
# Руководство пользователя HQbird 2024

1.0.3, от 24.09.2024

# Содержание

# Введение

## Об этом руководстве

Руководство пользователя HQbird содержит подробное описание функций и возможностей HQbird — расширенного дистрибутива Firebird, в том числе с примерами конфигурации и рекомендациями по передовому опыту.

## О IBSurgeon

Компания IBSurgeon (https://www.ib-aid.com) была основана в 2002 году с целью предоставить разработчикам и администраторам InterBase и Firebird услуги и инструменты, ориентированные на безопасность, производительность и доступность баз данных. В России IBSurgeon в основном известен как iBase.ru, известный своими российским порталом по InterBase и Firebird www.ibase.ru. IBSurgeon является членом Firebird Foundation и, как член технической рабочей группы, имеет прочные отношения с проектом Firebird, с прямыми представителями в Firebird-Admins и в комитете Firebird Foundation.

Сегодня IBSurgeon обслуживает тысячи компаний по всему миру, предлагая инструменты для экстренной помощи, оптимизации и обслуживания, а также различные услуги. Нашими клиентами являются медицинские учреждения, финансовые организации и независимые поставщики программного обеспечения в Германии, Бразилии, России и других странах, а также все, у кого есть приложения на основе Firebird и/или InterBase. Флагманским проектом IBSurgeon является HQbird, расширенный дистрибутив FirebirdSQL для больших баз данных с корпоративными функциями.

# Глава 1. Обзор HQbird

## 1.1. Что такое HQbird

HQbird — дистрибутив СУБД Firebird для предприятий от IBSurgeon Software (www.ib-aid.com[https://www.ib-aid.com]), включающий в себя дополнительные функции, преимущественно повышающие производительность для больших и высоконагруженных баз данных, а также набор инструментов для организации полного цикла обслуживания баз данных без администратор базы данных (включая инструменты для оптимизации производительности, мониторинга, локального и облачного резервного копирования, а также восстановления в случае сбоев).

HQbird ускоряет работу больших баз данных (от 50 ГБ до 2 ТБ) и позволяет компаниям управлять большими базами данных, не требуя выделенного администратора баз данных для нескольких серверов или снижая расходы на поддержку многих (сотни и тысячи) серверов Firebird.

Минимальные аппаратные требования для HQbird — 8 ГБ ОЗУ и 4 ядра. Первая версия HQbird была выпущена в 2015 году, текущая версия — HQbird 2024.

## 1.2. Как Firebird связан с HQbird?

Проще говоря, HQbird — это корпоративная версия СУБД Firebird с открытым исходным кодом. Следуя традиции open-source проектов, мы называем Firebird “ванильной” версией: точно так же существует “ванильная” версия PostgreSQL и коммерческие версии EnterpriseDB, PostgresPro и т. д.

HQbird не является “другой” базой данных с точки зрения совместимости с Firebird: нет необходимости делать резервное копирование-восстановление при переключении между HQbird и Firebird, не нужно переписывать SQL или менять клиентские приложения.

Без каких-либо проблем вы можете установить HQbird и ванильную версию Firebird параллельно на одном сервере, работать с файлом базы данных с помощью HQbird, затем переключиться на ванильный Firebird, и наоборот.

100% совместимость HQbird с ванильным Firebird — самая важная особенность HQbird!

Практически все функции, разработанные для HQbird, попадают в Firebird в пределах 1-2 версий: например, репликация появилась в HQbird версии 2.5, а в ванильном Firebird — в версии 4.0, пул внешних соединений был разработан в HQbird 3.0 и появился в Firebird 4.0, возможности многопоточного резервного копирования, восстановления и sweep появились начиная с HQbird 2.5, и стали доступны в ванильном Firebird 5.0 и т. д.

Помимо нового функционала, ошибки, исправленные в HQbird, также исправлены в соответствующих версиях ванильного Firebird.

Также компания IBSurgeon Software проводит публичное тестирование Firebird и HQbird на предмет надежности и производительности: результаты тестирования публикуются на

сайте www.firebirdtest.com.

## 1.3. Какова цена HQbird?

Постоянная лицензия на 1 сервер стоит 899 долларов США. В комплект также входит 1 лицензия на сервер реплик.

Какова будет цена HQbird для компании, которой необходимо использовать не 1 или 2 сервера, а несколько сотен или даже тысяч установок? Если количество серверов превышает 20, покупка постоянных лицензий (899 долларов США/сервер) становится слишком дорогой. Вот почему мы предлагаем безлимитную подписку HQbird для компаний-разработчиков программного обеспечения.

Неограниченная подписка HQbird для компаний-разработчиков программного обеспечения стоит 1200 долларов США в месяц (или 13450 долларов США в год с авансовым платежом) за годовой контракт (обратите внимание, что региональные цены могут отличаться).

Это специальная лицензия для компаний-разработчиков программного обеспечения, которая позволяет устанавливать и использовать неограниченное количество копий HQbird вместе с бизнес-приложениями (ERP, CRM и т. д.), производимыми компанией.

Например, если у компании 40 клиентов, то подписка будет стоить 1200 долларов США в месяц, то есть примерно 30 долларов США на одного клиента в месяц. Если у компании-разработчика программного обеспечения 400 клиентов, то стоимость 1 клиента в месяц составит 3 доллара в месяц.

## 1.4. Что нового в HQbird 2024

HQbird 2024 — это новая крупная версия, в которой добавлена поддержка Firebird 5.0 и ряд важных функций:

- Замена запросов "на лету"
- Улучшена технология Firebird Streaming:
  - Добавлен контрольный файл, с тем же форматом, который используется для репликации в Firebird 4 и выше
  - Добавлено восстановление службы `fb_streaming` после сбоя (сегменты репликации не удаляются до тех пор пока транзакции начатые в них не будут завершены или откачены. После сбоя происходит повтор транзакций, которые не успели зафиксироваться).
  - Добавлен новый плагин Kafka CDC (Change Data Capture)

Кроме того, HQbird 2024 по-прежнему предлагает расширенные функции репликации, пул внешних подключений, пул подготовленных операторов и другие функции, необходимые при работе с большими базами данных под высокой нагрузкой.

# 1.5. Матрица функций

Ниже представлена матрица функций и уровень их поддержки в различных версиях Firebird, поставляемых в составе HQBird.

| № | Функция | V5.0 | V4.0 | V3.0 | V2.5 | Уровень |
|---|---|---|---|---|---|---|
| 1 | Высокопроизводительная встроенная репликация | X | X | X | X | Сервер |
| 2 | Замена запросов "на лету" | X | X | X | X | Сервер |
| 3 | Плагины для доступа к ODBC/MySQL через оператор `EXECUTE STATEMENT ON EXTERNAL` | X | X | X | | Плагин |
| 4 | Кэширование BLOB во временном пространстве | X | X | X | X | Сервер |
| 5 | Улучшения в оптимизаторе для JOIN и больших сортировок | X | X | X | | Сервер |
| 6 | Кеш подготовленных запросов | X | X | X | | Сервер |
| 7 | Firebird Streaming (плагины Kafka CDC, RabbitMQ и др.) | X | X | | | Плагин |
| 8 | Полнотекстовый поиск | X | X | X | | Плагин |
| 9 | Многопоточный backup/restore/sweep, создание индексов | X | X | X | X | Сервер |
| 10 | Параллельное чтение согласованных данных | X | X | X | X | Сервер |
| 11 | Пул внешних соединений | X | X | X | | Сервер |
| 12 | Шифрование | X | X | X | | Плагин |
| 13 | Автоматическая коррекция `firebird.conf` ( `DefaultDbCachePages`) | X | X | X | X | Сервер |
| 14 | Расширенный мониторинг производительности (трассировка, MON, блокировки, ЦП, ОЗУ, диск) | X | X | X | X | Инструмент |
| 15 | Мониторинг запросов с большими сортировками | X | X | X | X | Сервер |
| 16 | Регистрация и настройка репликации для множества баз данных с помощью инструментов командной строки. | X | X | X | X | Инструмент |

| № | Функция | V5.0 | V4.0 | V3.0 | V2.5 | Уровень |
|---|---|---|---|---|---|---|
| 17 | Автоматизация backup,restore и backup/restore | X | X | X | X | Инструмент |
| 18 | Передача резервных копий, сегментов репликации через FTP в облако | X | X | X | X | Инструмент |
| 19 | Расширенное обслуживание базы данных: правильная сборка мусора и многое другое. | X | X | X | X | Инструмент |
| 20 | Поддержка множества экземпляров Firebird | X | X | X | X | Инструмент |
| 21 | Тихая установка в Window и Linux | X | X | X | X | Инструмент |
| 22 | Инструменты для анализа статистики | X | X | X | X | Инструмент |
| 23 | Инструмент для анализа соединений/транзакций/потребления памяти/операций ввода-вывода | X | X | X | X | Инструмент |
| 24 | Инструменты починки БД | X | X | X | X | Инструмент |
| 25 | Оптимизированные конфигурации | X | X | X | X | Инструмент |

# 1.6. Краткое описание функций HQBird

## 1.6.1. Высокопроизводительная встроенная репликация

HQbird включает встроенную репликацию для создания отказоустойчивых систем на базе баз данных Firebird:

- Репликация баз данных с более чем 1500 подключениями.
- Асинхронная репликация с задержкой 1-30 секунд,
- Синхронная репликация без задержек,
- Никаких триггеров или других изменений в схеме не требуется.
- Автоматическое распространение изменений DDL,
- Онлайн-реинициализация реплик.
- Встроенные инструменты для передачи изменений репликации, проверка переданных сегментов репликации.

Нативная репликация настраивается через специальный плагин, с возможностью исключения записей без PK/UK на уровне плагина.

HQbird имеет полный инструментарий для организации передачи сегментов для асинхронной репликации для схем “1-к-1” или “1-ко-многим”, с автоматической настройкой, передачей и проверкой сегментов репликации через сокеты или FTP. В HQbird есть инструменты командной строки для настройки баз данных для массовой репликации, выбора баз данных в папке или во вложенных папках.

## 1.6.2. Замена запросов "на лету"

Если у вас есть приложение с недоступными или отсутствующими исходными кодами, то HQbird может помочь вам изменить тексты несовместимых или наиболее ресурсоемких SQL-запросов “на лету”, а значит, помочь оптимизировать производительность или произвести миграцию приложения на новую версию Firebird. Замена легко настраивается, и реализуется парами файлов, содержащих текст исходного и заменяемого запросов.

С помощью расширенного мониторинга вы можете найти SQL-запросы, вызывающие проблемы, а затем настроить их замену, даже без доступа к исходному коду приложения. Замененный запрос появится в трассировке и MON$ таблицах с новым текстом.

## 1.6.3. Плагины для работы с внешними базами данных MySQL или через ODBC

HQbird имеет плагины внешних источников данных для ODBC и MySQL. Используя эти плагины, можно выполнять команды `EXECUTE STATEMENT ON EXTERNAL` с запросами к источнику данных MySQL или ODBC, чтобы читать данные из внешних источников данных или записывать данные во внешние источники данных.

Плагины поддерживают входные параметры и корректное отображение типов данных

(однако в случае ODBC это зависит от конкретной реализации драйвера).

Пример внешнего подключения ниже:

```
execute block
returns (
  emp_no bigint,
  birth_date date,
  first_name varchar(14),
  last_name varchar(16),
  gender char(1),
  hire_date date
)
as
  declare dsn_mysql varchar(128);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';
  for
    execute statement q'{
select
  emp_no,
  birth_date,
  first_name,
  last_name,
  gender,
  hire_date
from employees
order by birth_date desc limit 5
}'
    on external dsn_mysql
    as user null password 'sa'
    into
      emp_no, birth_date, first_name,
      last_name, gender, hire_date
  do
    suspend;
end
```

Подробнее см. Работа с внешними источниками данных (другие СУБД)

### 1.6.4. Кэширование BLOB во временном пространстве

HQbird может кэшировать BLOB-объекты во временном пространстве, чтобы ускорить операции с BLOB (на +15%-200% быстрее, чем в обычном Firebird) и предотвратить рост файла базы данных в случае ошибочных операций с BLOB.

HQbird использует дополнительный параметр `BlobTempSpace` в `firebird.conf` для управления этой функцией.

Вариант кэширования может быть:

- 0 — отключено,
- 1 — включено для PSQL (по умолчанию),
- 2 —- включено для всех операций с BLOB.

### 1.6.5. Улучшения в оптимизаторе для JOIN и больших сортировок

#### LeftJoinConversion / OuterLeftConversion

HQbird может автоматически преобразовывать неявные внутренние соединения в явные для лучшей оптимизации в версиях 3 и 4.

Чтобы активировать эту функцию, измените настройку `LeftJoinConversion` в `firebird.conf` на `true`. HQbird в версии 5.0 поддерживает опцию `OuterLeftConversion`, доступную в стандартной версии Firebird 5.0.

#### SortDataStorageThreshold / InlineSortThreshold

HQbird может оптимизировать запросы, включающие большие операции сортировки. В версиях 2.5 и 3.0 вы можете использовать параметр SortDataStorageThreshold, для активации метода доступа `Refetch`.

В “ванильной” версии Firebird 4.0 этот параметр переименован в `InlineSortThreshold`. Обычно мы рекомендуем установить для `SortDataStorageThreshold` значение 8192 или 16384 байта.

### 1.6.6. Кеш подготовленных запросов

Эта функция может повысить производительность повторяющихся запросов, особенно при использовании пула соединений (PHP и т. д.).

Кэш хранит определенное количество подготовленных запросов в памяти каждого соединения. HQbird имеет этот кеш в версиях 3.0 и 4.0, и вы можете настроить его с помощью параметра `DSQLCacheSize` (по умолчанию — 0, т. е. отключено).

В “ванильной” версии Firebird 5.0 есть аналогичная функция, регулируемая опцией `MaxCompiledCache`, которая измеряется в мегабайтах, по умолчанию — 2 МБ.

### 1.6.7. Firebird Streaming (плагины Kafka CDC, RabbitMQ и др.)

Firebird Streaming — это технология, которая отслеживает изменения в базе данных и отправляет их в другую систему, например Kafka, файлы JSON, RabbitMQ, плагин полнотекстового поиска и т. д.

HQbird предлагает плагин для сбора измененных данных (Change Data Capture), основанный на репликации. Плагин создает поток изменений, отражающий фиксации/откаты транзакций.

HQbird предоставляет готовые плагины для Kafka, RabbitMQ, JSON файлов, а также поддерживает их настройку для любого места назначения. CDC полезен для обработки

очередей, асинхронной отправки предупреждений и копирования изменений в другие системы (например, конвейеры бизнес-аналитики или обработки данных).

Плагин CDC поставляется по запросу. Для получения дополнительной информации обратитесь в службу поддержки IBSurgeon (support@ib-aid.com).

Подробнее см. Firebird Streaming

### 1.6.8. Полнотекстовый поиск

Полнотекстовый поиск — это технология, позволяющий искать любое слово или фразу в большой коллекции документов или данных. Полнотекстовый поиск отличается от поиска на основе метаданных или части текста, который может не отражать полное значение или контекст запроса. Полнотекстовый поиск использует полнотекстовый механизм, такой как Lucene, для выполнения поиска и возврата результатов.

IBSurgeon Full Text Search UDR — это набор внешних процедур и функций (UDR), которая интегрирует Lucene с Firebird. IBSurgeon Full Text Search UDR позволяет выполнять полнотекстовый поиск в таблицах Firebird в полях varchar и BLOB с использованием движка Lucene.

Эта UDR доступен с открытым исходным кодом, но HQbird предоставляет настраиваемый плагин на основе Firebird Streaming для оперативного обновления.

Более подробная информация: https://www.firebirdsql.org/en/full-text-search-udr/

### 1.6.9. Многопоточный backup/restore/sweep, создание индексов

HQbird реализует многопоточные операции обслуживания (sweep), резервного копирования, восстановления и создания индекса. Поддерживаются Firebird 2.5, 3.0 и 4.0, эта функциональность также появилась в “ванильной” версии Firebird 5.0.

Формат файлов резервных копий такой же, как и в “ванильном” Firebird. На тестовом сервере с 8 ядерным процессором и SSD мы имеем следующие результаты (по сравнению с 1 потоком);

- Резервное копирование — в 4-6 раз быстрее.
- Восстановление — в 2-4 раза быстрее на процессорах с 8 ядрами и твердотельных накопителях.
- sweep — в 4-6 раз быстрее.

Реальное ускорение зависит от процессора, дисковой подсистемы сервера и структуры базы данных. Установите HQbird в пробном режиме (до 30 дней) и проверьте, какие результаты будут на вашем сервере!

Более подробную информацию и результаты испытаний можно найти здесь: https://ib-aid.com/articles/firebird-gbak-backuptips-and-tricks#110hqbirdbackup.

### 1.6.10. Параллельное чтение согласованных данных

HQbird, начиная с версии 2.5, поддерживает две важные функции:

1. функция `make_dbkey()`, позволяющая читать таблицу, разбитую на блоки (по страницам указателей PP),
2. и “shared snapshot” для SNAPSHOT транзакций, который позволяет читать согласованные данные в разных соединениях.

