()` function returns whether the pointer to the transaction object is valid.

The `release_()` function releases the pointer to the transaction object. When the pointer is freed, the uncommitted transaction is rolled back and the driver returns to the automatic transaction confirmation mode.

The `connection()` function returns a pointer to the connection for which the transaction was started.

The `commit_()` function confirms the transaction.

The `rollback_()` function rolls back the transaction.

14.3.4. NANO\$STMT package

The header of this package looks like this:

```
CREATE OR ALTER PACKAGE NANO$STMT
AS
```

BEGIN

```

FUNCTION statement_(
    conn TY$POINTER DEFAULT NULL,
    query VARCHAR(8191) CHARACTER SET UTF8 DEFAULT NULL,
    scrollable BOOLEAN DEFAULT NULL /* NULL - default ODBC driver */,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$POINTER;

FUNCTION valid(stmt TY$POINTER NOT NULL) RETURNS BOOLEAN;

FUNCTION release_(stmt TY$POINTER NOT NULL) RETURNS TY$POINTER;

FUNCTION connected(stmt TY$POINTER NOT NULL) RETURNS BOOLEAN;
FUNCTION connection(stmt TY$POINTER NOT NULL) RETURNS TY$POINTER;

FUNCTION open_(
    stmt TY$POINTER NOT NULL,
    conn TY$POINTER NOT NULL
) RETURNS TY$NANO_BLANK;

FUNCTION close_(stmt TY$POINTER NOT NULL) RETURNS TY$NANO_BLANK;

FUNCTION cancel(stmt TY$POINTER NOT NULL) RETURNS TY$NANO_BLANK;

FUNCTION closed(stmt TY$POINTER NOT NULL) RETURNS BOOLEAN;

FUNCTION prepare_direct(
    stmt TY$POINTER NOT NULL,
    conn TY$POINTER NOT NULL,
    query VARCHAR(8191) CHARACTER SET UTF8 NOT NULL,
    scrollable BOOLEAN DEFAULT NULL /* NULL - default ODBC driver */,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION prepare_(
    stmt TY$POINTER NOT NULL,
    query VARCHAR(8191) CHARACTER SET UTF8 NOT NULL,
    scrollable BOOLEAN DEFAULT NULL /* NULL - default ODBC driver */,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION scrollable(
    stmt TY$POINTER NOT NULL,
    usage_ BOOLEAN DEFAULT NULL /* NULL - get usage */
) RETURNS BOOLEAN;

FUNCTION timeout(
    stmt TY$POINTER NOT NULL,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

```

```

FUNCTION execute_direct(
    stmt TY$POINTER NOT NULL,
    conn TY$POINTER NOT NULL,
    query VARCHAR(8191) CHARACTER SET UTF8 NOT NULL,
    scrollable BOOLEAN DEFAULT NULL /* NULL - default ODBC driver */,
    batch_operations INTEGER NOT NULL DEFAULT 1,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$POINTER;

FUNCTION just_execute_direct(
    stmt TY$POINTER NOT NULL,
    conn TY$POINTER NOT NULL,
    query VARCHAR(8191) CHARACTER SET UTF8 NOT NULL,
    batch_operations INTEGER NOT NULL DEFAULT 1,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION execute_(
    stmt TY$POINTER NOT NULL,
    batch_operations INTEGER NOT NULL DEFAULT 1,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$POINTER;

FUNCTION just_execute(
    stmt TY$POINTER NOT NULL,
    batch_operations INTEGER NOT NULL DEFAULT 1,
    timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION procedure_columns(
    stmt TY$POINTER NOT NULL,
    catalog_ VARCHAR(128) CHARACTER SET UTF8 NOT NULL,
    schema_ VARCHAR(128) CHARACTER SET UTF8 NOT NULL,
    procedure_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL,
    column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS TY$POINTER;

FUNCTION affected_rows(stmt TY$POINTER NOT NULL) RETURNS INTEGER;
FUNCTION columns(stmt TY$POINTER NOT NULL) RETURNS SMALLINT;
FUNCTION parameters(stmt TY$POINTER NOT NULL) RETURNS SMALLINT;
FUNCTION parameter_size(stmt TY$POINTER NOT NULL, parameter_index SMALLINT NOT NULL)
RETURNS INTEGER;

-----

FUNCTION bind_smallint(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ SMALLINT
) RETURNS TY$NANO_BLANK;

```

```

FUNCTION bind_integer(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ INTEGER
) RETURNS TY$NANO_BLANK;

/*
FUNCTION bind_bigint(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ BIGINT
) RETURNS TY$NANO_BLANK;
*/

FUNCTION bind_float(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ FLOAT
) RETURNS TY$NANO_BLANK;

FUNCTION bind_double(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ DOUBLE PRECISION
) RETURNS TY$NANO_BLANK;

FUNCTION bind_varchar(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ VARCHAR(32765) CHARACTER SET NONE,
    param_size SMALLINT NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION bind_char(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ CHAR(32767) CHARACTER SET NONE,
    param_size SMALLINT NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION bind_u8_varchar(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ VARCHAR(8191) CHARACTER SET UTF8,
    param_size SMALLINT NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION bind_u8_char(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,

```

```

    value_ CHAR(8191) CHARACTER SET UTF8,
    param_size SMALLINT NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION bind_blob(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ BLOB
) RETURNS TY$NANO_BLANK;

FUNCTION bind_boolean(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ BOOLEAN
) RETURNS TY$NANO_BLANK;

FUNCTION bind_date(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ DATE
) RETURNS TY$NANO_BLANK;

/*
FUNCTION bind_time(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ TIME
) RETURNS TY$NANO_BLANK
EXTERNAL NAME 'nano!stmt_bind'
ENGINE UDR;
*/

FUNCTION bind_timestamp(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    value_ TIMESTAMP
) RETURNS TY$NANO_BLANK;

FUNCTION bind_null(
    stmt TY$POINTER NOT NULL,
    parameter_index SMALLINT NOT NULL,
    batch_size INTEGER NOT NULL DEFAULT 1 -- <> 1 call nulls all batch
) RETURNS TY$NANO_BLANK;

FUNCTION convert_varchar(
    value_ VARCHAR(32765) CHARACTER SET NONE,
    from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS VARCHAR(32765) CHARACTER SET NONE;

```



```

FUNCTION convert_char(
    value_ CHAR(32767) CHARACTER SET NONE,
    from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS CHAR(32767) CHARACTER SET NONE;

FUNCTION clear_bindings(stmt TY$POINTER NOT NULL) RETURNS TY$NANO_BLANK;

-----

FUNCTION describe_parameter(
    stmt TY$POINTER NOT NULL,
    idx SMALLINT NOT NULL,
    type_ SMALLINT NOT NULL,
    size_ INTEGER NOT NULL,
    scale_ SMALLINT NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION describe_parameters(stmt TY$POINTER NOT NULL) RETURNS TY$NANO_BLANK;

FUNCTION reset_parameters(stmt TY$POINTER NOT NULL, timeout INTEGER NOT NULL DEFAULT
0)
    RETURNS TY$NANO_BLANK;

END

```

The NANO\$STMT package contains functions for working with SQL queries.