Эти функции помогают добиться параллельного чтения больших наборов данных и ускорить операции экспорта в 2-10 раз (например, для экспорта BI или для конвейера данных). Эти функции также доступны в “ванильной” Firebird, начиная с версии 4.0.

- Больше деталей читайте в статье: https://ib-aid.com/articles/parallel-reading-of-data-in-firebird
- Пример приложения с исходными кодами: https://github.com/IBSurgeon/FBCSVExport

### 1.6.11. Пул внешних соединений

В HQbird есть пул внешних подключений для Firebird 2.5, 3.0, и этот пул также доступен в “ванильной” версии, начиная с Firebird 4.0.

Пул внешних подключений позволяет выполнять инструкции `EXECUTE STATEMENT ON EXTERNAL` с меньшими затратами на повторное соединение с внешней базой данных.

Эта функция контролируется в файле `firebird.conf` с помощью параметров `ExtConnPoolSize` и `ExtConnPoolLifeTime`.

С точки зрения приложения не требуется никаких дополнительных действий для использования или неиспользования — оно включается или выключается в конфигурации сервера и полностью прозрачно для приложений. Также можно отключить сборку мусора для запросов, выполняемых во внешних соединениях. Это регулируется параметром конфигурации `ExtConnNoGarbageCollect`.

Подробнее см. Пул внешних подключений

### 1.6.12. Шифрование

HQbird поддерживает шифрование с помощью плагина Encryption Framework. Основные особенности:

1. Плагин шифрования БД (доступен по запросу) для версий 3, 4, 5 под Windows и Linux. Это комплексная и быстрая платформа плагинов шифрования с AES256. Потеря производительности составляет от 4% до 20%, в зависимости от оперативной памяти и конфигурации.
2. Поддержка многопоточной работы (для промежуточных приложений, с подключением к нескольким базам данных).
3. Отправка ключей через `fbclient.dll` для реализации шифрования без изменения

приложения. Если у вас есть инструмент базы данных, не поддерживающий передачу ключей, или стороннее приложение, ключ можно отправить через `fbclient.dll` со специальной конфигурацией.

4. Окно ввода пароля для `fbclient.dll` в Windows и ввода пароля на терминале в Linux.

По запросу мы можем предоставить примеры клиентских приложений на различных языках, таких как Delphi, NET, Java, PHP, C++ и т.д.

### 1.6.13. Автоматическая коррекция `firebird.conf` (`DefaultDbCachePages`)

Неправильная настройка `DefaultDbCachePages` в `firebird.conf`, `databases.conf` или в заголовке базы данных — распространенная ошибка конфигурации, которая часто случается во время миграции между версиями. Например, это могут быть слишком большие значения Page Buffers в заголовке базы данных для Classic или SuperClassic или слишком низкие для SuperServer.

HQbird автоматически исправит неправильные настройки в файлах `firebird.conf` и `databases.conf` и перезапишет их, если конфигурация не подходит для выбранной архитектуры.

### 1.6.14. Расширенный мониторинг производительности (трассировка, MON, блокировки, ЦП, ОЗУ, диск)

Расширенный мониторинг производительности в HQbird — это функция, которая позволяет вам отслеживать и анализировать производительность ваших баз данных Firebird (версии 5.0, 4.0, 3.0, 2.5) в режиме реального времени. Он собирает данные из различных источников, таких как Trace API, таблицы MON$, таблица блокировок, транзакции, использование ЦП и ОЗУ, и отображает их в графической и табличной формах. Вы можете увидеть общие тенденции производительности, а также детализировать каждую минуту, запрос или транзакцию.

Вы также можете выявить проблемы с производительностью, такие как медленные и частые запросы, длительные транзакции, скачки в таблице блокировок и т.д., а также просмотреть их планы и статистику.

- Более подробная информация: https://ib-aid.com/monitoring-in-hqbird
- Видео: https://www.youtube.com/watch?v=GuRmHZ8ErZ4

### 1.6.15. Мониторинг запросов с большими сортировками

Эта функция помогает устранять неполадки в запросах, которые создают большие отчеты, в которых необходимо отсортировать множество записей. HQbird может отслеживать запросы и операции, которые создают файлы сортировки, превышающие заданный размер. При обнаружении такого запроса его текст записывается в файл `firebird.log`.

Для настройки используется параметр `TempSpaceLogThreshold` в `firebird.conf`, который определяет размер файла сортировки для мониторинга.

### 1.6.16. Управление и настройка репликации для множества баз данных с помощью инструментов командной строки

Если в папке хранится много баз данных, и вы хотите зарегистрировать их все в HQbird для настройки репликации, в HQbird v2024 есть новая команда командной строки для создания файла JSON из папки с регистрационной информацией, который можно использовать для массовой регистрации.

Что касается реплик, существует специальная версия HQBird Central для реплик, которая позволяет хранить сотни реплик (с разных серверов) на одном сервере. HQbird Central для реплик поставляется по запросу.

### 1.6.17. Автоматизация backup,restore и backup/restore

1. Резервные копии: HQbird реализует все типы резервных копий со сложным или простым планированием (все можно делать онлайн с подключенными пользователями):
    a. Проверенная резервная копия с помощью `gbak.exe`. Традиционный формат резервного копирования Firebird, когда Firebird считывает каждую запись в базе данных, гарантируя ее работоспособность. В HQbird (версии 2.5-5.0) проверенное резервное копирование выполняется очень быстро за счет поддержки многопоточности. HQbird реализует ротацию проверенных резервных копий, сжатие и тестовое восстановление. HQbird рассчитывает необходимое пространство для резервных копий, чтобы гарантировать, что резервная копия поместится в свободное пространство, и создает подробные журналы для всех операций.
    b. Инкрементное резервное копирование. Быстрое резервное копирование на физическом уровне, при котором копируются измененные страницы данных. HQbird предлагает 3 схемы резервного копирования: простое еженедельное трехуровневое резервное копирование, расширенное многоуровневое резервное копирование (до 5 уровней) и резервное копирование дампа для создания копии базы данных. Файлы резервных копий ротируются, рассчитывается необходимое пространство.
2. Восстановление
    a. Восстанавливает базы данных из резервных копий. HQbird позволяет восстановить базу данных из FBK. Это особенно важно для облачных экземпляров, когда FBK загружается в облачный экземпляр, поэтому нет необходимости подключаться к консоли сервера (т. е. по ssh или RDP).
    b. Тестовое восстановление как часть проверенного процесса резервного копирования. Вы можете выполнить проверку восстановления из новой резервной копии, это будет выполнено как часть процесса восстановления из проверенной резервной копии.
    c. Плановое восстановление. Возможна организация планового восстановления проверенных (gbak) резервных копий и/или инкрементных (nbackup) резервных копий, например, в рамках инфраструктуры резервного копирования.
3. Автоматическое резервное копирование-восстановление. Поддержка полного цикла резервного копирования-восстановления, как планового, так и по запросу. HQbird

выполнит полное резервное копирование и восстановление безопасным и быстрым способом: остановит всех пользователей, выполнит резервное копирование и восстановление, разрешит пользователей. Старая копия базы данных будет сохранена. В случае возникновения проблемы процесс будет отменен. Если места будет недостаточно, резервное копирование-восстановление не запустится.

С HQbird вы всегда можете отслеживать свои резервные копии и не терять их, независимо от того, сколько у вас баз данных и где они находятся.

### 1.6.18. Передача резервных копий, сегментов репликации через FTP в облако

HQbird может передавать резервные копии (или другие файлы по маске) через FTP, сокеты или на Amazon S3 (требуется плагин, который доступен по запросу).

HQbird также имеет встроенный FTP-сервер и сервер сокетов с простой настройкой.

### 1.6.19. Расширенное обслуживание базы данных: правильная сборка мусора и многое другое

Чрезмерное количество версии записей, также известные как мусорные версии, значительно замедляют работу баз данных Firebird. HQbird реализует правильное сочетание операций sweep и “мягкого” завершения длительных write транзакций и позволяет избежать частого резервного копирования/восстановления базы данных. С HQbird рекомендуется выполнять резервное копирование/восстановление не чаще одного раза в год.

Обслуживание также может включать пересчет статистики индексов и проверку работоспособности индексов, а также проверку работоспособности метаданных.

### 1.6.20. Поддержка множества экземпляров

HQbird позволяет установить несколько экземпляров Firebird разных версий на одном сервере. Это упрощает миграцию с одной версии на другую. HQbird для Windows по умолчанию устанавливает все поддерживаемые версии Firebird (5.0, 4.0, 3.0, 2.5), каждый экземпляр имеет свой порт. Во время установки вы можете выбрать установку только одной версии или нескольких версий.

Чтобы установить HQbird для Linux с несколькими экземплярами, используйте единый установщик (это новая функция HQbird v2024) и укажите, какие версии вам нужны.

### 1.6.21. Тихая установка в Window и Linux

Самый быстрый способ установки HQbird — использовать автоматическую установку из командной строки.

В приведенном ниже примере мы установим HQbird с Firebird 3.0 в `c:\HQbird`, конфигурация будет в `c:\HQbirdData\config`, выходные файлы будут в `c:\HQbirdData\output`.

```
HQbirdServer2024.exe /VERYSILENT /SP- /TYPE="hqbird30x64" /DIR="C:\HQbird2020"
/CONFIGDIR=C:\HQBirdData\config /OUTPUTDIR=C:\HQBirdData\output
```

Смотри также:

- Как установить на Linux: Установка HQbird Server в Linux
- Подробно о тихой установке: Тихая установка в Windows

### 1.6.22. Инструменты для анализа статистики

Пакет администратора HQbird (он работает в Windows) включает в себя Database Analyst, инструмент, который помогает пользователю подробно анализировать статистику базы данных Firebird и выявлять возможные проблемы с производительностью базы данных, ее обслуживанием и взаимодействием приложения с базой данных. IBAnalyst графически отображает статистику базы данных Firebird в удобной для пользователя форме и выявляет следующие проблемы:

- Фрагментация таблиц и BLOB
- Версии записей
- Сборка мусора
- Неэффективные индексы

Подробнее см. Анализ структуры базы данных

### 1.6.23. Инструмент для анализа соединений/транзакций/потребления памяти/операций ввода-вывода

HQbird MonLogger — это инструмент для анализа вывода таблиц мониторинга в Firebird, поиска проблем с медленными SQL-запросами, неправильно спроектированных транзакций (длительных транзакций, транзакций с неправильным уровнем изоляции и т. д.), а также выявления проблемных приложений.

MonLogger может подключиться к базе данных Firebird с проблемами производительности и определить причину медленности: какое-то пользовательское подключение, медленный SQL-запрос или длительная транзакция?

MonLogger поддерживает Firebird 2.1, 2.5, 3.0, 4.0 и 5.0 — для более старых версий Firebird или InterBase используйте FBScanner (не входит в состав HQbird, приобретается отдельно).

MonLogger может показать вам:

- Топ соединений с наибольшим количеством операций ввода-вывода, неиндексированными и индексированными чтениями.
- Топ операторов SQL с наибольшим количеством операций ввода-вывода, неиндексированных и индексированных операций чтения.
- Проблемные транзакции: длительные транзакции, транзакции с ошибочным уровнем

изоляции, транзакции чтения/записи и сопутствующая информация: когда они начались, какие приложения запустили эти транзакции, с какого IP-адреса и т. д.

- Соединения и операторы с наиболее интенсивными действиями по сборке мусора.
- Соотношение чтения/записи, соотношение INSERT/UPDATE/DELETE и многое другое.

### 1.6.24. Инструменты починки БД

HQbird включает лицензию FirstAID, инструмента восстановления для Firebird. IBSurgeon FirstAID — это инструмент, который может автоматически диагностировать и восстанавливать поврежденные базы данных Firebird или InterBase. Он может восстанавливать повреждения, которые не могут исправить ни `gbak`, ни `gfix`. Поддерживаемые версии: Firebird 1.0, 2.0, 2.1, 2.5, 3.0, 4.0, 5.0, InterBase с 4.0 по 2020.

Он использует свой слой для низкоуровневого доступа к базе данных без использования движков InterBase или Firebird, поэтому он может выполнять настоящие “хирургические” операции и восстанавливать вашу базу данных, когда все другие стандартные инструменты (`gfix` и `gbak`) не могут.

### 1.6.25. Оптимизированные конфигурации

HQbird по умолчанию поставляется с оптимизированной конфигурацией, позволяющей максимально эффективно использовать ресурсы мощных серверов и виртуальных машин. Чтобы улучшить конфигурацию HQbird, вы можете использовать “Калькулятор конфигурации” для Firebird, где вы можете выбрать HQbird, чтобы получить базовую оптимизированную конфигурацию для вашей системы здесь: https://cc.ib-aid.com/democalc.html.

Обратите внимание, что Калькулятор создает консервативные конфигурации, и для создания индивидуальной конфигурации вам необходимо отслеживать и анализировать журналы производительности. IBSurgeon может помочь вам создать идеальную конфигурацию в контексте оптимизации/конфигурации/аудита для Firebird: https://ib-aid.com/en/firebird-interbase-performance-optimization-service/

# Глава 2. Установка HQbird

HQbird состоит из двух частей: Server и Admin. Рассмотрим, как их установить.

## 2.1. Установка HQbird Server на Windows

HQbird Server 2024 включает в себя Firebird 2.5, 3.0, 4.0 и 5.0 с репликацией, поддержкой многопоточности и другими улучшениями как часть его установщика, поэтому Firebird необходимо устанавливать как часть HQbird.

Обязательно установите Firebird в комплекте с установщиком HQbird Server, если вы планируете использовать репликацию (для этого также требуется лицензия HQbird Master, Replica или Trial) и другие улучшения.

При желании вы можете выбрать HQbird, который не будет устанавливать Firebird, поэтому вы можете использовать ранее установленный Firebird — в этом случае убедитесь, что установленная версия совместима (2.5.x, 3.0.x, 4.0.x, 5.0.x).

Обратите внимание, что в HQbird полностью поддерживаются только Firebird 2.5, 3.0, 4.0 и 5.0, старые версии Firebird поддерживаются частично. Мы предлагаем комплексную службу миграции Firebird с гарантированным и быстрым результатом для миграции Firebird на последнюю версию.

### 2.1.1. Тихая установка в Windows

Самый быстрый способ установить HQbird — использовать команду тихой установки.

В приведенном ниже примере мы установим HQbird с Firebird 5.0 в `c:\HQbird2024`, конфигурация будет `c:\HQbirdData\config`, выходные файлы в `c:\HQbirdData\output`.

```
HQBird2024.exe /VERYSILENT /SP- /TYPE="hqbird50x64" /DIR="C:\HQbird2024"
/CONFIGDIR=C:\HQBirdData\config /OUTPUTDIR=C:\HQBirdData\output
```

Следующие параметры являются обязательными для выполнения тихой установки:

- `/VERYSILENT /SP` - параметр для выполнения тихой установки
- `/TYPE` – какая версия HQbird должна быть установлена. Если вы выполняете молчаливое обновление, убедитесь, что версия та же, что была установлена ранее.
  - "HQBird25x64" - "HQbird (with Firebird 2.5 x64)";
  - "HQBird30x64" - "HQbird (with Firebird 3.0 x64)";
  - "HQBird40x64" - "HQbird (with Firebird 4.0 x64)";
  - "HQBird50x64" - "HQbird (with Firebird 5.0 x64)".
- `/DIR` - куда установить HQBird. Если вы выполняете молчаливое обновление, убедитесь, что версия та же, что была установлена ранее.
- `/CONFIGDIR` – где хранить данные конфигурации для HQbird.

- /OUTPUTDIR – где хранить выходные данные (расположение по умолчанию для резервных копий, отчетов о производительности и т. д.).

Дополнительные параметры для тихой установки HQbird:

- /fbport=3050 - порт для установки Firebird
- /LOG=C:\temp\HQBirdServerSetup.log - куда будет сохранён журнал установки
- DataGuard параметры:
  - /DGPORT=8082 – порт для веб-интерфейса HQbird (FBDataGuard)
  - /DGLOGIN=admin – логин для веб-интерфейса HQbird (FBDataGuard)
  - /DGPASSWORD=strong password – пароль для веб-интерфейса HQbird (FBDataGuard)
  - /DISSVC - отключает все установленные службы
  - /DISMONSVC - отключает все установленные службы за исключением FBDataGuard
  - /AVMPORM=8083 - порт для Advanced Monitor Viewer
- Параметры автоматической регистрации:
  - /REGEMAIL=youremail@company.com - электронная почта для автоматической регистрации HQbird
  - /REGPASS=yourpassword – пароль от учетной записи IBSurgeon Deploy Center для регистрации HQbird
  - /REGTYPE=R|M|T == Replica, Master, Trial – тип лицензии, необходимо указать, если вам необходимо зарегистрировать HQbird во время установки
- Оффлайн регистрация (несовместимы с REG **)
  - /REGUIK=<uik filename>
  - /REGUNLOCK=<unlock filename>

Должны быть установлены парами, необходимы оба!

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

- Параметры оповещений по электронной почте:
  - /EAHOST=smtp.company.com – SMTP-сервер для оповещений по электронной почте
  - /EAPORT=25 – SMTP-порт для оповещений по электронной почте
  - /EALOGIN=support – Логин SMTP для отправки оповещений по электронной почте
  - /EAPASSWORD=psw – Пароль SMTP для отправки оповещений по электронной почте
  - /EATO=support@email.to – куда отправлять оповещения по электронной почте
  - /EAFROM=someemaildg@company.com – с какого адреса отправляются сообщения
  - /EAENABLED=true – включить или отключить оповещения по электронной почте
  - /EADEFALT=true – отправить копию оповещений по электронной почте в Центр управления IBSurgeon
- Параметры встроенного FTP-сервера:

- `/FTPENABLED=true` – включить или отключить FTP-сервер
- `/FTPPORT=8721` - FTP порт
- `/FTPLOGIN=admin2` - FTP логин
- `FTPPASSWORD=strong password2` - FTP пароль

Обратите внимание, что в случае ошибки, например, если вы пытаетесь запустить тихую установку для установки HQbird в директорию, отличной от текущей, появится всплывающее окно с сообщением об ошибке, и установка будет отменена.

## 2.2. Установка HQbird Server для Windows с помощью установщика

Скачайте HQbird с https://ib-aid.com/en/download-hqbird

Дистрибутив сервера HQbird содержит только 64-битной версий движка Firebird, 32-битные версии Firebird предоставляются по запросу пользователя.

Убедитесь, что установщик HQbird подписан действующим сертификатом IBSurgeon ("iBase LLC") и запустите его:

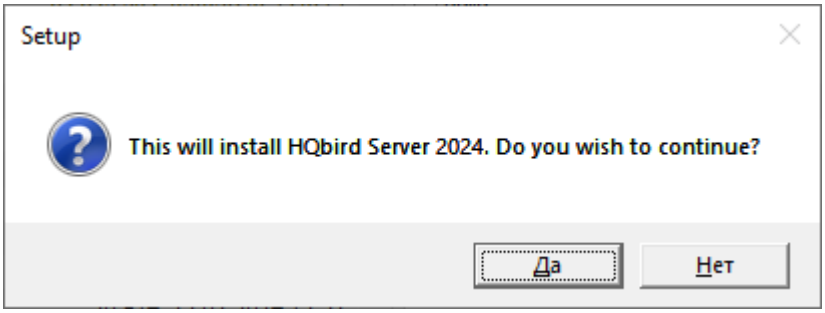



*Рисунок 1. Подтверждение начала процесса установки*

После этого будет запущен мастер установки HQbird Server Side, который проведет вас через несколько шагов, таких как согласие с лицензионным соглашением и выбор папки для установки.

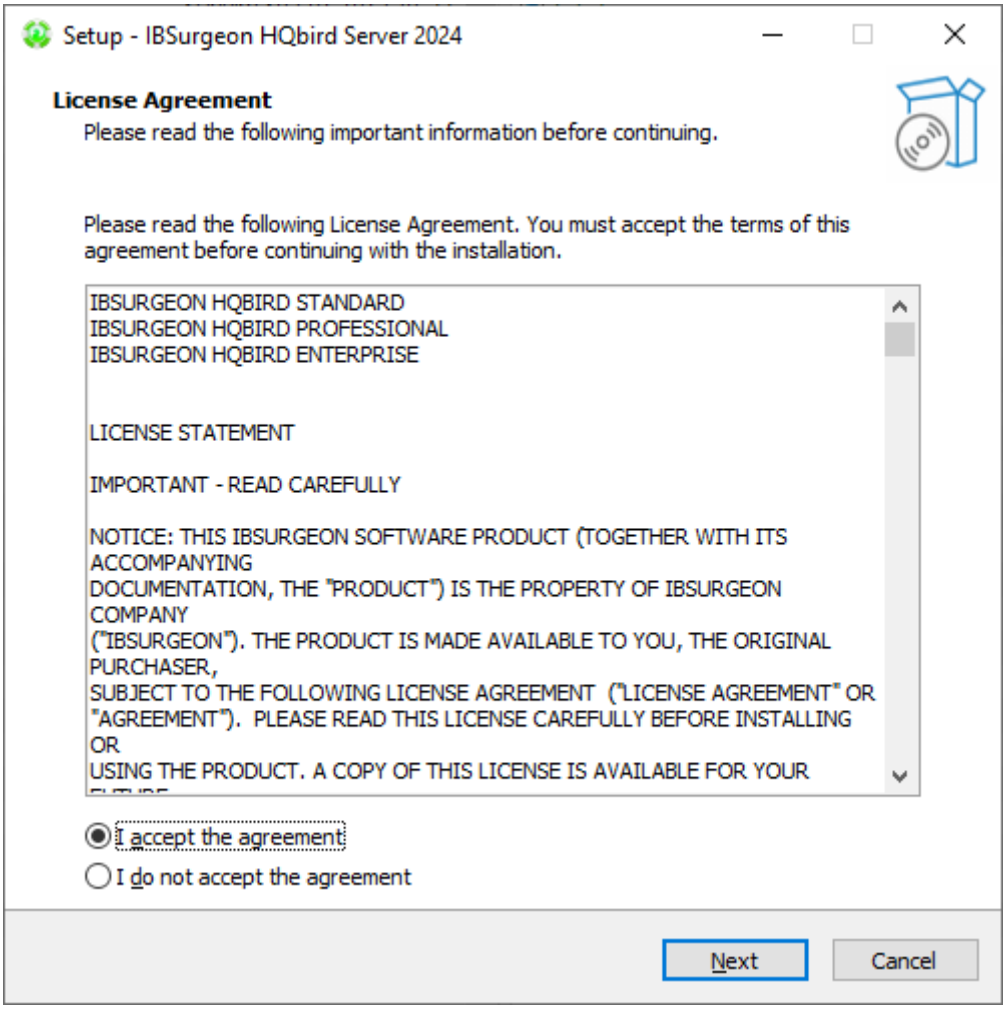



*Рисунок 2. Лицензионное соглашение*

Сначала установщик спросит, куда установить HQbird:

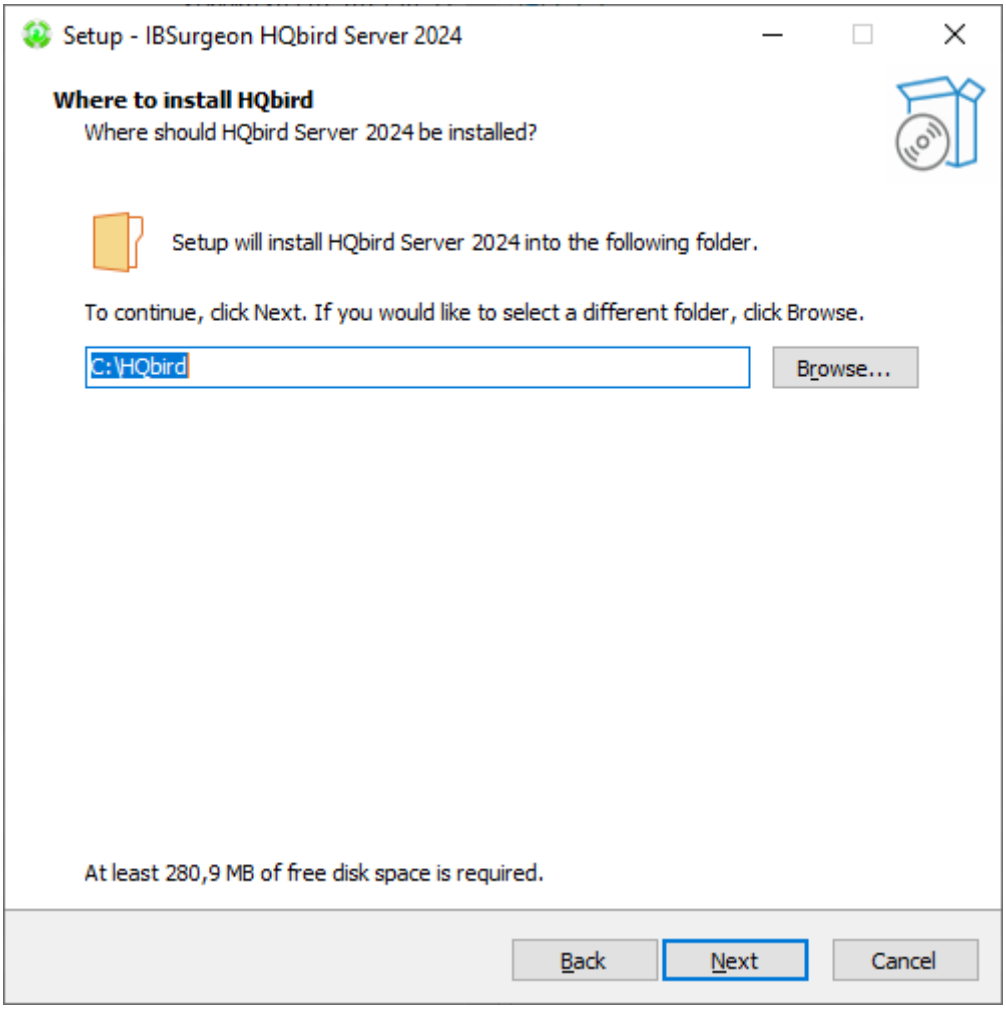



*Рисунок 3. Куда устанавливается HQbird*

Мы рекомендуем использовать расположение по умолчанию `c:\HQbird`, но вы можете использовать любое подходящее расположение.

После этого следует выбрать папки для хранения конфигурационных файлов, резервных копий баз данных, статистики и лог-файлов HQbird:

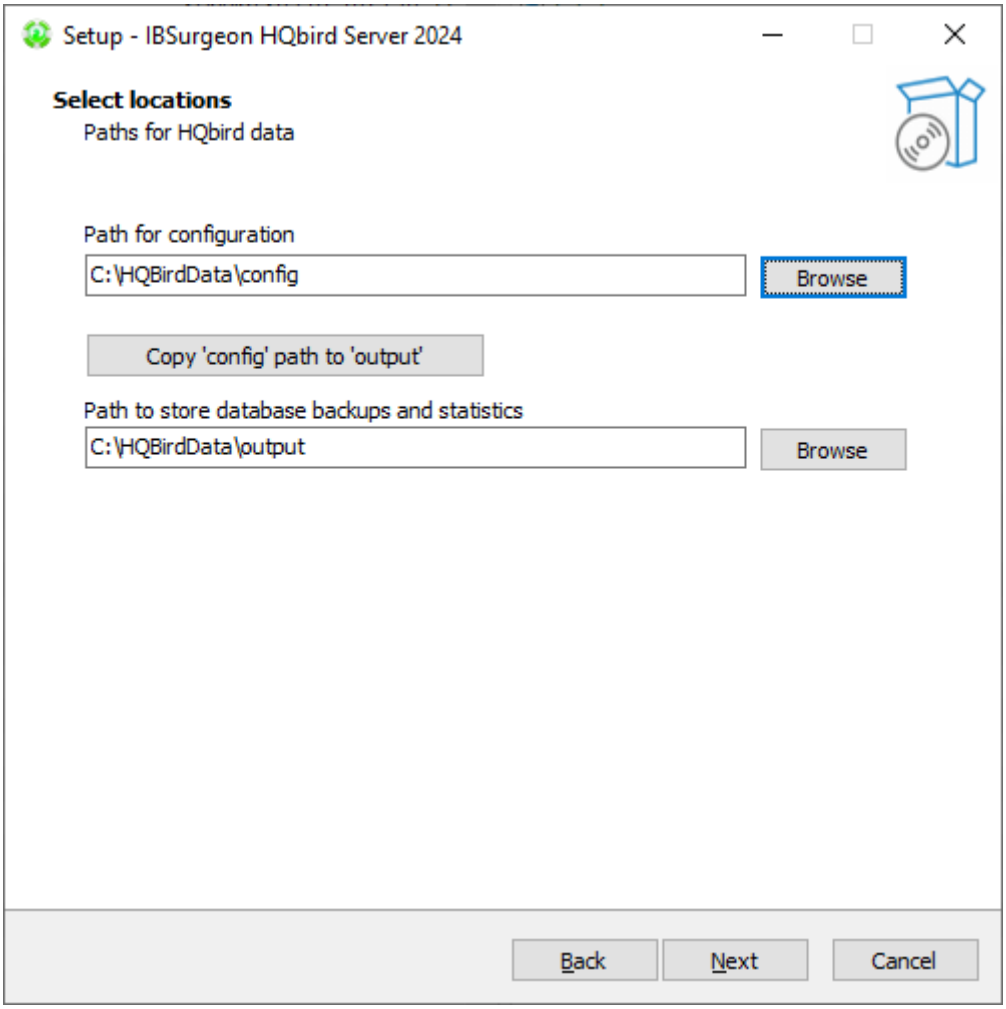



*Рисунок 4. Выберите папки для файлов конфигурации и журналов HQbird.*

По умолчанию мастер установки предлагает создать папки для файлов конфигурации и журналов в `C:\HQbirdData`.

Обычно для этой цели мы рекомендуем выбирать диск с большим количеством свободного места, но вы можете настроить его позже.

Если файлы конфигурации уже существуют в выбранном месте, мастер установки выдаст соответствующее предупреждение:

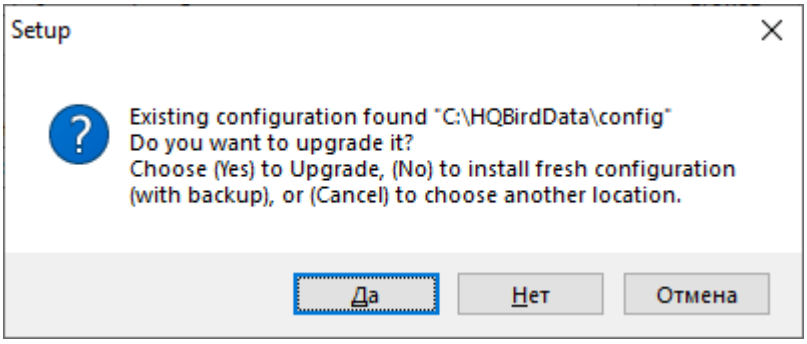



*Рисунок 5. Предупреждение о существующих файлах конфигурации*

Мы рекомендуем автоматическое обновление, поэтому ответ по умолчанию должен быть Yes.

Однако вы можете создать новую конфигурацию HQbird и нажать No — в этом случае программа установки предупредит вас о том, что существующие файлы конфигурации будут перемещены:

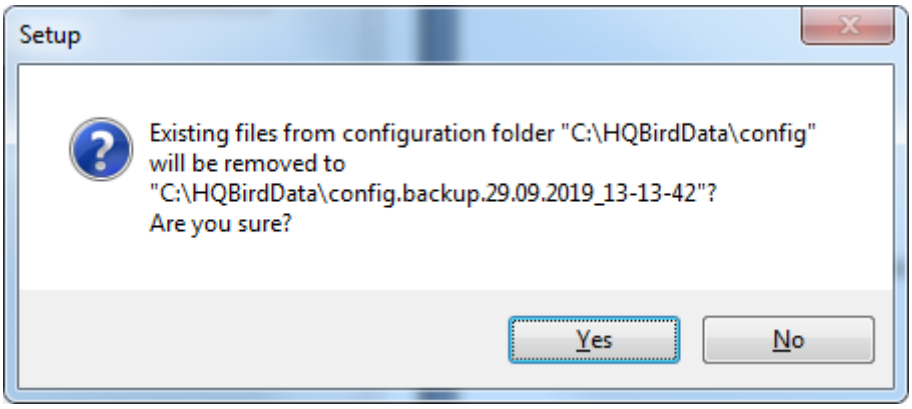



*Рисунок 6. Подтверждение резервной копии*

В случае выбора Cancel вам необходимо указать другое место для файлов конфигурации и output/backups файлов.

После подтверждения папка с существующими конфигурационными файлами будет переименована, и установка продолжится.