The `statement_()` function creates and returns a pointer to an SQL query object. Parameters:

- `conn` - pointer to the connection object;
- `query` - the text of the SQL query;
- `scrollable` - whether the cursor is scrollable (if, of course, the operator returns a cursor), if not set (NULL value), then the default behavior of the ODBC driver is used;
- `timeout` - SQL statement timeout.

If no parameter is specified, then it returns a pointer to the newly created SQL query object, without binding to the connection. You can later associate this pointer with a connection and set other query properties.

The `valid()` function returns whether the pointer to the SQL query object is valid.

The `release_()` function releases the pointer to the SQL query object.

The `connected()` function returns whether the request is attached to a connection.

The `connection()` function is a pointer to the bound connection.

The `open_()` function opens a connection and binds it to the request. Parameters:

- `stmt` - pointer to SQL query;
- `conn` - connection pointer.

The `close_()` function closes a previously opened request and clears all resources allocated by the request.

The `cancel()` function cancels the execution of the request.

The `closed()` function returns whether the request is closed.

The `prepare_direct()` function prepares an SQL statement and binds it to the specified connection. Parameters:

- `stmt` - a pointer to the statement;
- `conn` - connection pointer;
- `query` - the text of the SQL query;
- `scrollable` - whether the cursor is scrollable (if, of course, the operator returns a cursor), if not set (NULL value), then the default behavior of the ODBC driver is used;
- `timeout` - SQL statement timeout.

The `prepare_()` function prepares the SQL query. Parameters:

- `stmt` - a pointer to the statement;
- `query` - the text of the SQL query;
- `scrollable` - whether the cursor is scrollable (if, of course, the operator returns a cursor), if not set (NULL value), then the default behavior of the ODBC driver is used;
- `timeout` - SQL statement timeout.

The `scrollable_()` function returns or sets whether the cursor is scrollable. Parameters:

- `stmt` - a pointer to the statement;
- `usage_` - whether the cursor is scrollable (if, of course, the operator returns a cursor), if not set (NULL value), then it returns the current value of this flag.

The `timeout()` function sets the timeout for the SQL query.

The `execute_direct()` function prepares and executes an SQL statement. The function returns a pointer to a data set (cursor), which can be processed using the functions of the `NANO$RSLT` package. Parameters:

- `stmt` - a pointer to the statement;
- `conn` - connection pointer;
- `query` - the text of the SQL query;
- `scrollable` - whether the cursor is scrollable (if, of course, the operator returns a cursor), if not set (NULL value), then the default behavior of the ODBC driver is used;

- `batch_operations` - the number of batch operations. The default is 1;
- `timeout` - SQL statement timeout.

The `just_execute_direct()` function prepares and executes an SQL statement. The function is designed to execute SQL statements that do not return data (do not open a cursor). Parameters:

- `stmt` - a pointer to the statement;
- `conn` - connection pointer;
- `query` - the text of the SQL query;
- `batch_operations` - the number of batch operations. The default is 1;
- `timeout` - SQL statement timeout.

The `execute_()` function executes the prepared SQL statement. The function returns a pointer to a data set (cursor), which can be processed using the functions of the `NANO$RSLT` package. Parameters:

- `stmt` - a pointer to a prepared statement;
- `batch_operations` - the number of batch operations. By default, `NANO $ STMT` is 1;
- `timeout` - SQL statement timeout.

The `just_execute()` function executes the prepared SQL statement. The function is designed to execute SQL statements that do not return data (do not open a cursor). Parameters:

- `stmt` - a pointer to a prepared statement;
- `batch_operations` - the number of batch operations. The default is 1;
- `timeout` - SQL statement timeout.

The `procedure_columns()` function—returns the description of the output field of the stored procedure as a `nano$rslt` dataset. Function parameters:

- `stmt` - a pointer to the statement;
- `catalog_` - the name of the catalog to which the SP belongs;
- `schema_` - the name of the schema in which the SP is located;
- `procedure_` - the name of the stored procedure;
- `column_` - the name of the output column of the SP.

The `affected_rows()` function returns the number of rows affected by the SQL statement. This function can be called after the statement is executed.

The `columns()` function returns the number of columns returned by the SQL query.

The `parameters()` function returns the number of SQL query parameters. This function can be called only after preparing the SQL query.

The `parameter_size()` function returns the size of the parameter in bytes.

- `stmt` - a pointer to a prepared statement;

- `parameter_index` - parameter index.

Functions of the `bind_<type> ...` family bind a value to a parameter if the DBMS supports batch operations see. `execute()` parameter `batch_operations`, then the number of transmitted values is not limited, within reasonable limits. Otherwise, only the first set of values entered is transmitted. The binding itself occurs already when you call `execute()`.

The `bind_smallint()` function binds a `SMALLINT` value to an SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_integer()` function binds an `INTEGER` value to a SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_bigint()` function binds a `BIGINT` value to a SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_float()` function binds a `FLOAT` value to an SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_double()` function binds a `DOUBLE PRECISION` value to an SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_varchar()` function binds a `VARCHAR` value to a SQL parameter. Used for single-byte encodings.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value;
- `param_size` - the size of the parameter (string).

The `bind_char()` function binds a `CHAR` value to a SQL parameter. Used for single-byte encodings.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value;
- `param_size` - the size of the parameter (string).

The `bind_u8_varchar()` function binds a `VARCHAR` value to a SQL parameter. Used for UTF8 encoded strings.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value;
- `param_size` - the size of the parameter (string).

The `bind_u8_char()` function binds a `VARCHAR` value to a SQL parameter. Used for UTF8 encoded strings.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value;
- `param_size` - the size of the parameter (string).

The `bind_blob()` function binds a `BLOB` value to an SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_boolean()` function binds a `BOOLEAN` value to an SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_date()` function binds a `DATE` value to a SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_time()` function binds a `TIME` value to an SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.



Using `bind_time()` loses milliseconds unlike `bind_timestamp()`.