После этого вы увидите шаг установки, где вы можете выбрать компоненты для установки:

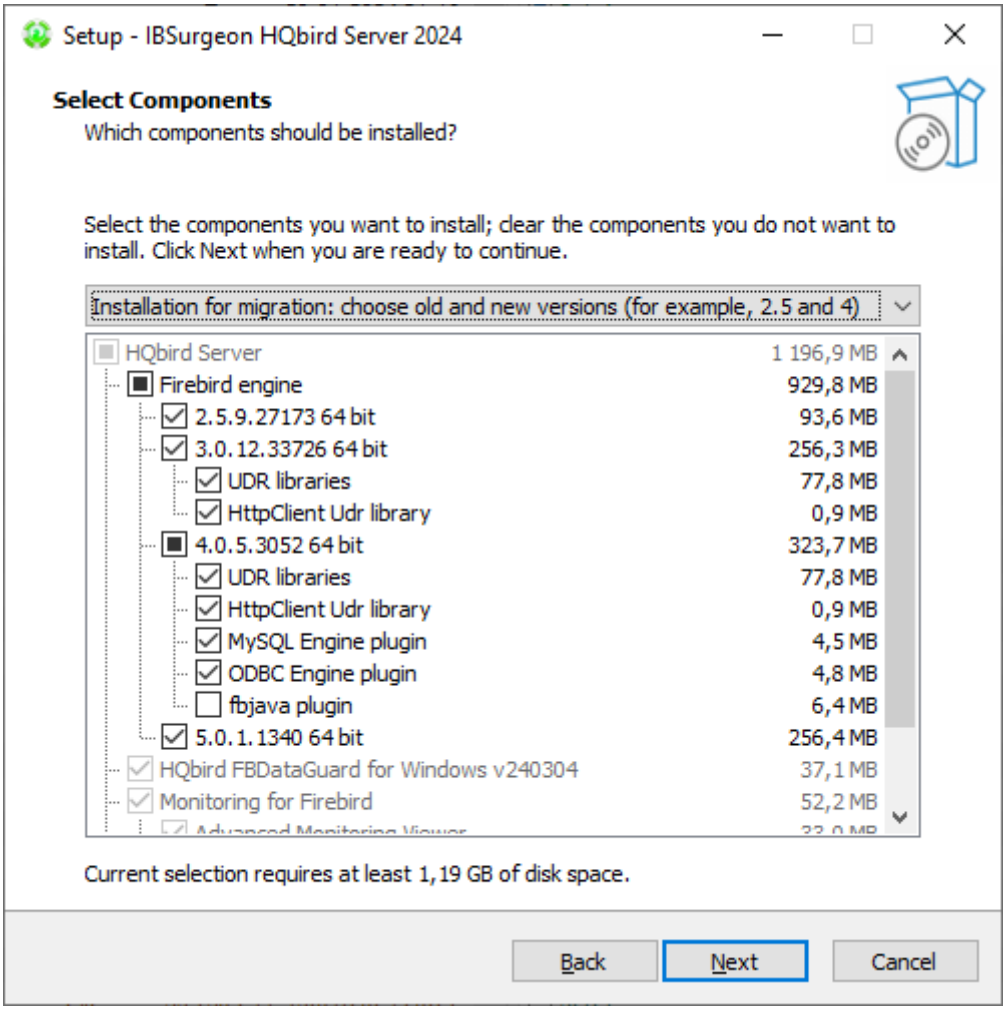



*Рисунок 7. Выберите компоненты сервера HQbird для установки*

Мы рекомендуем вам установить все компоненты HQbird и Firebird, чтобы избежать дальнейшей настройки. Все модули HQbird устанавливаются в неактивном режиме и не влияют на работу сервера Firebird до момента их настройки или использования.

Если вы выберете установку HQbird с Firebird, по умолчанию каждая версия Firebird будет установлен в подпапку установки HQbird. По умолчанию для каждой из версий Firebird:

- `C:\HQbird\Firebird25`
- `C:\HQbird\Firebird30`
- `C:\HQbird\Firebird40`
- `C:\HQbird\Firebird50`

Затем мастер установки попросит указать порт для каждой из версий Firebird, установленных вместе HQbird:

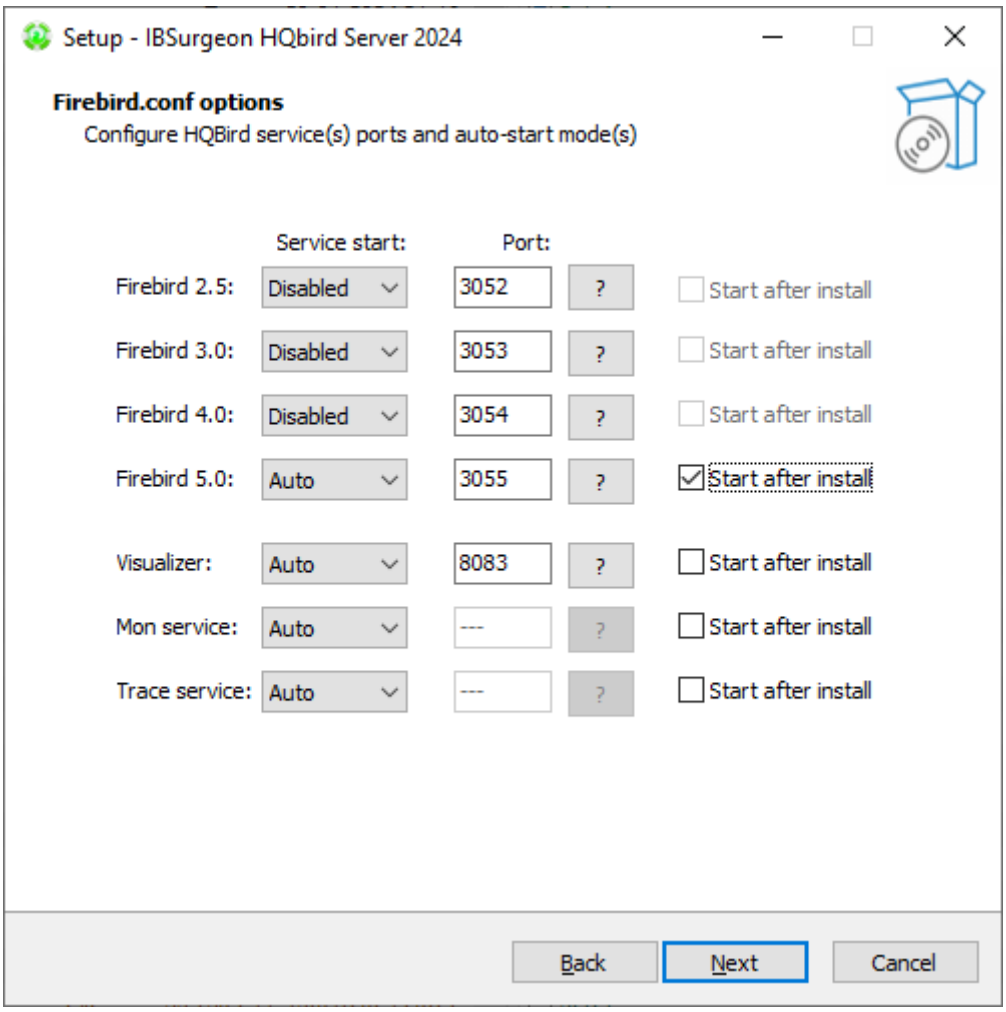



*Рисунок 8. Указание портов для серверов Firebird*

Если порт будет занят другим запущенным Firebird, мастер установки предупредит вас и предложит выбрать другой порт. Или вы можете остановить и удалить другую службу Firebird.

Здесь же вы можете выбрать службы, которые будут стартовать автоматически при старте системы.

Затем вам будет предложено указать порт для HQbird FBDataGuard (веб-интерфейс для управления HQbird):

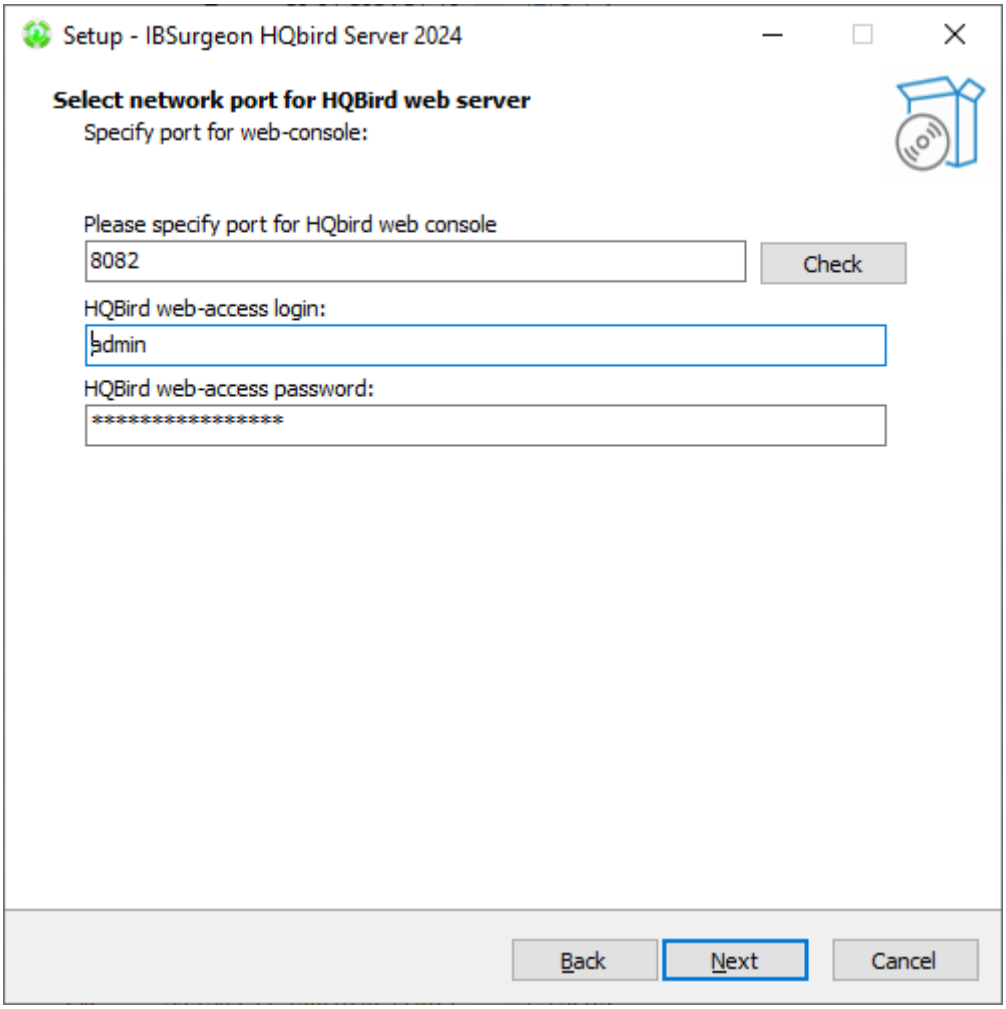



*Рисунок 9. Указание порта, логина и пароля для HQbird FBDataGuard и HQBird Advanced Monitoring Viewer*

Мы рекомендуем оставить 8082, но иногда этот порт может быть занят, поэтому его можно изменить.

Пароль по умолчанию: **strong password**

На следующем шаге вы можете установить настройки встроенного FTP сервера.

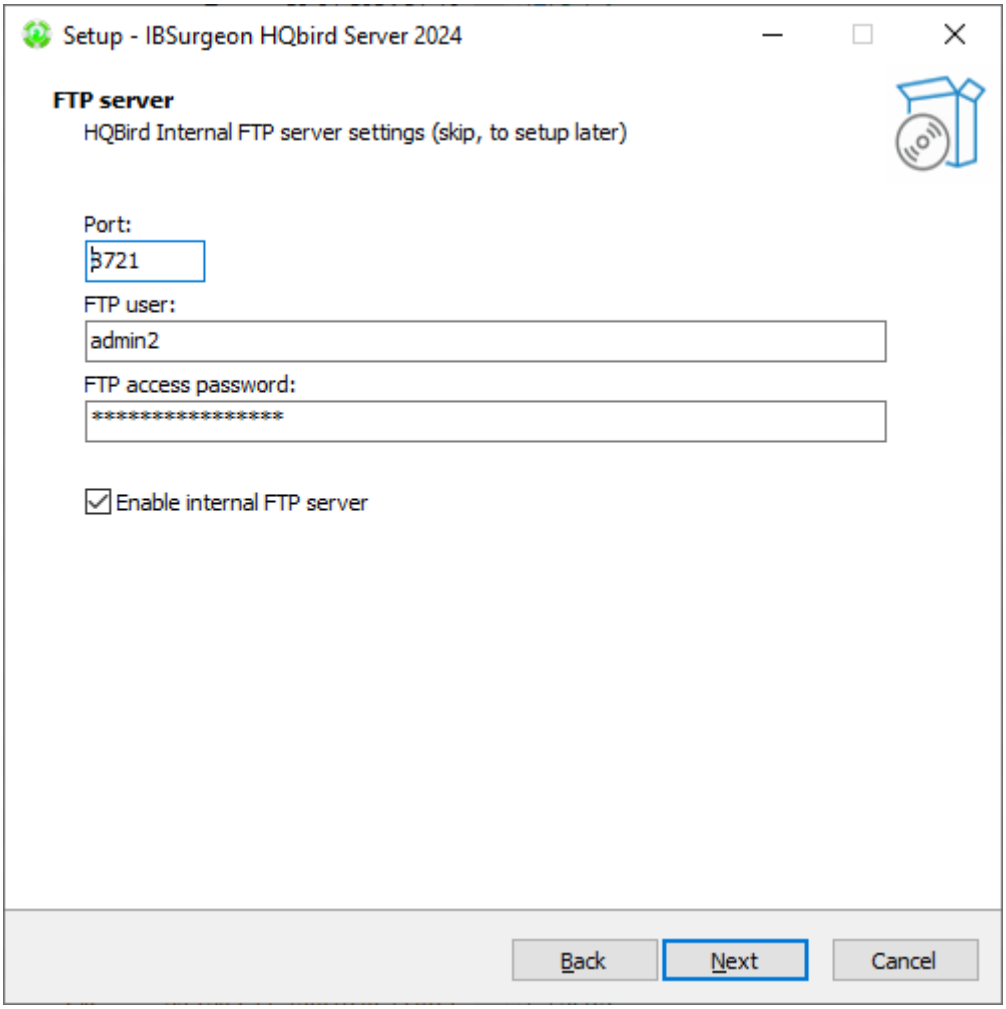



*Рисунок 10. Настройка параметров FTP-сервера*

Пароль по умолчанию: **strong password**

После этого установщик спросит о настройках электронной почты, которые будут использоваться для отправки уведомлений по электронной почте:

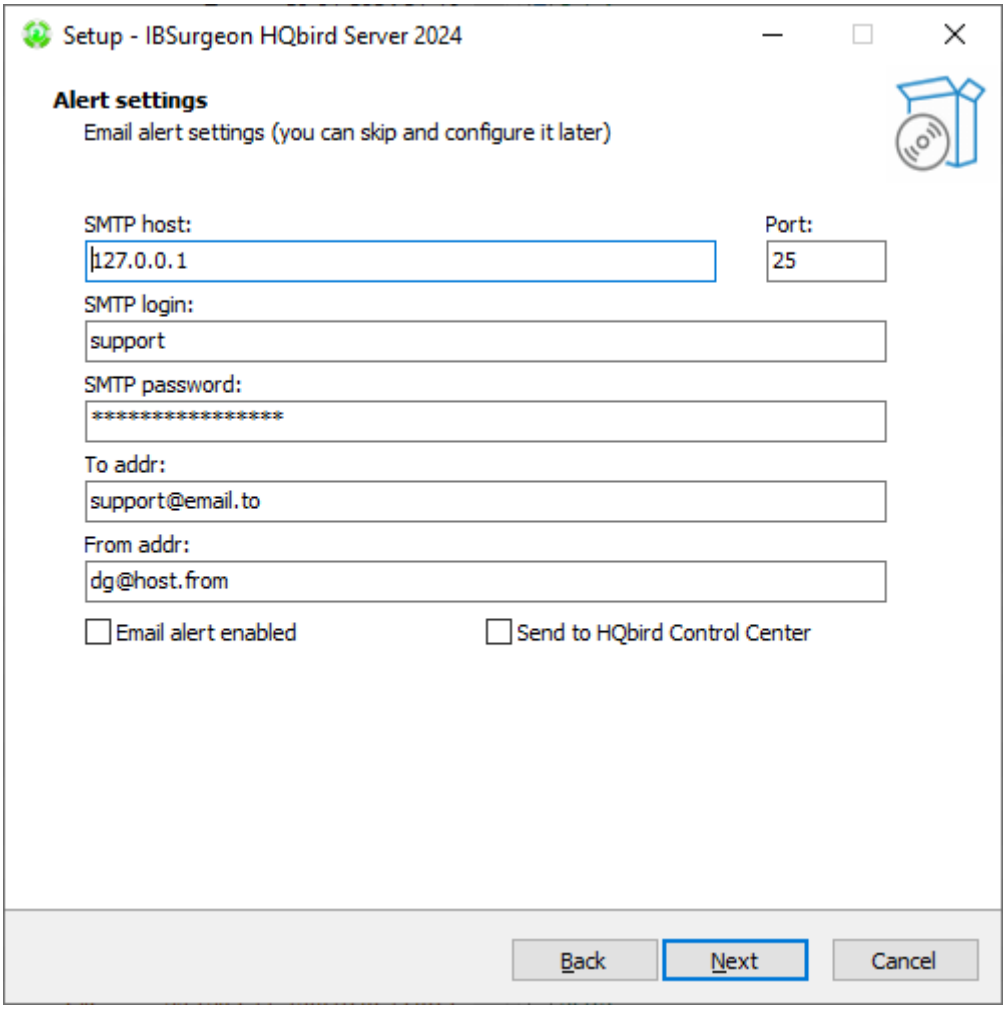



*Рисунок 11. Настройки оповещений по электронной почте*

Вы можете пропустить этот шаг: все оповещения по электронной почте можно настроить позже в веб-интерфейсе.

Затем вы можете указать имя папки и местоположение в меню Windows:

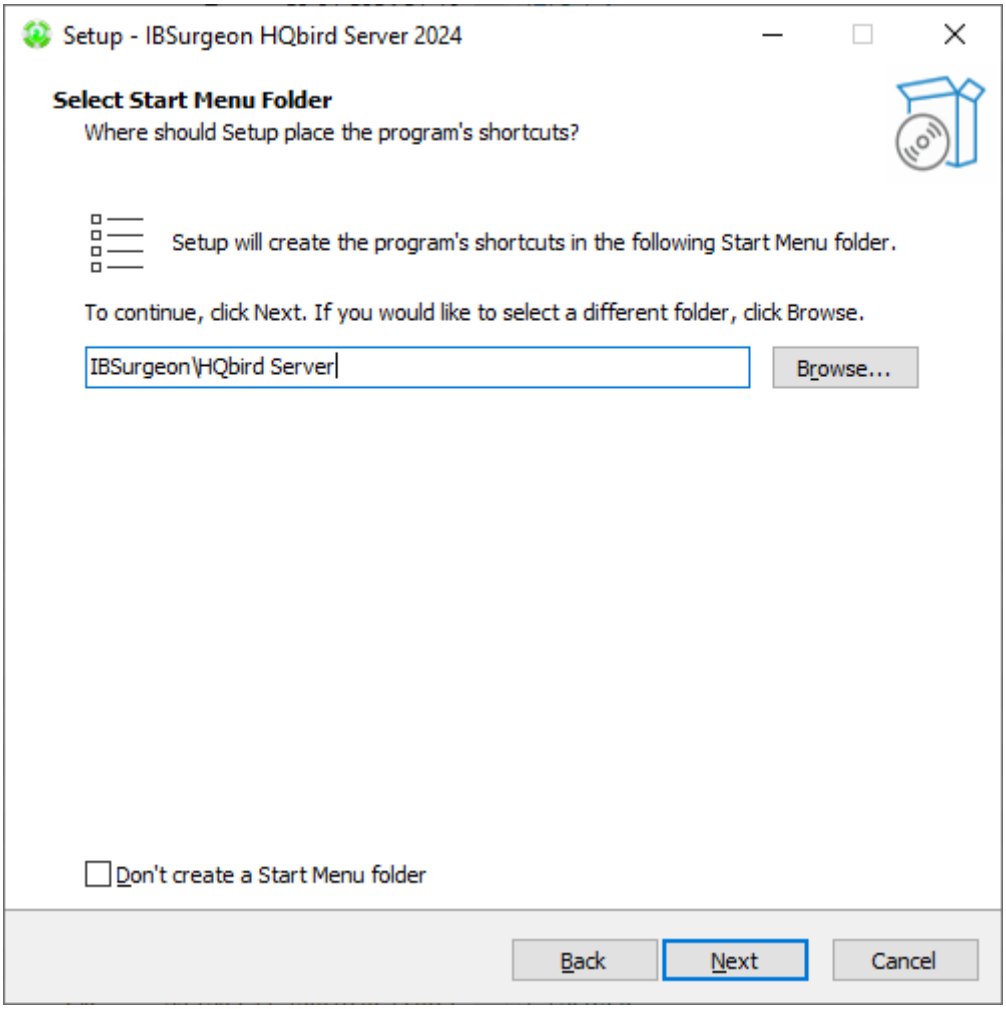



*Рисунок 12. Выбор папки меню "Пуск" Windows.*

На следующем шаге установщик предложит вам предварительно настроить HQbird для использования в качестве главного сервера или сервера-реплики:

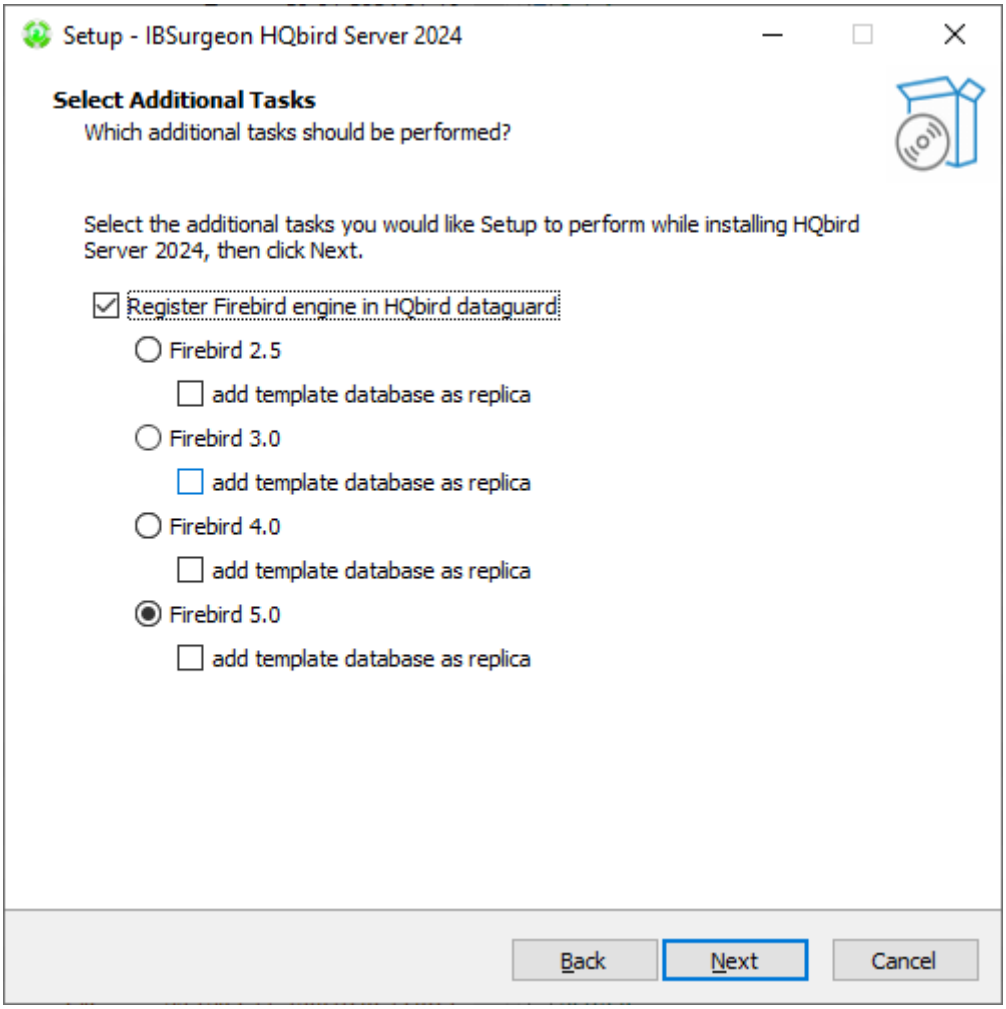



*Рисунок 13. Предварительная настройка для репликации.*

Вы можете пропустить этот шаг, эту настройку можно сделать позже.

Последним шагом является сводка компонентов, которые необходимо установить, и пути к ним:

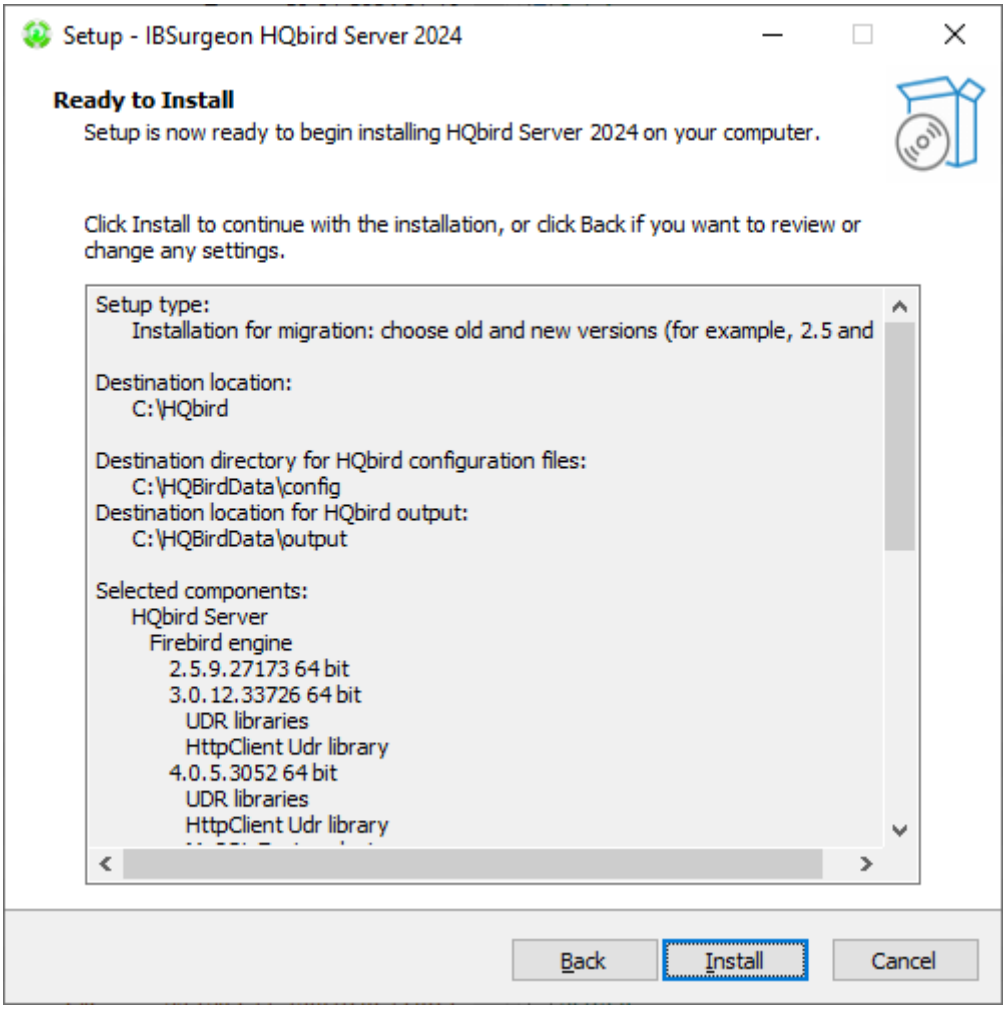



*Рисунок 14. Нажмите Install для завершения установки.*

После этого вам необходимо активировать HQbird (Как активировать HQbird) и перейти к настройке компонентов HQbird.

В конце процесса установки вас спросят о следующих шагах:

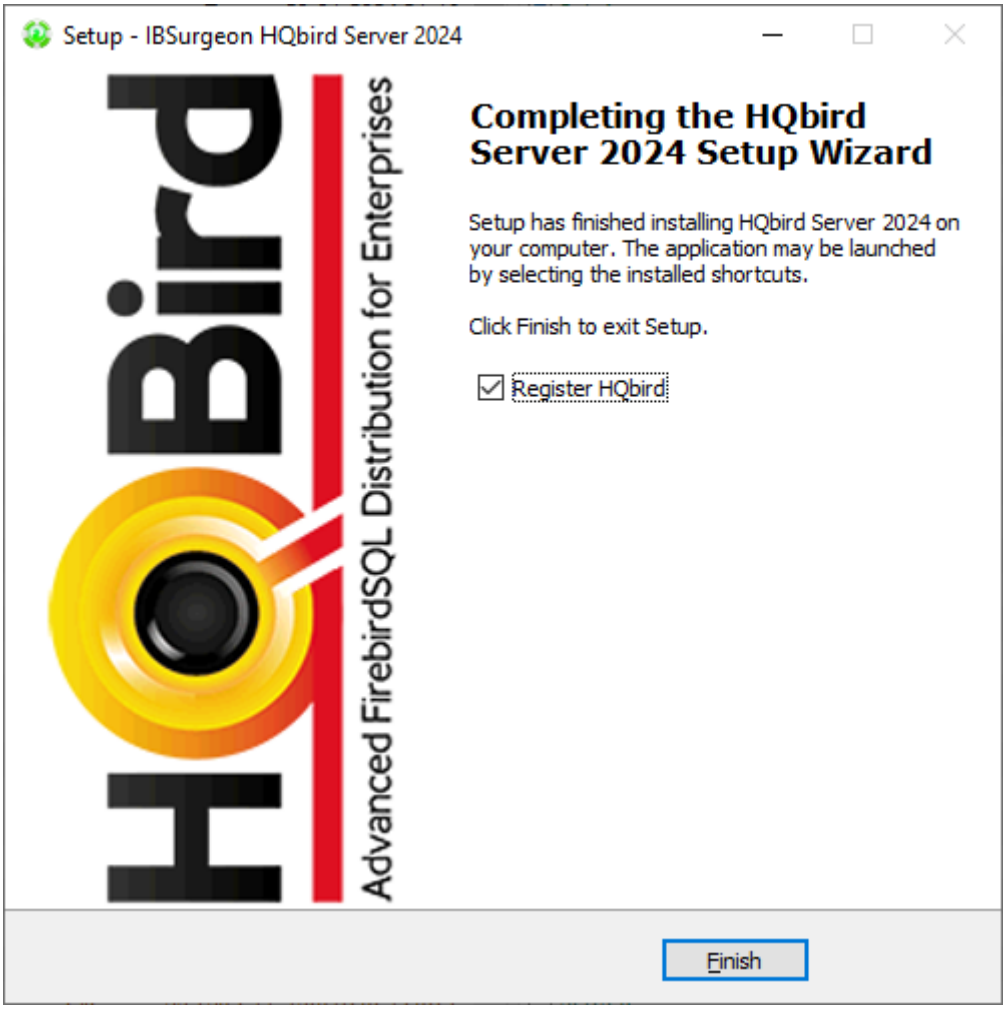



*Рисунок 15. Действия после установки.*

## 2.3. Установка HQbird Administrator в Windows

Для установки HQBird Administrator скачайте дистрибутив по ссылке: https://ib-aid.com/en/hqbird/, или из личного кабинета https://deploy.ib-aid.com.

Имя пакета HQbird Administrator: `HQbirdAdminNNNN.exe` (находится в zip архиве).

Запустите мастер установки и выполните стандартные шаги установки: проверка цифровой подписи, согласие с лицензией, и затем выберите папку для установки:

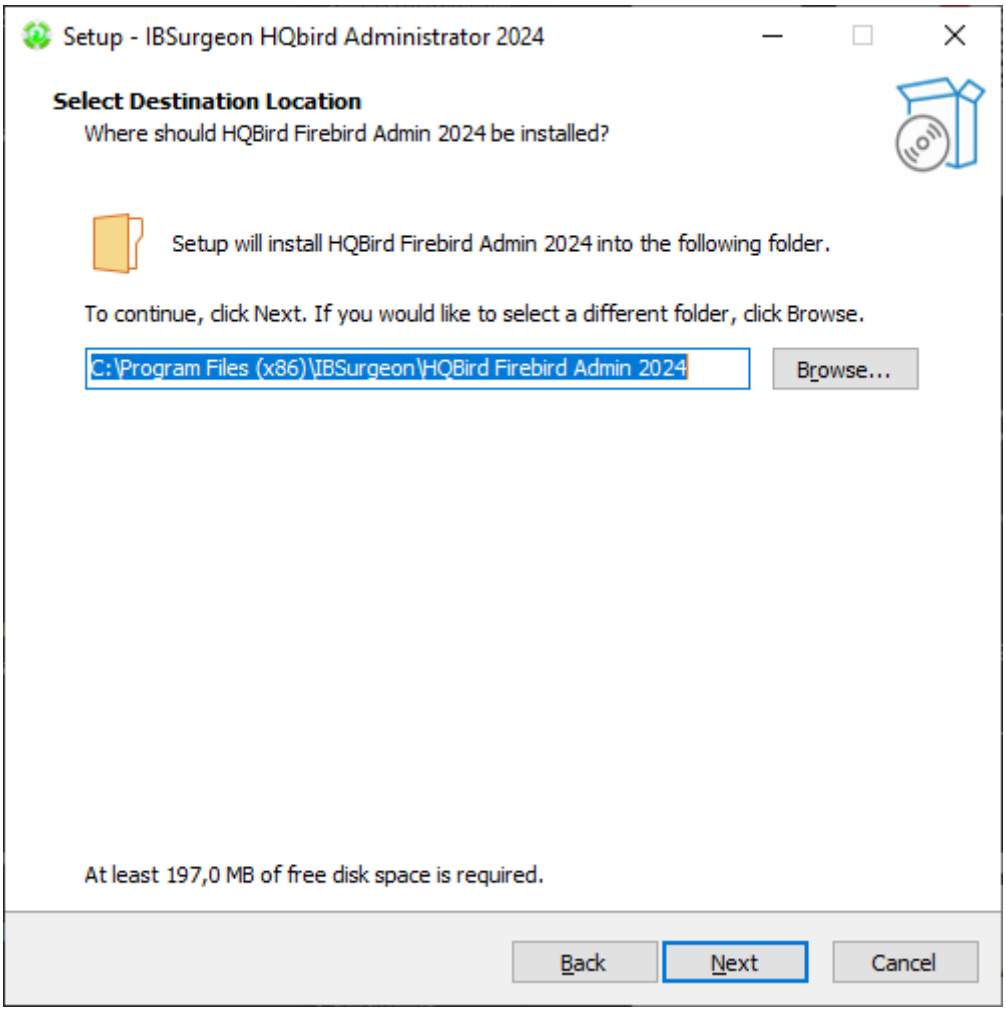



*Рисунок 16. Выбор куда устанавливать HQbird Admin.*

После этого выберите инструменты для установки. Мы рекомендуем установить все инструменты.