The `bind_timestamp()` function binds a `TIMESTAMP` value to a SQL parameter.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `value_` - parameter value.

The `bind_null()` function binds a `NULL` value to an SQL parameter. It is not fundamentally necessary to assign a `NULL` value directly to a single value, unless it follows from the processing logic. You can also bind `NULL` by calling the corresponding function `bind_...` if `NULL` is passed to it.

- `stmt` - a pointer to a prepared statement;
- `parameter_index` - parameter index;
- `batch_size` - batch size (default 1). Allows you to set the `NULL` value for the parameter with the specified index, in several elements of the package at once.

The `convert_varchar()` function converts a `VARCHAR` value to a different encoding. Parameters:

- `value_` - string value;
- `from_` - encoding from which to recode the string;
- `to_` - encoding into which you want to recode the string;
- `convert_size` - sets the size of the input buffer for conversion (for speed), for UTF8, for example, the number of characters should be * 4. The size of the output buffer is always equal to the size of the returns declaration (you can create your own functions), the size change depends on where and from where it is converted string value: single-byte encoding to multibyte - possibly increasing relative to `convert_size` and vice versa - decreasing if multibyte encoding is converted to single-byte. The result is always truncated according to the size of the received parameter.

This is a helper function for converting strings to the desired encoding, since the other ODBC side may not always respond in the correct encoding.

The `convert_char()` function converts a `CHAR` value to a different encoding. Parameters:

- `value_` - string value;
- `from_` - encoding from which to recode the string;
- `to_` - encoding into which you want to recode the string;
- `convert_size` - sets the size of the input buffer for conversion (for speed), for UTF8, for example, the number of characters should be * 4. The size of the output buffer is always equal to the size of the returns declaration (you can create your own functions), the size change depends on where and from where it is converted string value: single-byte encoding to multibyte - possibly increasing relative to `convert_size` and vice versa - decreasing if multibyte encoding is converted to single-byte. The result is always truncated according to the size of the received parameter.

This is a helper function for converting strings to the desired encoding, since the other ODBC side may not always respond in the correct encoding.

The `clear_bindings ()` function clears the current bindings for parameters. This function call is required when reusing a prepared statement with new values.

The `describe_parameter()` function fills a buffer for describing the parameter, that is, it allows you to specify the type, size and scale of the parameter.

- `stmt` - a pointer to a prepared request;
- `idx` - parameter index;
- `type_` - parameter type;
- `size_` - size (for strings);
- `scale_` - scale.

The `describe_parameters()` function sends this parameter description buffer to ODBC, actually describes the parameters.

The `reset_parameters()` function resets the parameter information of a prepared query.

14.3.5. NANO\$RSLT package

The header of this package looks like this:

```
CREATE OR ALTER PACKAGE NANO$RSLT
AS
BEGIN

    FUNCTION valid(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;

    FUNCTION release_(rslt TY$POINTER NOT NULL) RETURNS TY$POINTER;

    FUNCTION connection(rslt TY$POINTER NOT NULL) RETURNS TY$POINTER;

    FUNCTION rowset_size(rslt TY$POINTER NOT NULL) RETURNS INTEGER;
    FUNCTION affected_rows(rslt TY$POINTER NOT NULL) RETURNS INTEGER;
    FUNCTION has_affected_rows(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;
    FUNCTION rows_(rslt TY$POINTER NOT NULL) RETURNS INTEGER;
    FUNCTION columns(rslt TY$POINTER NOT NULL) RETURNS SMALLINT;

    -----

    FUNCTION first_(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;
    FUNCTION last_(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;
    FUNCTION next_(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;
    FUNCTION prior_(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;
    FUNCTION move(rslt TY$POINTER NOT NULL, row_ INTEGER NOT NULL) RETURNS BOOLEAN;
    FUNCTION skip_(rslt TY$POINTER NOT NULL, row_ INTEGER NOT NULL) RETURNS BOOLEAN;
    FUNCTION position_(rslt TY$POINTER NOT NULL) RETURNS INTEGER;
```

```
FUNCTION at_end(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;
```

```
-----
FUNCTION get_smallint(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS SMALLINT;
```

```
FUNCTION get_integer(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS INTEGER;
```

```
/*
FUNCTION get_bigint(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS BIGINT;
*/
```

```
FUNCTION get_float(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS FLOAT;
```

```
FUNCTION get_double(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS DOUBLE PRECISION;
```

```
FUNCTION get_varchar_s(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(64) CHARACTER SET NONE;
```

```
FUNCTION get_varchar(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(256) CHARACTER SET NONE;
```

```
FUNCTION get_varchar_l(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(1024) CHARACTER SET NONE;
```

```
FUNCTION get_varchar_xl (
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(8192) CHARACTER SET NONE;
```

```
FUNCTION get_varchar_xxl (
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(32765) CHARACTER SET NONE;
```

```
FUNCTION get_char_s (
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(64) CHARACTER SET NONE;
```

```
FUNCTION get_char (
```



```

    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(256) CHARACTER SET NONE;

FUNCTION get_char_l (
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(1024) CHARACTER SET NONE;

FUNCTION get_char_xl(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(8192) CHARACTER SET NONE;

FUNCTION get_char_xxl(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(32767) CHARACTER SET NONE;

FUNCTION get_u8_varchar(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(64) CHARACTER SET UTF8;

FUNCTION get_u8_varchar_l(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(256) CHARACTER SET UTF8;

FUNCTION get_u8_varchar_xl(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(2048) CHARACTER SET UTF8;

FUNCTION get_u8_varchar_xxl(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS VARCHAR(8191) CHARACTER SET UTF8;

FUNCTION get_u8_char(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(64) CHARACTER SET UTF8;

FUNCTION get_u8_char_l(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(256) CHARACTER SET UTF8;

FUNCTION get_u8_char_xl(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(2048) CHARACTER SET UTF8;

FUNCTION get_u8_char_xxl(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS CHAR(8191) CHARACTER SET UTF8;

FUNCTION get_blob(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS BLOB;