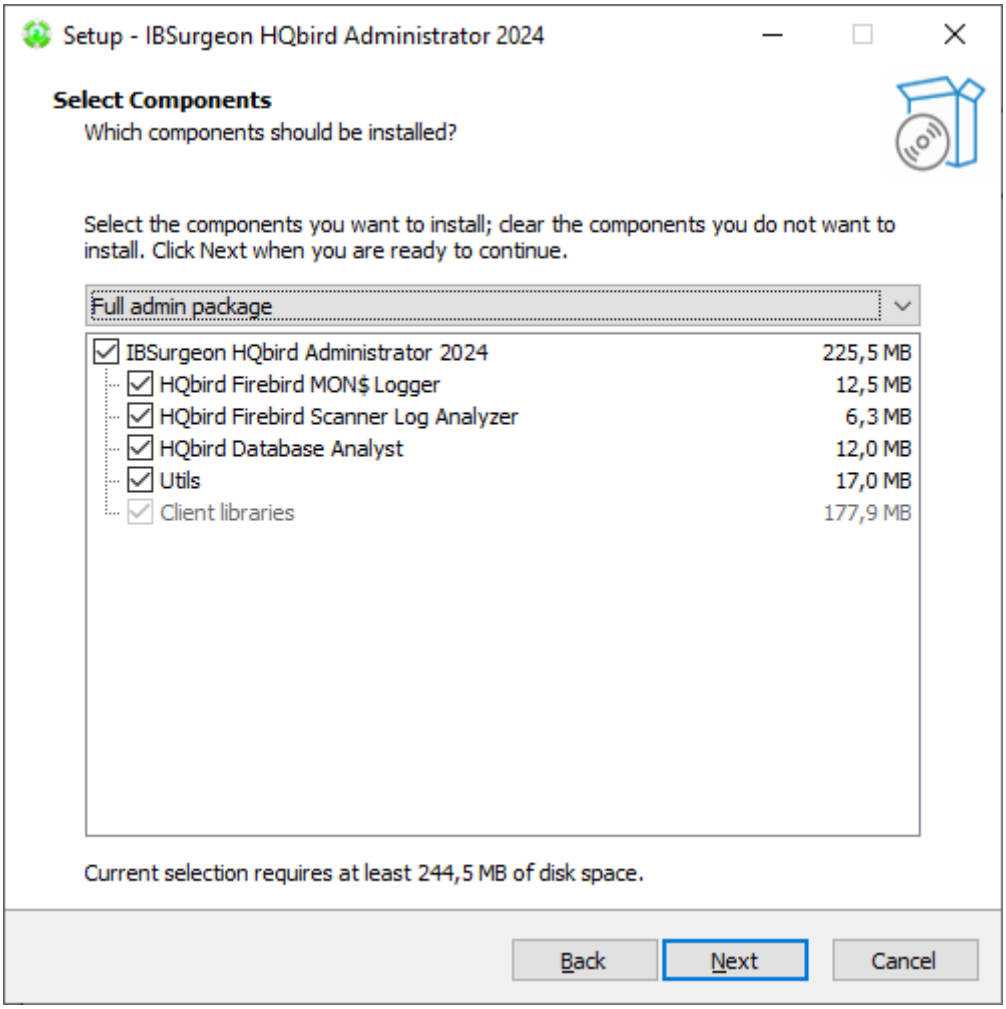



*Рисунок 17. Выбор инструментов для установки.*

После этого следуйте инструкциям. После завершения установки вам будет предложено запустить мастер активации. Если вы устанавливаете HQbird Admin на тот же компьютер, на котором уже был установлен HQbird Server, лицензия будет автоматически обнаружена инструментами HQbird Admin.

### 2.3.1. Как установить общедоступную версию Firebird на Windows

Проще всего установить Firebird в комплекте с HQbird — просто выберите нужную версию во время установки. Однако иногда необходимо использовать HQbird с общедоступной версией Firebird.

Обратите внимание: чтобы включить функции репликации и повышения производительности в HQbird, вам необходимо установить Firebird в комплекте с HQbird ServerSide.

Чтобы установить Firebird отдельно, загрузите zip-архив Firebird с www.firebirdsql.org

Распакуйте файл архива в подходящее место (например, `C:\Firebird25`), после чего скопируйте в эту папку оптимизированный файл конфигурации `firebird.conf` (см. Оптимизированная конфигурация) в эту папку.

Затем для Firebird 2.5 перейдите в папку `bin` (в 3.0 и выше это не нужно), после чего

используйте **Запуск от имени Администратора** для запуска пакетного файла с нужной вам архитектурой.

- Для Firebird 2.5 — запустите `install-superclassic.bat`.
- Для Firebird 3.0 и старше — установите `ServerMode=Super` и запустите `install_service.bat`.

Конечно, вы можете выбрать архитектуру SuperServer для версии 2.5 или ClassicServer для версии 3.0, если считаете, что вам это нужно.

В результате запуска командного файла, Firebird выбранной архитектуры будет установлен и запущен как сервис.

Вы можете убедиться, что служба Firebird установлена и запущена в оснастке **Services** (`services.msc` в командной строке):

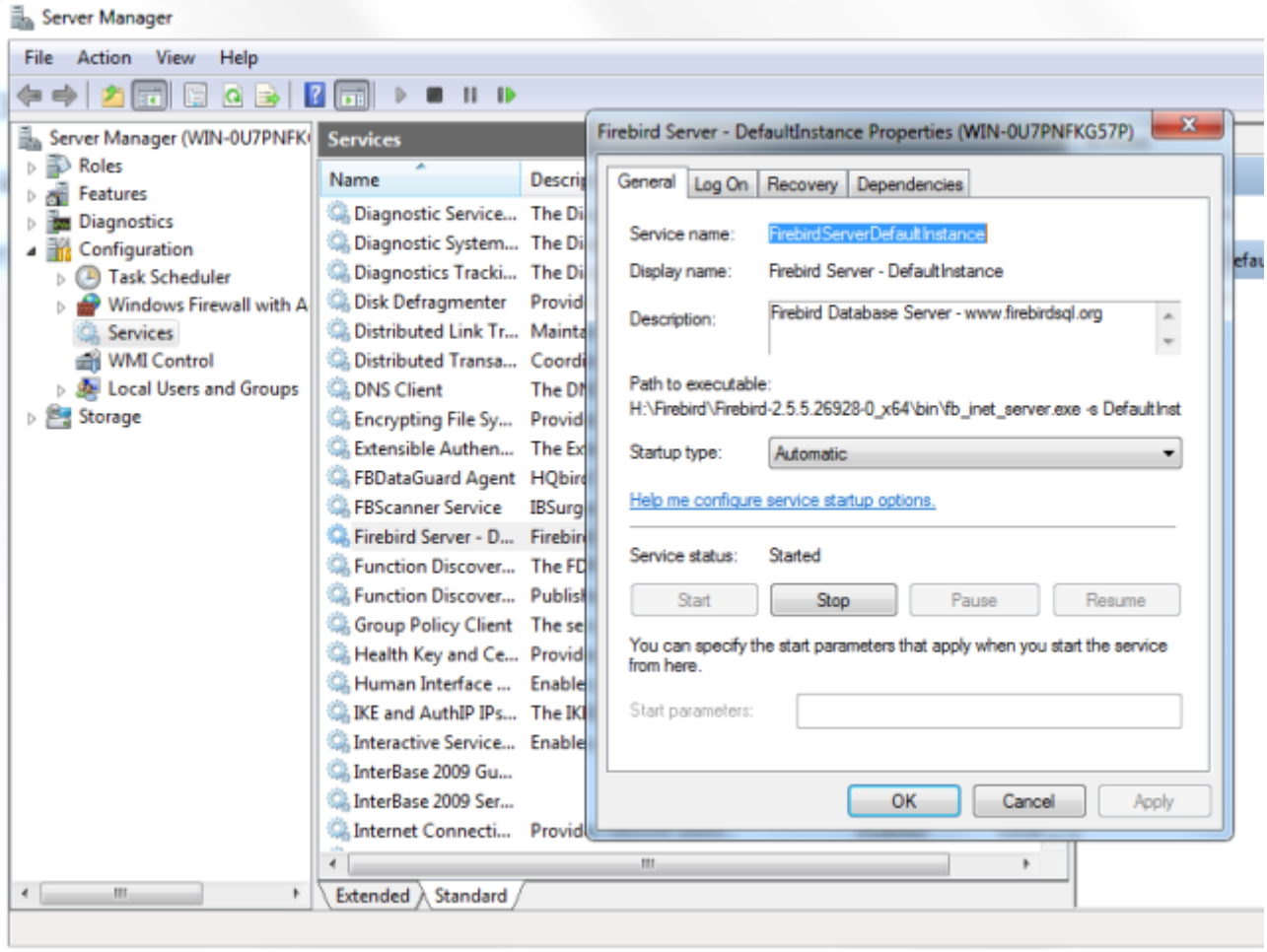



*Рисунок 18. Служба Firebird.*

В этом примере Firebird устанавливается в папку `H:\Firebird\Firebird-2.5.5.26928-0_x64` и работает как служба с архитектурой SuperClassic.

# 2.4. Установка HQbird Server в Linux

Чтобы установить HQbird Server Side в Linux, вам необходимо скачать HQbird ServerSide для Linux со встроенным Firebird соответствующей версии по ссылке https://ib-aid.com/en/download-hqbird/

В зависимости от версии Firebird вам необходимо скачать один из 4 файлов:

- `install_fb25_hqbird2024.sh`
- `install_fb30_hqbird2024.sh`
- `install_fb40_hqbird2024.sh`
- `install_fb50_hqbird2024.sh`

Вы должны быть root или sudoer, чтобы установить HQbird в Linux!

**Основное требование**: установите **java версии 1.8** перед установкой HQbird! Мы рекомендуем OpenJDK, но и Oracle Java тоже подойдет.

## 2.4.1. Установка HQbird с Firebird 2.5 в Linux

1. Перед запуском этой программы установки удалите все ранее установленные версии Firebird. Убедитесь, что у вас не установлен Firebird из репозиториев!
2. Дайте права на выполнение инсталляционному пакету:

```
chmod +x install_fb25_hqbird2024.sh
```

3. Запустите инсталляционный скрипт `install_fb25_hqbird2022.sh`. Он установит Firebird в `/opt/firebird` и HQbird в `/opt/hqbird`
4. По умолчанию Firebird 2.5 устанавливается как Classic. Мы рекомендуем установить его как SuperClassic. Для этого запустите скрипт `/opt/firebird/bin/changeMultiConnectMode.sh` и выберите **thread**

Следующие шаги:

1. Обратите внимание, что Firebird 2.5 будет установлен с SYSDBA/masterkey
2. Вы можете остановить/запустить Firebird 2.5 с помощью команд `service firebird stop` или `service firebird start`. Проверить работает ли он можно с помощью команды `ps aux | grep firebird`
3. Вы можете остановить/запустить HQbird с помощью команд `service hqbird stop` или `service hqbird start`. Проверить работает ли он можно с помощью команды `ps aux | grep dataguard`
4. Запустите браузер и войдите в HQbird FBDataGuard **http://serverurl:8082**, с пользователем/паролем = **admin/strong password**
5. Выберите “I have HQbird” и зарегистрируйте HQbird, используя адрес электронной почты и пароль, которые вы получили от IBSurgeon Deploy Center.

6. При необходимости выполните шаги по настройке или смотрите соответствующую главу данного Руководства.

### 2.4.2. Установка HQbird с Firebird 3.0 в Linux

*Предварительное требование*: убедитесь, что у вас установлены **libtommath**, **libncurses5-dev** и **ICU** (если они не установлены, появится соответствующее сообщение об ошибке).

1. Перед запуском этой программы установки удалите все ранее установленные версии Firebird.
2. Дайте права на выполнение инсталляционному пакету:

```
chmod +x install_fb30_hqbird2024.sh
```

3. Запустите инсталляционный скрипт `install_fb30_hqbird2022.sh`. Он установит Firebird в `/opt/firebird` и HQbird в `/opt/hqbird`
4. По умолчанию Firebird 3.0 устанавливается как SuperServer.
5. Firebird 3.0 будет установлен с SYSDBA/masterkey

Следующие шаги:

1. Вы можете остановить/запустить Firebird 3.0 с помощью команд `service firebird-superserver stop` или `service firebird-superserver start`. Проверить работает ли он можно с помощью команды `ps aux | grep firebird`
2. Вы можете остановить/запустить HQbird с помощью команд `service hqbird stop` или `service hqbird start`. Проверить работает ли он можно с помощью команды `ps aux | grep dataguard`
3. Запустите браузер и войдите в HQbird FBDataGuard **http://serverurl:8082**, с пользователем/паролем = **admin/strong password**
4. Выберите “I have HQbird” и зарегистрируйте HQbird, используя адрес электронной почты и пароль, которые вы получили от IBSurgeon Deploy Center.
5. При необходимости выполните шаги по настройке или смотрите соответствующую главу данного Руководства.

### 2.4.3. Установка HQbird с Firebird 4.0 в Linux

*Предварительное требование*: убедитесь, что у вас установлены **libtommath** и **ICU** (если они не установлены, появится соответствующее сообщение об ошибке).

1. Перед запуском этой программы установки удалите все ранее установленные версии Firebird.
2. Дайте права на выполнение инсталляционному пакету:

```
chmod +x install_fb40_hqbird2024.sh
```

3. Запустите инсталляционный скрипт `install_fb40_hqbird2022.sh`. Он установит Firebird в `/opt/firebird` и HQbird в `/opt/hqbird`
4. По умолчанию Firebird 4.0 устанавливается как SuperServer.
5. Firebird 4.0 будет установлен с SYSDBA/masterkey

Следующие шаги:

1. Вы можете остановить/запустить Firebird 4.0 с помощью команд `service firebird-superserver stop` или `service firebird-superserver start`. Проверить работает ли он можно с помощью команды `ps aux | grep firebird`
2. Вы можете остановить/запустить HQbird с помощью команд `service hqbird stop` или `service hqbird start`. Проверить работает ли он можно с помощью команды `ps aux | grep dataguard`
3. Запустите браузер и войдите в HQbird FBDataGuard **http://serverurl:8082**, с пользователем/паролем = **admin/strong password**
4. Выберите “I have HQbird” и зарегистрируйте HQbird, используя адрес электронной почты и пароль, которые вы получили от IBSurgeon Deploy Center.
5. При необходимости выполните шаги по настройке или смотрите соответствующую главу данного Руководства.

### 2.4.4. Установка HQbird с Firebird 5.0 в Linux

*Предварительное требование*: убедитесь, что у вас установлены **libtommath** и **ICU** (если они не установлены, появится соответствующее сообщение об ошибке).

1. Перед запуском этой программы установки удалите все ранее установленные версии Firebird.
2. Дайте права на выполнение инсталляционному пакету:

```
chmod +x install_fb50_hqbird2024.sh
```

3. Запустите инсталляционный скрипт `install_fb50_hqbird2022.sh`. Он установит Firebird в `/opt/firebird` и HQbird в `/opt/hqbird`
4. По умолчанию Firebird 5.0 устанавливается как SuperServer.
5. Firebird 5.0 будет установлен с SYSDBA/masterkey

Следующие шаги:

1. Вы можете остановить/запустить Firebird 5.0 с помощью команд `service firebird-superserver stop` или `service firebird-superserver start`. Проверить работает ли он можно с помощью команды `ps aux | grep firebird`
2. Вы можете остановить/запустить HQbird с помощью команд `service hqbird stop` или `service hqbird start`. Проверить работает ли он можно с помощью команды `ps aux | grep dataguard`

3. Запустите браузер и войдите в HQbird FBDataGuard **http://serverurl:8082**, с пользователем/паролем = **admin/strong password**
4. Выберите “I have HQbird” и зарегистрируйте HQbird, используя адрес электронной почты и пароль, которые вы получили от IBSurgeon Deploy Center.
5. При необходимости выполните шаги по настройке или смотрите соответствующую главу данного Руководства.

### 2.4.5. Установка HQbird без Firebird в Linux

Если вы не хотите изменять существующую установку Firebird, выполните следующую команду:

```
install_fb4_hqbird2024.sh –-nofirebird
```

Она установит HQbird без бинарных файлов Firebird.