```

```

FUNCTION get_boolean(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS BOOLEAN;

FUNCTION get_date(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS DATE;

/*
FUNCTION get_time(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS TIME;
*/

FUNCTION get_timestamp(
    rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL
) RETURNS TIMESTAMP;

FUNCTION convert_varchar_s(
    value_ VARCHAR(64) CHARACTER SET NONE,
    from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS VARCHAR(64) CHARACTER SET NONE;

FUNCTION convert_varchar(
    value_ VARCHAR(256) CHARACTER SET NONE,
    from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS VARCHAR(256) CHARACTER SET NONE;

FUNCTION convert_varchar_l(
    value_ VARCHAR(1024) CHARACTER SET NONE,
    from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS VARCHAR(1024) CHARACTER SET NONE;

FUNCTION convert_varchar_xl(
    value_ VARCHAR(8192) CHARACTER SET NONE,
    from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS VARCHAR(8192) CHARACTER SET NONE;

FUNCTION convert_varchar_xxl(
    value_ VARCHAR(32765) CHARACTER SET NONE,
    from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
    convert_size SMALLINT NOT NULL DEFAULT 0

```

```
) RETURNS VARCHAR(32765) CHARACTER SET NONE;
```

```
FUNCTION convert_char_s(
  value_ CHAR(64) CHARACTER SET NONE,
  from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS CHAR(64) CHARACTER SET NONE;
```

```
FUNCTION convert_char(
  value_ CHAR(256) CHARACTER SET NONE,
  from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS CHAR(256) CHARACTER SET NONE;
```

```
FUNCTION convert_char_l(
  value_ CHAR(1024) CHARACTER SET NONE,
  from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS CHAR(1024) CHARACTER SET NONE;
```

```
FUNCTION convert_char_xl(
  value_ CHAR(8192) CHARACTER SET NONE,
  from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS CHAR(8192) CHARACTER SET NONE;
```

```
FUNCTION convert_char_xxl(
  value_ CHAR(32767) CHARACTER SET NONE,
  from_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  to_ VARCHAR(20) CHARACTER SET NONE NOT NULL,
  convert_size SMALLINT NOT NULL DEFAULT 0
) RETURNS CHAR(32767) CHARACTER SET NONE;
```

```
-----
FUNCTION unbind(rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT
NULL)
  RETURNS TY$NANO_BLANK;
```

```
FUNCTION is_null(rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8
NOT NULL)
  RETURNS BOOLEAN;
```

```
FUNCTION is_bound( -- now hiding exception out of range
  rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8 NOT NULL)
  RETURNS BOOLEAN;
```

```

FUNCTION column_(rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET UTF8
NOT NULL)
    RETURNS SMALLINT;

FUNCTION column_name(rslt TY$POINTER NOT NULL, index_ SMALLINT NOT NULL)
    RETURNS VARCHAR(63) CHARACTER SET UTF8;

FUNCTION column_size(rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET
UTF8 NOT NULL)
    RETURNS INTEGER;

FUNCTION column_decimal_digits(rslt TY$POINTER NOT NULL, column_ VARCHAR(63)
CHARACTER SET UTF8 NOT NULL)
    RETURNS INTEGER;

FUNCTION column_datatype(rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER SET
UTF8 NOT NULL)
    RETURNS INTEGER;

FUNCTION column_datatype_name(rslt TY$POINTER NOT NULL, column_ VARCHAR(63)
CHARACTER SET UTF8 NOT NULL)
    RETURNS VARCHAR(63) CHARACTER SET UTF8;

FUNCTION column_c_datatype(rslt TY$POINTER NOT NULL, column_ VARCHAR(63) CHARACTER
SET UTF8 NOT NULL)
    RETURNS INTEGER;

FUNCTION next_result(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;

-----

FUNCTION has_data(rslt TY$POINTER NOT NULL) RETURNS BOOLEAN;

END

```

The NANODBC package contains functions for working with a dataset returned by an SQL query.

The `valid()` function returns whether the pointer to the dataset is valid.

The `release_()` function releases the pointer to the dataset.

The `connection()` function returns a pointer to a database connection.

The `rowset_size()` function returns the size of the dataset (how many active cursors are in the dataset).

The `affected_rows()` function returns the number of rows affected by the statement (fetched in the cursor).

The `has_affected_rows()` function returns whether at least one row is affected by the statement.

The `rows_()` function returns the number of records in the open cursor.

The `columns()` function returns the number of columns in the current cursor.

The `first_()` function moves the current cursor to the first record. Works only for bidirectional (scrollable cursors). Returns true if the operation is successful.

The `last_()` function moves the current cursor to the last record. Works only for bidirectional (scrollable cursors). Returns true if the operation is successful.

The `next_()` function moves the current cursor to the next record. Returns true if the operation is successful.

The `prior_()` function moves the current cursor to the previous record. Works only for bidirectional (scrollable cursors). Returns true if the operation is successful.

The `move()` function moves the current cursor to the specified record. Works only for bidirectional (scrollable cursors). Returns true if the operation is successful.

- `rslt` - a pointer to a prepared dataset;
- `row_` - record number.

The `skip_()` function moves the current cursor by the specified number of records. Works only for bidirectional (scrollable cursors). Returns true if the operation is successful.

- `rslt` - a pointer to a prepared dataset;
- `row_` - how many records to skip.

The `position_()` function returns the current position of the cursor.

The `at_end()` function returns whether the cursor has reached the last record.

The `get_smallint()` function returns the value of the SMALLINT column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_integer()` function returns the value of an INTEGER column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_bigint()` function returns the value of a BIGINT column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_float()` function returns the value of a FLOAT column.

- `rslt` - a pointer to a prepared dataset;

- `column_` - the name of the column or its number 0..n-1.

The `get_double()` function returns the value of a `DOUBLE PRECISION` column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_varchar()` function returns the value of column `VARCHAR (256) CHARACTER SET NONE`. The function is intended for single-byte encodings.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

There is a whole family of these suffixed functions. The maximum size of the returned string changes depending on the suffix:

- `_s` - `VARCHAR (64) CHARACTER SET NONE`;
- `_l` - `VARCHAR (1024) CHARACTER SET NONE`;
- `_xl` - `VARCHAR (8192) CHARACTER SET NONE`;
- `_xxl` - `VARCHAR (32765) CHARACTER SET NONE`.

The data retrieval speed depends on the maximum row size. So filling the buffer for a `VARCHAR (32765)` string is several times slower than for a `VARCHAR (256)` string, so you need to choose the size of a smaller value if you don't need a larger one.

The `get_char()` function returns the value of column `CHAR (256) CHARACTER SET NONE`. The function is intended for single-byte encodings.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

There is a whole family of these suffixed functions. The maximum size of the returned string changes depending on the suffix:

- `_s` - `CHAR (64) CHARACTER SET NONE`;
- `_l` - `CHAR (1024) CHARACTER SET NONE`;
- `_xl` - `CHAR (8192) CHARACTER SET NONE`;
- `_xxl` - `CHAR (32767) CHARACTER SET NONE`.