Обратите внимание, что расширенные функции (репликация, поддержка многопоточности, шифрование, аутентификация) требуют HQbird с бинарными файлами Firebird!

### 2.4.6. Настройки брандмауэра

Убедитесь, что эти порты разрешены в конфигурации вашего брандмауэра:

- 3050 - Firebird;
- 3051 - события Firebird
- 8082 - веб консоль DataGuard
- 8083 - мониторинг
- 8721 - пассивный FTP
- 40000-40005 - порты пассивного FTP
- 8720 - активный FTP
- 8722 - socket server
- 8765 - сервер лицензии

Эти порты можно изменить в `/opt/firebird/firebird.conf` (`RemoteServicePort`, `RemoteAuxService`), `/opt/hqbird/conf/network.properties` (`server.port`) и `/opt/hqbird/conf/license.properties` (`serverlicense.port`).

**Внимание!**

После обновления убедитесь, что запущена только одна копия HQbird! Если есть 2 копии, остановите их (`service hqbird stop` для первого экземпляра и `kill <process-number>` для второго экземпляра) и запустите снова.

## 2.5. Обновление существующей версии HQbird

Установщик HQbird в Windows (начиная с версии 2018R2) и на Linux (начиная с версии 2018R3) поддерживает автоматическое обновление конфигурации уже установленной версии HQbird 2017R2 и выше.

Если установщик HQbird заметит предыдущую версию HQbird, он попросит вас подтвердить обновление, и в случае положительного ответа остановит Firebird, HQbird и обновит их файлы.

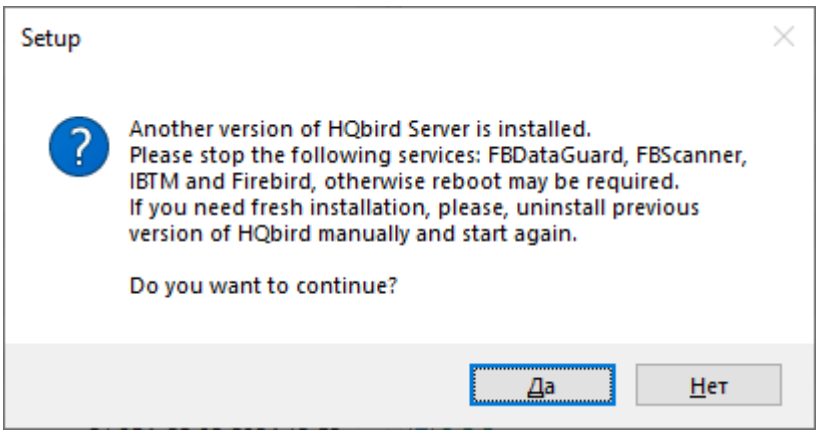



*Рисунок 19. Предупреждение об обновлении.*

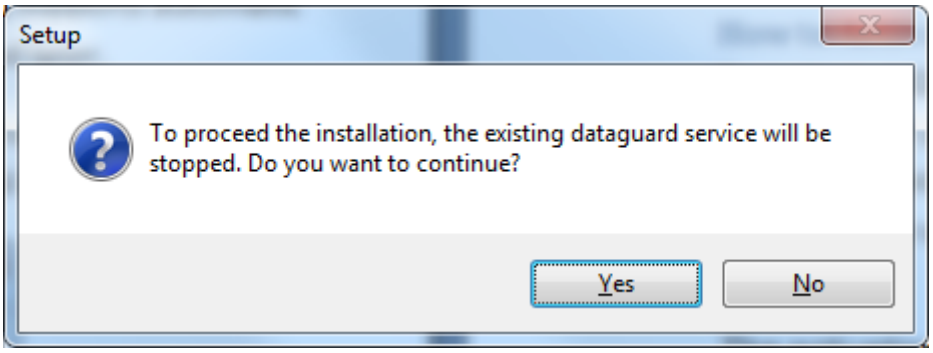


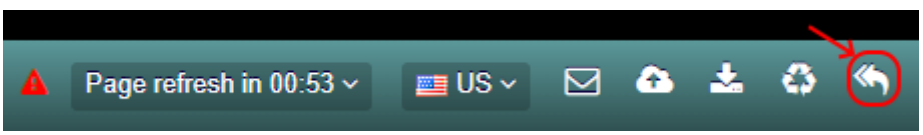

*Рисунок 20. Предупреждение о перезапуске запущенного HQbird FBDataGuard.*

Конфигурация будет сохранена — это означает, что файлы конфигурации `firebird.conf`, `aliases.conf` или `databases.conf`, `securityX.fdb`, и HQbird не будут удалены (файлы конфигурации HQbird будут обновлены до конфигурации новой версии).

Обновление не изменяет настройки службы Windows для Firebird и HQbird — это означает, что если вы изменили свойства службы “Запуск от имени”, то они будут сохранены.

После обновления на Linux Firebird и HQbird нужно перезапустить вручную!

После обновления HQbird откройте веб-консоль и выберите в правом верхнем углу: “Refresh HQbird web-console”. Необходимо очистить кеш JavaScript-части приложения.



Обратите внимание: если вы устанавливаете HQbird 2024 поверх старой версии HQbird для Windows, диалоговое окно с параметрами установки будет отображаться как отключенное, потому что мы не можем автоматически обновить версию 2.5 до версии 3.0, 4.0 или 5.0, а установщик может обновить только те же компоненты. Если вам нужна другая установка, удалите старую версию HQbird с компьютера перед установкой 2024.

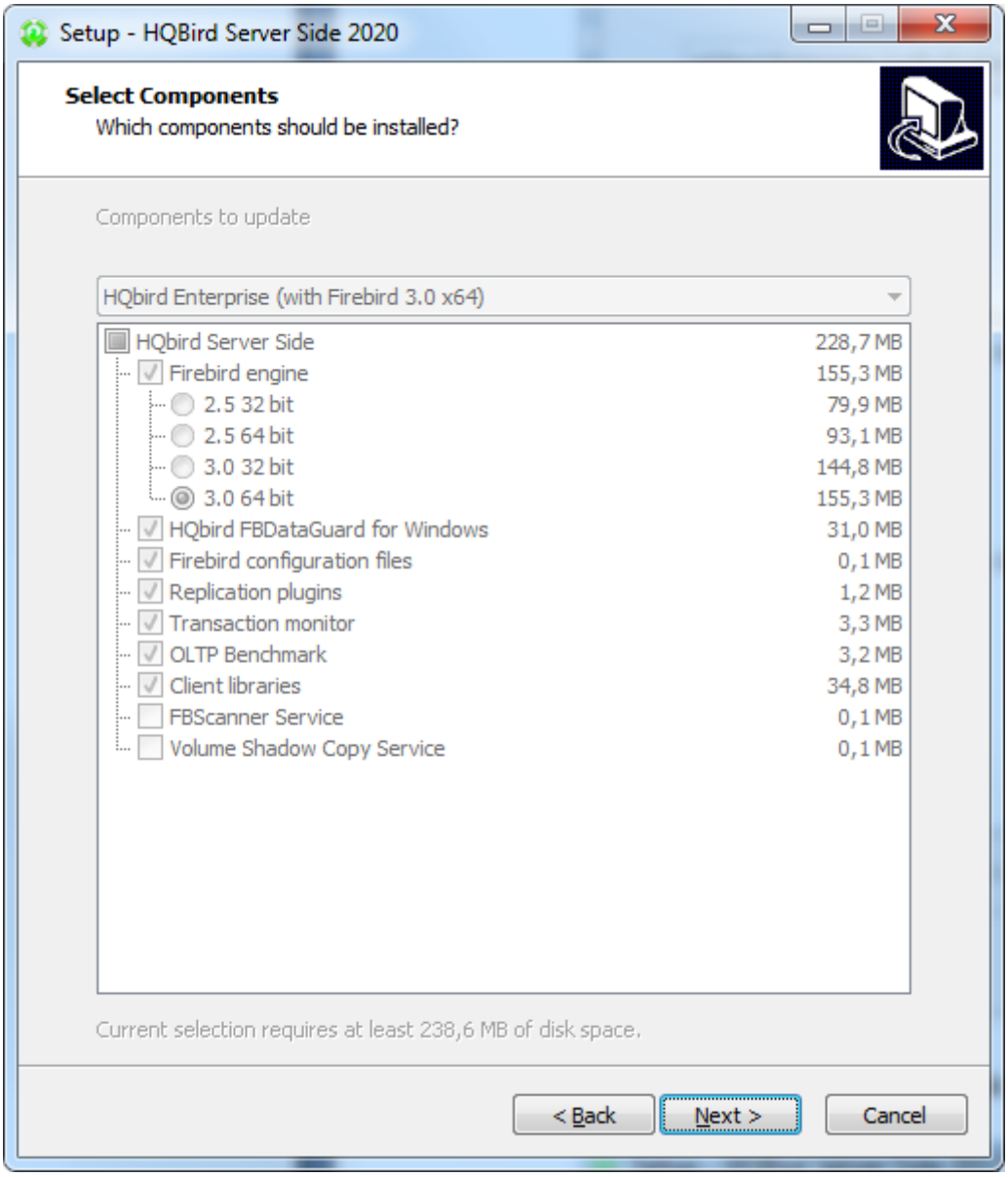



*Рисунок 21. Пример диалога выбора отключенных компонентов при обновлении.*

# 2.6. Регистрация HQbird

## 2.6.1. Как активировать HQbird

Для активации HQbird можно либо использовать отдельную утилиту, входящую в пакеты сервера и администратора для Windows, либо воспользоваться механизмом регистрации, встроенным в веб-интерфейс HQbird Firebird DataGuard (для Windows и Linux), либо запустить любой инструмент из программ администрирования HQbird и воспользуйтесь встроенным мастером активации.

Мастер активации выглядит и работает одинаково в инструментах и в инструменте активации. Достаточно один раз выполнить активацию на любом компьютере, который может подключиться к серверу с установленным HQbird ServerSide.

Вы можете запустить утилиту регистрации из меню **Пуск** (IBSurgeon\Register HQbird):

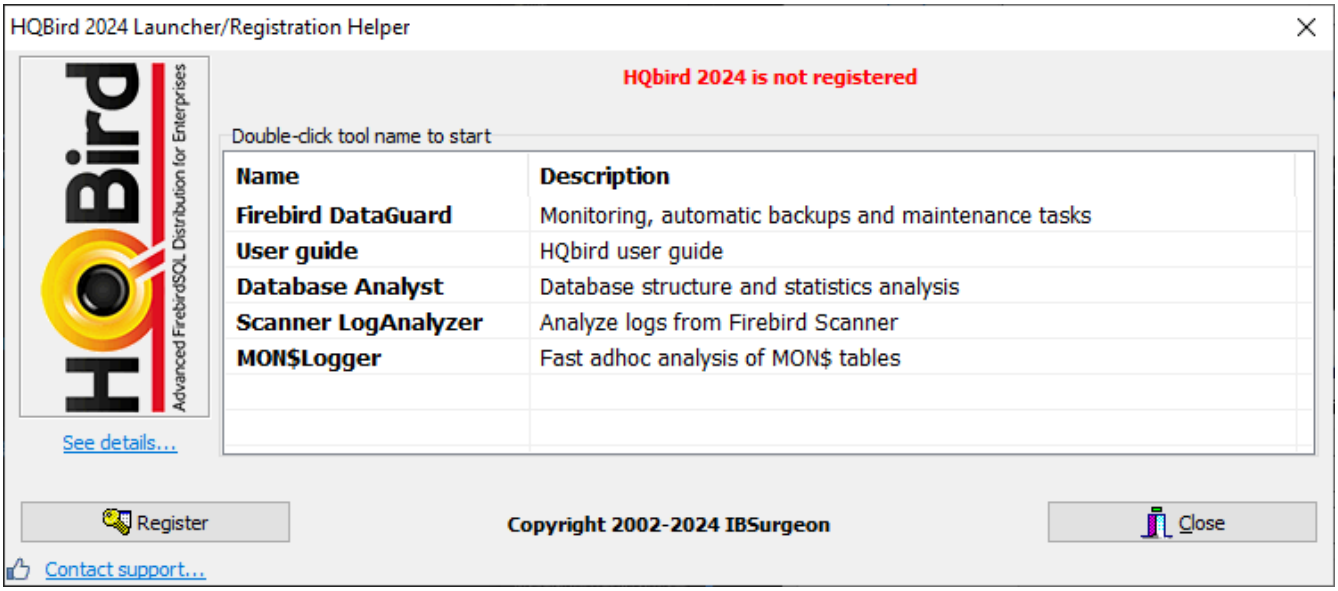



*Рисунок 22. Помощник регистрации HQbird.*

Если вы нажмете кнопку **Register** (или Re-Register для повторной регистрации), вы увидите мастер активации:

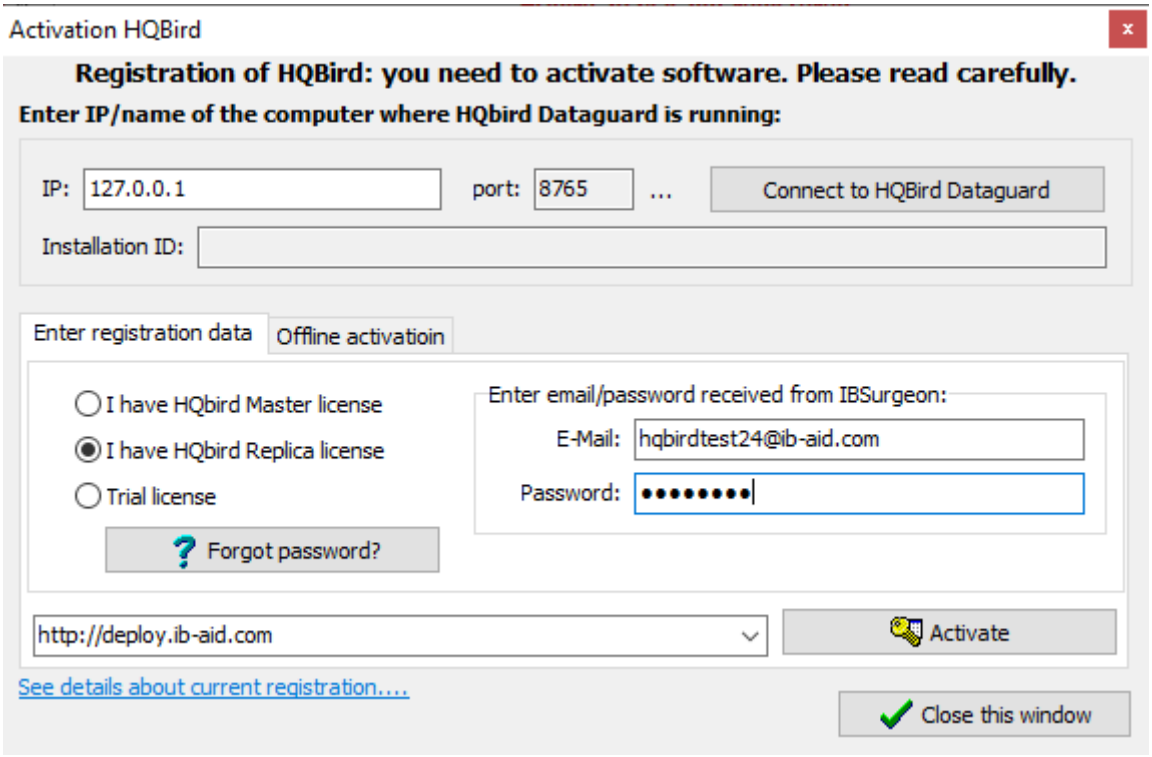



*Рисунок 23. Окно активации HQbird.*

После этого укажите **IP адрес** или **имя компьютера** сервера, на котором установлен HQbird, в верхнем поле ввода и нажмите **Connect to HQbird Server**. Если вы запустили утилиту регистрации на одном компьютере с HQbird Server, то это будет “localhost”, иначе — какой-то удаленный адрес.

Затем введите свои регистрационные данные. Если у вас есть лицензия, введите свой адрес электронной почты и пароль, которые вы использовали для регистрации в IBSurgeon Deploy Center и нажмите **Activate**.

Если у вас нет лицензии, выберите Trial license, укажите свой адрес электронной почты и нажмите **Activate**. Вы будете автоматически зарегистрированы, и пароль будет отправлен на ваш адрес электронной почты.

Сразу после того, как вы нажмете **Activate**, астер регистрации попытается подключиться к IBSurgeon Deploy Center и получить лицензию. В случае успеха вы увидите соответствующее сообщение. Если есть какие-либо проблемы, вы увидите сообщение об ошибке.

Если вы забыли пароль, нажмите кнопку **Forgot password...** она откроет браузер с формой восстановления пароля.