The data retrieval speed depends on the maximum row size. So filling the buffer for the `CHAR (32767)` string is several times slower than for the `CHAR (256)` string, so you need to choose the size of a smaller value if you don't need a larger one.

The `get_u8_varchar()` function returns the value of column `VARCHAR (64) CHARACTER SET UTF8`.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

There is a whole family of these suffixed functions. The maximum size of the returned string changes depending on the suffix:

- `_l` - VARCHAR (256) CHARACTER SET UTF8;
- `_xl` - VARCHAR (2048) CHARACTER SET UTF8;
- `_xxl` - VARCHAR (8191) CHARACTER SET UTF8.

The `get_u8_char()` function returns the value of column CHAR (64) CHARACTER SET UTF8.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

There is a whole family of these suffixed functions. The maximum size of the returned string changes depending on the suffix:

- `_l` - CHAR (256) CHARACTER SET UTF8;
- `_xl` - CHAR (2048) CHARACTER SET UTF8;
- `_xxl` - CHAR (8191) CHARACTER SET UTF8.

The `get_blob()` function returns the value of a BLOB column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_boolean()` function returns the value of a BOOLEAN column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_date()` function returns the value of a DATE column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_time()` function returns the value of a TIME column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `get_timestamp()` function returns the value of a TIMESTAMP column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `convert_varchar()` function converts a VARCHAR value to a different encoding. Parameters:

- `value_` - string value;

- `from_` - encoding from which to recode the string;
- `to_` - encoding into which you want to recode the string;
- `convert_size` - sets the size of the input buffer for conversion. See `nano$stmt.convert_[var]char`.

There is a whole family of these suffixed functions. The maximum size of the returned string changes depending on the suffix:

- `_s` - VARCHAR (64) CHARACTER SET NONE;
- `_l` - VARCHAR (1024) CHARACTER SET NONE;
- `_xl` - VARCHAR (8192) CHARACTER SET NONE;
- `_xxl` - VARCHAR (32765) CHARACTER SET NONE.

The `convert_char()` function converts a CHAR value to a different encoding. Parameters:

- `value_` - string value;
- `from_` - encoding from which to recode the string;
- `to_` - encoding into which you want to recode the string;
- `convert_size` - set the size of the input buffer for conversion. See `nano$stmt.convert_[var]char`.

There is a whole family of these suffixed functions. The maximum size of the returned string changes depending on the suffix:

- `_s` - CHAR (64) CHARACTER SET NONE;
- `_l` - CHAR (1024) CHARACTER SET NONE;
- `_xl` - CHAR (8192) CHARACTER SET NONE;
- `_xxl` - CHAR (32765) CHARACTER SET NONE.

The `unbind()` function unbinds a buffer from a given column. The peculiarity of transferring large data types in some ODBC implementations.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number $0..n-1$.

The `is_null()` function returns whether the value of a column is null.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number $0..n-1$.

The `is_bound()` function checks if a buffer of values for a given column is bound.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number $0..n-1$.

The `column_()` function returns the number of a column by its name.

- `rslt` - a pointer to a prepared dataset;

- `column_` is the name of the column.

The `column_name()` function returns the name of a column by its index.

- `rslt` - a pointer to a prepared dataset;
- `index_` - column number 0..n-1.

The `column_size()` function returns the size of a column. For string fields, the number of characters.

The `column_decimal_digits()` function returns the precision for numeric types.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `column_datatype()` function returns the type of the column.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `column_datatype_name()` function returns the name of the column type.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `column_c_datatype()` function returns the type of the column as encoded in ODBC constants.

- `rslt` - a pointer to a prepared dataset;
- `column_` - the name of the column or its number 0..n-1.

The `next_result()` function switches to the next data set.

- `rslt` - a pointer to a prepared dataset.

The `has_data()` function returns whether there is data in a dataset.

- `rslt` - a pointer to a prepared dataset.

14.3.6. NANO\$FUNC package

The header of this package looks like this:

```

CREATE OR ALTER PACKAGE NANO$FUNC
AS
BEGIN

/* Note:
   Result cursor by default ODBC driver (NANODBC implementation),
   scrollable into NANO$STMT
*/

FUNCTION execute_conn(
  conn TY$POINTER NOT NULL,
  query VARCHAR(8191) CHARACTER SET UTF8 NOT NULL,
  batch_operations INTEGER NOT NULL DEFAULT 1,
  timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$POINTER;

FUNCTION just_execute_conn(
  conn TY$POINTER NOT NULL,
  query VARCHAR(8191) CHARACTER SET UTF8 NOT NULL,
  batch_operations INTEGER NOT NULL DEFAULT 1,
  timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

FUNCTION execute_stmt(
  stmt TY$POINTER NOT NULL, batch_operations INTEGER NOT NULL DEFAULT 1
) RETURNS TY$POINTER;

FUNCTION just_execute_stmt(
  stmt TY$POINTER NOT NULL, batch_operations INTEGER NOT NULL DEFAULT 1
) RETURNS TY$NANO_BLANK;

FUNCTION transact_stmt(
  stmt TY$POINTER NOT NULL, batch_operations INTEGER NOT NULL DEFAULT 1
) RETURNS TY$POINTER;

FUNCTION just_transact_stmt(
  stmt TY$POINTER NOT NULL, batch_operations INTEGER NOT NULL DEFAULT 1
) RETURNS TY$NANO_BLANK;

FUNCTION prepare_stmt(
  stmt TY$POINTER NOT NULL,
  query VARCHAR(8191) CHARACTER SET UTF8 NOT NULL,
  timeout INTEGER NOT NULL DEFAULT 0
) RETURNS TY$NANO_BLANK;

END

```

The NANO\$FUNC package contains functions for working with SQL queries. This package is a lightweight version of the NANO\$STMT package. The peculiarity is that the functions performed have

inherited the behavior of NANODBC without changes and their own modifications of the UDR in terms of the exchange of parameters and values. Possible direction of use: performing ODBC connection settings through executing SQL commands (`just_execute ...`), if supported, event logging, etc. simple operations.

The `execute_conn()` function prepares and executes an SQL statement. The function returns a pointer to a data set (cursor), which can be processed using the functions of the `NANO$RSLT` package. Parameters:

- `conn` - connection pointer;
- `query` - the text of the SQL query;
- `batch_operations` - the number of batch operations. The default is 1;
- `timeout` - SQL statement timeout.

The `just_execute_conn()` function prepares and executes the SQL statement. The function is designed to execute SQL statements that do not return data (do not open a cursor). A pointer to the SQL query object is not created. Parameters:

- `conn` - connection pointer;
- `query` - the text of the SQL query;
- `batch_operations` - the number of batch operations. The default is 1;
- `timeout` - SQL statement timeout.

The `execute_stmt()` function executes the prepared SQL statement. The function returns a pointer to a data set (cursor), which can be processed using the functions of the `NANO$RSLT` package. Parameters:

- `stmt` - a pointer to a prepared statement;
- `batch_operations` - the number of batch operations. The default is 1.

The `transact_stmt()` function - executes a previously prepared SQL statement, starting and ending its own (autonomous) transaction. The function returns a pointer to a data set (cursor), which can be processed using the functions of the `NANO$RSLT` package. Parameters:

- `stmt` - a pointer to a prepared statement;
- `batch_operations` - the number of batch operations. The default is 1.

Function `just_transact_stmt()` - executes a previously prepared SQL statement, starting and ending its own (autonomous) transaction. The function is designed to execute SQL statements that do not return data (do not open a cursor). Parameters:

- `stmt` - a pointer to a prepared statement;
- `batch_operations` - the number of batch operations. The default is 1.

The `prepare_stmt()` function prepares the SQL query. Parameters:

- `stmt` - a pointer to the statement;

- query - the text of the SQL query;
- timeout - SQL statement timeout.

14.4. Examples

14.4.1. Fetching data from a Postgresql table

This example fetches from a Postgresql database. The block text is provided with comments to understand what is happening.

```
EXECUTE BLOCK
RETURNS (
  id bigint,
  name VARCHAR(1024) CHARACTER SET UTF8
)
AS
DECLARE conn_str varchar(512) CHARACTER SET UTF8;
declare variable sql_txt VARCHAR(8191) CHARACTER SET UTF8;
DECLARE conn ty$pointer;
DECLARE stmt ty$pointer;
DECLARE rs ty$pointer;
DECLARE tnx ty$pointer;
BEGIN
  conn_str = 'DRIVER={PostgreSQL ODBC
Driver(UNICODE)};SERVER=localhost;DATABASE=test;UID=postgres;PASSWORD=mypassword';
  sql_txt = 'select * from t1';

  -- initialize nanodbc
  -- this function can be called in the ON CONNECT trigger
  nano$udr.initialize();

BEGIN
  -- connect to ODBC data source
  conn = nano$conn.connection(conn_str);
  WHEN EXCEPTION nano$nanodbc_error DO
  BEGIN
    -- if the connection was unsuccessful
    -- call the function to terminate nanodbc
    -- instead of an explicit call in the script, this function can be called
    -- in the ON DISCONNECT trigger
    nano$udr.finalize();
    -- rethrow exception
  EXCEPTION;
  END
END

BEGIN
  -- allocate a pointer to an SQL statement
  stmt = nano$stmt.statement_(conn);
```

```

-- prepare query
nano$stmt.prepare_(stmt, sql_txt);
-- execute query
-- function returns a pointer to a dataset
rs = nano$stmt.execute_(stmt);
-- while there are records in the cursor, move forward along it
while (nano$rslt.next_(rs)) do
begin
  -- for each column, depending on its type, it is necessary to call
  -- the corresponding function or function with the type to which
  -- the initial column can be converted
  id = nano$rslt.get_integer(rs, 'id');
  -- note, since we are working with UTF8, the function is called with u8
  name = nano$rslt.get_u8_char_l(rs, 'name');
  suspend;
end

-- release the previously allocated resource
/*
rs = nano$rslt.release_(rs);
stmt = nano$stmt.release_(stmt);
*/
-- the above functions can be omitted, since calling
-- nano$conn.release_ will automatically release all resources
-- bound to the connection
conn = nano$conn.release_(conn);
-- call the function to terminate nanodbc
-- instead of an explicit call in the script, this function can be called
-- in the ON DISCONNECT trigger
nano$udr.finalize();

WHEN EXCEPTION nano$invalid_resource,
      EXCEPTION nano$nanodbc_error,
      EXCEPTION nano$binding_error
DO
BEGIN
  -- if an error occurs
  -- release previously allocated resources
  /*
  rs = nano$rslt.release_(rs);
  stmt = nano$stmt.release_(stmt);
  */
  -- the above functions can be omitted, since calling
  -- nano$conn.release_ will automatically release all resources
  -- bound to the connection
  conn = nano$conn.release_(conn);
  -- call the function to terminate nanodbc
  -- instead of an explicit call in the script, this function can be called
  -- in the ON DISCONNECT trigger
  nano$udr.finalize();
  -- rethrow exception

```

```

    EXCEPTION;
  END
END
END

```

14.4.2. Inserting data into a Postgresql table

This example inserts a new row into a table. The block text is provided with comments to understand what is happening.

```

EXECUTE BLOCK
RETURNS (
  aff_rows integer
)
AS
  DECLARE conn_str varchar(512) CHARACTER SET UTF8;
  declare variable sql_txt VARCHAR(8191) CHARACTER SET UTF8;
  DECLARE conn ty$pointer;
  DECLARE stmt ty$pointer;
  DECLARE txn ty$pointer;
BEGIN
  conn_str = 'DRIVER={PostgreSQL ODBC
Driver(UNICODE)};SERVER=localhost;DATABASE=test;UID=postgres;PASSWORD=mypassword';
  sql_txt = 'insert into t1(id, name) values(?, ?)';

  -- initialize nanodbc
  -- this function can be called in the ON CONNECT trigger
  nano$udr.initialize();

BEGIN
  -- connect to ODBC data source
  conn = nano$conn.connection(conn_str);
  WHEN EXCEPTION nano$nanodbc_error DO
  BEGIN
    -- if the connection was unsuccessful
    -- call the function to terminate nanodbc
    -- instead of an explicit call in the script, this function can be called
    -- in the ON DISCONNECT trigger
    nano$udr.finalize();
  EXCEPTION;
  END
END

BEGIN
  -- allocate a pointer to an SQL statement
  stmt = nano$stmt.statement_(conn);
  -- prepare query
  nano$stmt.prepare_(stmt, sql_txt);
  -- set query parameters
  -- index starts from 0!

```

```

nano$stmt.bind_integer(stmt, 0, 4);
nano$stmt.bind_u8_varchar(stmt, 1, 'Row 4', 4 * 20);
-- execute INSERT statement
nano$stmt.just_execute(stmt);
-- get the number of affected rows
aff_rows = nano$stmt.affected_rows(stmt);
-- release the previously allocated resource
conn = nano$conn.release_(conn);
-- call the function to terminate nanodbc
-- instead of an explicit call in the script, this function can be called
-- in the ON DISCONNECT trigger
nano$udr.finalize();

WHEN EXCEPTION nano$invalid_resource,
      EXCEPTION nano$nanodbc_error,
      EXCEPTION nano$binding_error
DO
BEGIN
  -- release the previously allocated resource
  conn = nano$conn.release_(conn);
  -- call the function to terminate nanodbc
  -- instead of an explicit call in the script, this function can be called
  -- in the ON DISCONNECT trigger
  nano$udr.finalize();
EXCEPTION;
END
END

suspend;
END

```

14.4.3. Batch insert into a Postgresql table

If the DBMS and ODBC driver support batch execution of queries, then batch operations can be used.

```

EXECUTE BLOCK
AS
  DECLARE conn_str varchar(512) CHARACTER SET UTF8;
  declare variable sql_txt VARCHAR(8191) CHARACTER SET UTF8;
  DECLARE conn ty$pointer;
  DECLARE stmt ty$pointer;
  DECLARE tnx ty$pointer;
BEGIN
  conn_str = 'DRIVER={PostgreSQL ODBC
Driver(UNICODE)};SERVER=localhost;DATABASE=test;UID=postgres;PASSWORD=mypassword';
  sql_txt = 'insert into t1(id, name) values(?, ?)';

  -- initialize nanodbc
  -- this function can be called in the ON CONNECT trigger

```

```

nano$udr.initialize();

BEGIN
  -- connect to ODBC data source
  conn = nano$conn.connection(conn_str);
  WHEN EXCEPTION nano$nanodbc_error DO
  BEGIN
    -- if the connection was unsuccessful
    -- call the function to terminate nanodbc
    -- instead of an explicit call in the script, this function can be called
    -- in the ON DISCONNECT trigger
    nano$udr.finalize();
  EXCEPTION;
  END
END

BEGIN
  -- allocate a pointer to an SQL statement
  stmt = nano$stmt.statement_(conn);
  -- prepare query
  nano$stmt.prepare_(stmt, sql_txt);
  -- set query parameters
  -- index starts from 0!
  -- first row
  nano$stmt.bind_integer(stmt, 0, 5);
  nano$stmt.bind_u8_varchar(stmt, 1, 'Row 5', 4 * 20);
  -- second row
  nano$stmt.bind_integer(stmt, 0, 6);
  nano$stmt.bind_u8_varchar(stmt, 1, 'Row 6', 4 * 20);
  -- execute an INSERT statement with a batch size of 2
  nano$stmt.just_execute(stmt, 2);
  -- release the previously allocated resource
  conn = nano$conn.release_(conn);
  -- call the function to terminate nanodbc
  -- instead of an explicit call in the script, this function can be called
  -- in the ON DISCONNECT trigger
  nano$udr.finalize();

  WHEN EXCEPTION nano$invalid_resource,
    EXCEPTION nano$nanodbc_error,
    EXCEPTION nano$binding_error
  DO
  BEGIN
    -- release the previously allocated resource
    conn = nano$conn.release_(conn);
    -- call the function to terminate nanodbc
    -- instead of an explicit call in the script, this function can be called
    -- in the ON DISCONNECT trigger
    nano$udr.finalize();
  EXCEPTION;
  END

```



```
END
END
```

14.4.4. Using transaction

```
EXECUTE BLOCK
AS
  DECLARE conn_str varchar(512) CHARACTER SET UTF8;
  DECLARE sql_txt VARCHAR(8191) CHARACTER SET UTF8;
  DECLARE sql_txt2 VARCHAR(8191) CHARACTER SET UTF8;
  DECLARE conn ty$pointer;
  DECLARE stmt ty$pointer;
  DECLARE stmt2 ty$pointer;
  DECLARE txn ty$pointer;
BEGIN
  conn_str = 'DRIVER={PostgreSQL ODBC
Driver(UNICODE)};SERVER=localhost;DATABASE=test;UID=postgres;PASSWORD=myspassword';
  sql_txt = 'insert into t1(id, name) values(?, ?)';
  sql_txt2 = 'insert into t2(id, name) values(?, ?)';

  -- initialize nanodbc
  -- this function can be called in the ON CONNECT triggerpe
  nano$udr.initialize();

BEGIN
  -- connect to ODBC data source
  conn = nano$conn.connection(conn_str);
  WHEN EXCEPTION nano$nanodbc_error DO
  BEGIN
    -- if the connection was unsuccessful
    -- call the function to terminate nanodbc
    -- instead of an explicit call in the script, this function can be called
    -- in the ON DISCONNECT trigger
    nano$udr.finalize();
  EXCEPTION;
  END
END

BEGIN
  -- prepare first SQL query
  stmt = nano$stmt.statement_(conn);
  nano$stmt.prepare_(stmt, sql_txt);
  -- prepare second SQL query
  stmt2 = nano$stmt.statement_(conn);
  nano$stmt.prepare_(stmt2, sql_txt2);
  -- start transaction
  txn = nano$txn.transaction_(conn);
  -- execute first statement within the transaction
  nano$stmt.bind_integer(stmt, 0, 8);
  nano$stmt.bind_u8_varchar(stmt, 1, 'Row 8', 4 * 20);
```

```

nano$stmt.just_execute(stmt);
-- execute second statement within the transaction
nano$stmt.bind_integer(stmt2, 0, 1);
nano$stmt.bind_u8_varchar(stmt2, 1, 'Row 1', 4 * 20);
nano$stmt.just_execute(stmt2);
-- commit transaction
nano$tnx.commit_(tnx);

-- release the previously allocated resource
conn = nano$conn.release_(conn);
-- call the function to terminate nanodbc
-- instead of an explicit call in the script, this function can be called
-- in the ON DISCONNECT trigger
nano$udr.finalize();

WHEN EXCEPTION nano$invalid_resource,
      EXCEPTION nano$nanodbc_error,
      EXCEPTION nano$binding_error
DO
BEGIN
  -- release the previously allocated resource
  -- in case of an error, the unconfirmed transaction will be rolled back
  automatically
  conn = nano$conn.release_(conn);
  -- call the function to terminate nanodbc
  -- instead of an explicit call in the script, this function can be called
  -- in the ON DISCONNECT trigger
  nano$udr.finalize();
EXCEPTION;
END
END
END

```

Chapter 15. Firebird Streaming

The Firebird Streaming project is a set of libraries for parsing Firebird replication logs after being processed by the `fb_repl_print` utility. For each event in the log, you can write your own handler by inheriting the `SegmentProcessEventListener` interface.

The source code of project can be downloaded at <https://github.com/sim1984/fbstreaming>

This library can be used to notify about events through the queue system, or to write your own replicator in other DBMS.

The release can be downloaded from the link <https://github.com/sim1984/fbstreaming/releases/download/v1.0/release.zip>

The `journals/incoming` folder contains replication logs after being processed by the `fb_repl_print` utility.

The `journals/outgoing` folder some output files that are the output of one of the plugins.

The `journals/segments.journal` file is a file in which the file names of the processed segments are written. Used by some plugins.

JAR files containing plugins are located in the `plugins` folder. The example contains the following plugins:

- converting replication segments to JSON format
- sending the associated DML statement data (in JSON format) to the RabbitMQ queue on a transaction confirmation event
- writing DML statements to SQL files. Recording occurs on the event of a transaction confirmation.

For configuration, the `config.properties` file is used, which must be located next to the `fbstreaming-1.0.jar` file. Properties in the config file:

- `pluginClassName` — fully qualified plugin class name
- `incomingFolder` — folder with input logs (replication segments)
- `outgoingFolder` — folder with plugin results (used by JSON and SQL plugins)
- `journalFileName` — file with processed replication segments (used by SQL and RabbitMQ plugins)
- `segmentFileNameMask` — mask for input files (replication segments)
- `segmentFileCharset` — encoding of replication segments
- `includeTables` — regular expression for filtering tables (if not specified, all tables are processed)
- `rabbit.host` — host for RabbitMQ plugin
- `rabbit.queueName` — the name of the queue for the RabbitMQ plugin

To be able to process a replication segment, it must be processed with the `fb_repl_print` utility using the following command:

```
fb_repl_print -d -b <archive_journal_file> > <journal_file_for_streaming>
```

Example

```
fb_repl_print -d -b f:\fbdata\archives\examples.fdb.arch-000000010 >  
f:\journals\incoming\examples.fdb.arch-000000010.txt
```

To start the handler of received files, enter the command:

```
java -jar fbstreaming-1.0.jar
```

15.1. Json plugin description

Full name of plugin `com.hqbird.fbstreaming.plugin.json.JsonStreamPlugin`.

The `JsonStreamPlugin` plugin reads replication segment files and stores DML statement data over tables in JSON format. A JSON data file is created for each replication segment file. JSON files are saved in the folder specified in the `outgoingFolder` parameter.

The JSON file has the following format:

```
[
  {
    "transactionNumber": 409036118,
    "statements": [
      {
        "tableName": "CLBULLETS",
        "statementType": "INSERT",
        "keyValues": {
          "NUMBULL": 156803509
        },
        "newFieldValues": {
          "CLAGE": 45,
          "REPL$GRPID": 15,
          "PROFID": 990000090,
          "SENDSTATUS": 0,
          "POL": 1,
          "FILIAL": 15,
          "BDATE": "1976-05-03",
          "CLINICID": 38,
          "PRIMLIST": 1,
          "MODIFYDATE": "16-AUG-2021 23:50:14.2480",
          "DGOPEN": 14438,
          "DATEOPEN": "16-AUG-2021",
          "TREATCODE": 156802708,
          "PCODE": 150002994,
          "ISSUEDATE": "16-AUG-2021",
          "DISABILITYID": 150050614,
          "DCODEOPEN": 150000278,
          "NUMBULL": 156443509
        },
        "oldFieldValues": {}
      }
    ]
  }
]
```

The JSON file at the top level is an array of transactions. For each transaction, its number is stored in the `transactionNumber` key. An array with a description of all DML statements on tables produced within this transaction is stored in the `statements` key. The following data is saved for each DML statement:

- `tableName` — the name of the table on which the operation is performed;
- `statementType` — the type of the statement (INSERT, UPDATE, DELETE);
- `keyValues` — values of key fields;
- `newFieldValues` — new field values;
- `oldFieldValues` — old field values.

Example configuration file `config.properties`:

```

pluginClassName=com.hqbird.fbstreaming.plugin.json.JsonStreamPlugin
incomingFolder=./journals/incoming
outgoingFolder=./journals/outgoing/json
segmentFileNameMask=*.txt
segmentFileCharset=windows-1251
includeTables=CLBULLETS|CLREFDET

```

15.2. Sql plugin description

Full name of plugin `com.hqbird.fbstreaming.plugin.sql.SqlStreamPlugin`.

The `SqlStreamPlugin` reads the replication segment files and saves them to files with DML statements as SQL. For each transaction from the replication segment, a file with SQL statements is created. SQL files are saved in the folder specified in the `outgoingFolder` parameter.

The file stores script DML statements separated by semicolons ";". BLOB of subtype TEXT is converted to character literal, if subtype is BINARY, then it is converted to binary literal in hexadecimal notation. Attention, for BLOBs longer than 65535 bytes, an error script will be generated. This is planned to be fixed in the future.

Example configuration file `config.properties`:

```

pluginClassName=com.hqbird.fbstreaming.plugin.sql.SqlStreamPlugin
incomingFolder=./journals/incoming
outgoingFolder=./journals/outgoing/sql
journalFileName=./journals/segments.journal
segmentFileNameMask=*.txt
segmentFileCharset=windows-1251
includeTables=CLBULLETS|CLREFDET

```

15.3. Rabbitmq plugin description

Full name of plugin `com.hqbird.fbstreaming.plugin.rabbitmq.RabbitMQStreamPlugin`.

The `RabbitMQStreamPlugin` plugin reads replication segment files and stores DML statement data over tables in JSON format. When a transaction commit event is detected, JSON data is sent to the RabbitMQ queue. The JSON message format is similar to the one described above.

Example configuration file `config.properties`:

```
pluginClassName=com.hqbird.fbstreaming.plugin.rabbitmq.RabbitMQStreamPlugin
incomingFolder=./journals/incoming
journalFileName=./journals/segments.journal
segmentFileNameMask=.*txt
segmentFileCharset=windows-1251
includeTables=CLBULLETS|CLREFDET
rabbit.host=localhost
rabbit.queueName=hello
```

The delivery contains the simplest example of a client that reads a message from the RabbitMQ queue: `RabbitMQReceiver-1.0.jar`.

The client is launched with the command:

```
java -jar RabbitMQReceiver-1.0.jar
```

Appendix A: Support contacts

We will answer all your questions regarding HQbird FBDataGuard. Please send all your inquiries to support@ib-aid.com

Please note, that customers with active Firebird Support have the priority in the technical support <https://ib-aid.com/en/firebird-support-service/>