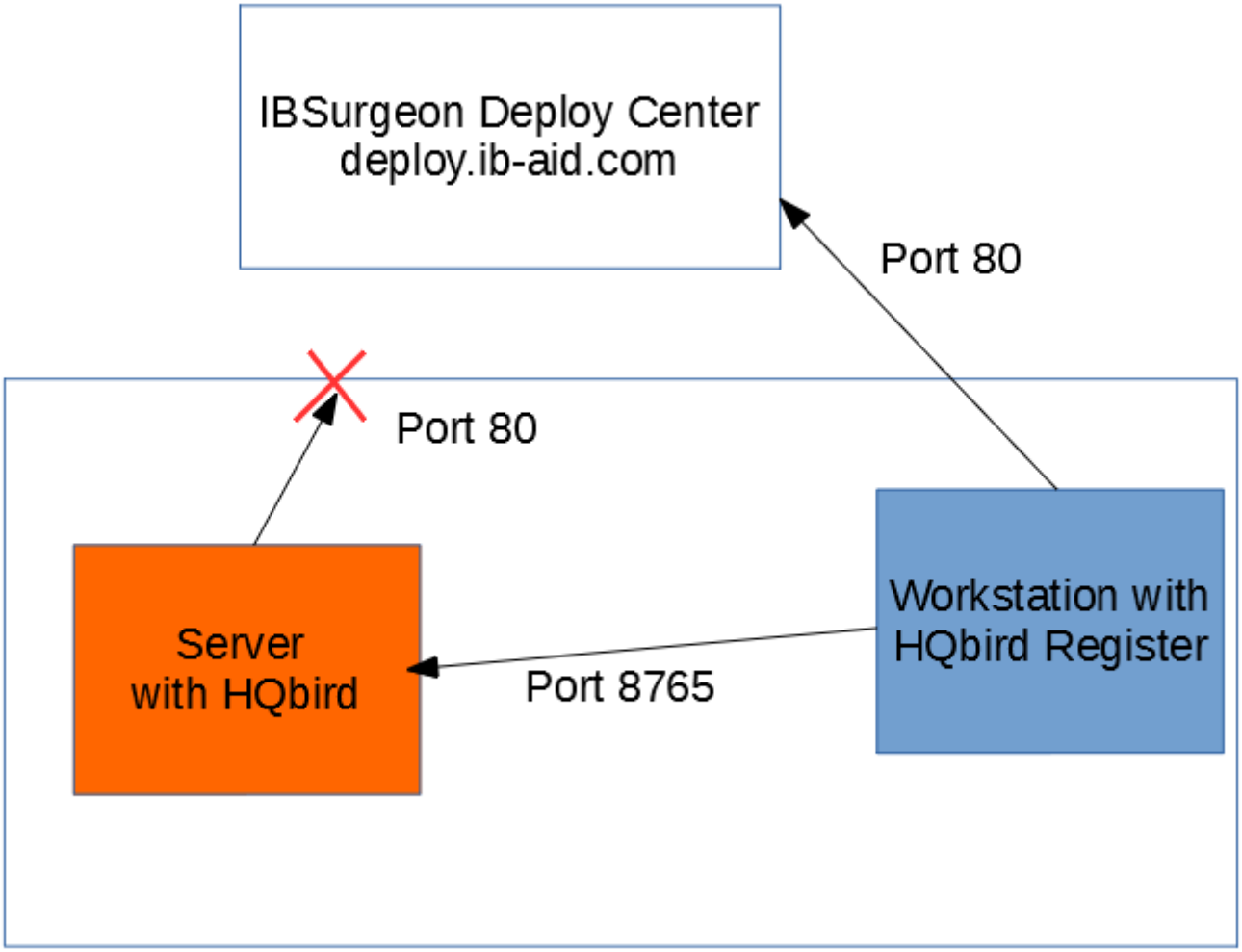
Если вам нужно приобрести новую или дополнительную лицензию, или продлить подписку, нажмите **Purchase.**

Нажмите **Close this window** после завершения регистрации.

### Интернет-активация через клиентский компьютер

Если сервер с HQbird Server не имеет доступа в Интернет, вы все равно можете

активировать его через Интернет: вы можете установить HQbird Administrator на любой клиентский компьютер с Windows, у которого есть и выход в Интернет, и доступ к HQbird Server, и выполнить активацию .




Запустите инструмент HQbird Register и введите туда: IP-адрес вашего сервера (или имя сервера — например, mylinuxserver), адрес электронной почты и лицензию и нажмите **Activate**:

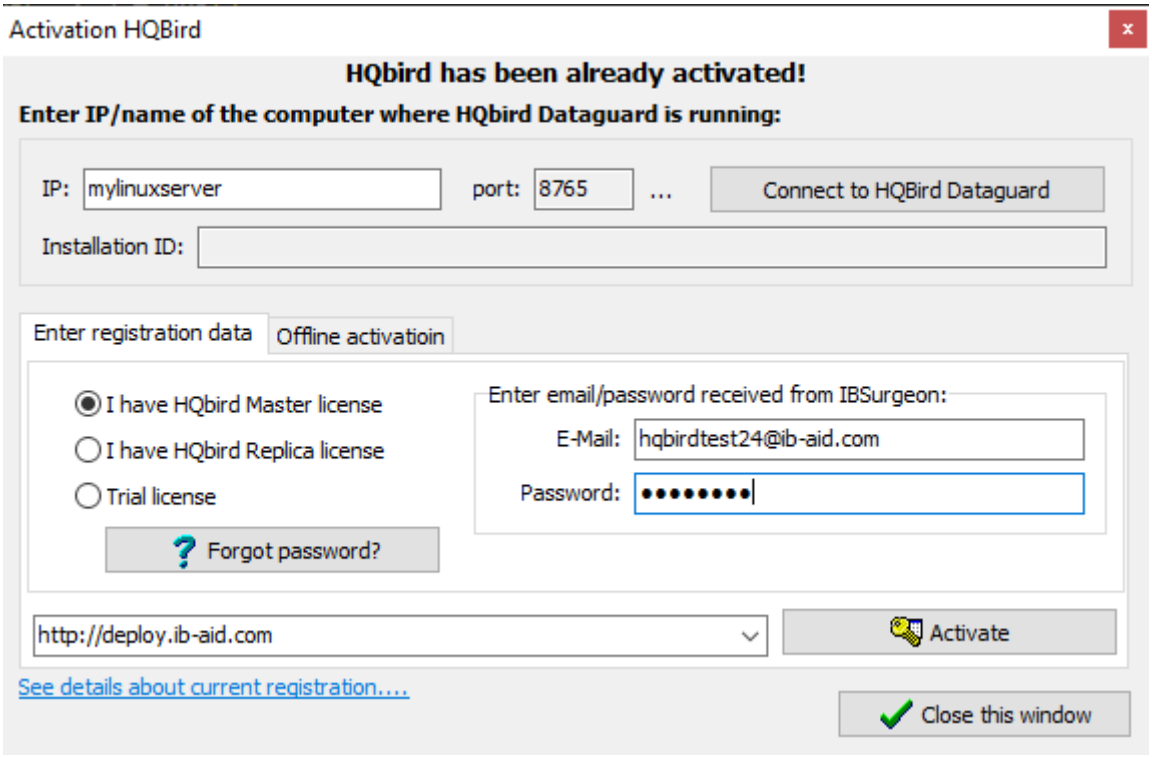



*Рисунок 24. Окно активации HQbird.*

### 2.6.2. Автономная активация

Если сервер и все клиентские компьютеры не имеют доступа к Интернету, следует использовать автономную активацию. Для этого перейдите на вкладку Offline activation и следуйте инструкциям. В случае возникновения проблем обращайтесь.

### 2.6.3. Активация в веб-интерфейсе

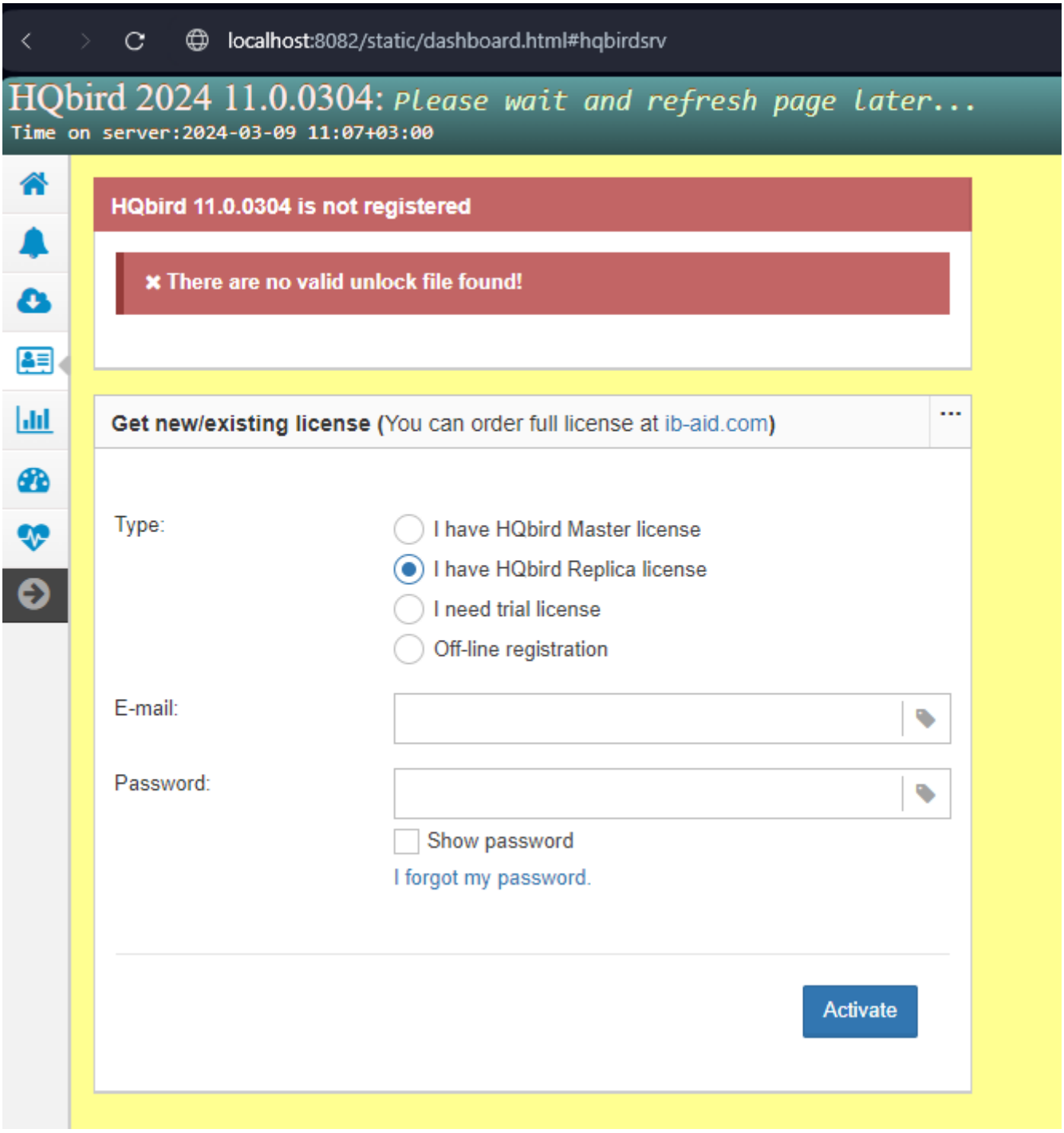



*Рисунок 25. Активация в веб-интерфейсе.*

## 2.7. Настройка firebird.conf для лучшей производительности

HQbird включает набор оптимизированных конфигурационных файлов для всех версий Firebird от 1.5 до 5.0 — они находятся в `HQbird\Configurations`.

Если вы не выполнили обоснованную настройку `firebird.conf` или используете `firebird.conf` с настройками по умолчанию, рассмотрите возможность использования одного из оптимизированных файлов из этой коллекции.

Существует три варианта конфигурационных файлов Firebird для каждой архитектуры Firebird: сбалансированный, с интенсивным чтением и с интенсивной записью. Мы всегда рекомендуем начинать со сбалансированного `firebird.conf`. Затем мы рекомендуем измерить фактическое соотношение между операциями чтения и записи с помощью инструмента HQbird MonLogger (вкладка “Aggregated Performance Statistics”). В 90% случаев операций чтения намного больше, чем операций записи, поэтому следующим шагом будет использование оптимизированного для чтения конфигурационного файла firebird.

Конфигурация Firebird сильно зависит от аппаратного обеспечения, поэтому если вы хотите правильно настроить Firebird, прочтите также “Firebird Hardware Guide”, это поможет вам понять, какие параметры должны быть настроены.

Для глубокой настройки высоконагруженных баз данных Firebird IBSurgeon предлагает услугу оптимизации базы данных Firebird: https://ib-aid.com/en/firebird-interbase-performance-optimization-service/

Кроме того, HQbird FBDataGuard анализирует состояние базы данных и отправляет оповещения с интеллектуальными предложениями по увеличению определенных параметров в `firebird.conf`, таких как `TempCacheLimit` или `LockHashSlots`.

**Внимание!**

Если вы указали много буферов страниц в заголовке вашей базы данных и установили SuperClassic или Classic, это может повлиять на производительность Firebird. Чтобы избежать потенциальной проблемы, установите для буферов страниц в заголовке вашей базы данных значение 0, это гарантирует, что будет использоваться значение из `firebird.conf`:

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

# Глава 3. Настройка заданий, мониторинга и обслуживания в FBDataGuard

Пожалуйста, выполните следующие шаги:

1. Убедитесь что у вас установлен Firebird 2.5.5 или новее, и он работает;
2. Служба HQbird FBDataGuard установлена и запущена
   a. Проверить это можно с помощью апплета “Службы” в Панели управления (щелкните правой кнопкой мыши “Мой компьютер”, выберите “Управление”, затем “Службы и приложения”, “Службы” и найдите в списке “Agent FBDataGuard”
   b. В Linux вы можете проверить это с помощью команды `ps aux | grep dataguard`.
3. Убедитесь, что порт FBDataGuard доступен (8082) и не заблокирован брандмауэром или другими антивирусными инструментами. При необходимости настройте порт в файле конфигурации FBDataGuard (см. Настройка порта веб-консоли).

## 3.1. Запуск веб-консоли

Чтобы открыть веб-консоль, введите в браузере `http://localhost:8082` или используйте IP-адрес вашего сервера с установленным HQbird ServerSide.

Или вы можете выбрать в меню “Пуск” **IBSurgeon › FBDataGuard**.

### 3.1.1. Поддерживаемые браузеры

Веб-интерфейс FBDataGuard корректно работает с Firefox, Chrome, Safari, Microsoft Edge и Internet Explorer 11.

### 3.1.2. Сообщение об ошибке сертификата веб-сайта

Если вы настроили FBDataGuard на использование https, браузер укажет, что этот сайт небезопасен, и предложит покинуть сайт. Это сообщение вызвано сертификатом безопасности по умолчанию для веб-консоли FBDataGuard.

Пожалуйста, проигнорируйте это сообщение и нажмите открыть веб-консоль FBDataGuard.

Он запросит у вас имя пользователя и пароль (диалог входа в систему может отличаться для Firefox, Safari или Chrome).

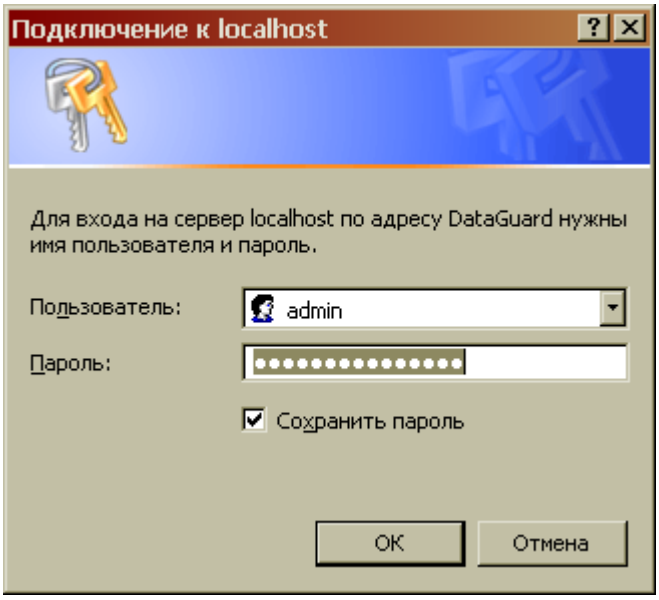



*Рисунок 26. Ввод логина и пароля для веб-консоли FBDataGuard.*

**Внимание!**

По умолчанию пользователь/пароль для HQbird FBDataGuard — "**admin**"/"**strong password**" (без кавычек).

### 3.1.3. Функция автоматического обнаружения FBDataGuard

При первом запуске FBDataGuard проверит компьютер на наличие установленных серверов Firebird. FBDataGuard для Windows ищет в реестре записи Firebird, FBDataGuard для Linux проверяет местоположения по умолчанию для установок Firebird.

FBDataGuard покажет список всех установленных копий Firebird, но FBDataGuard может отслеживать только один экземпляр Firebird. Выберите его, нажав **[ Добавить в мониторинг >> ]**

Если вы не видите экземпляр Firebird в списке автоматического обнаружения, вы можете выбрать **[ Добавить сервер вручную >> ]** и настроить параметры экземпляра вручную.

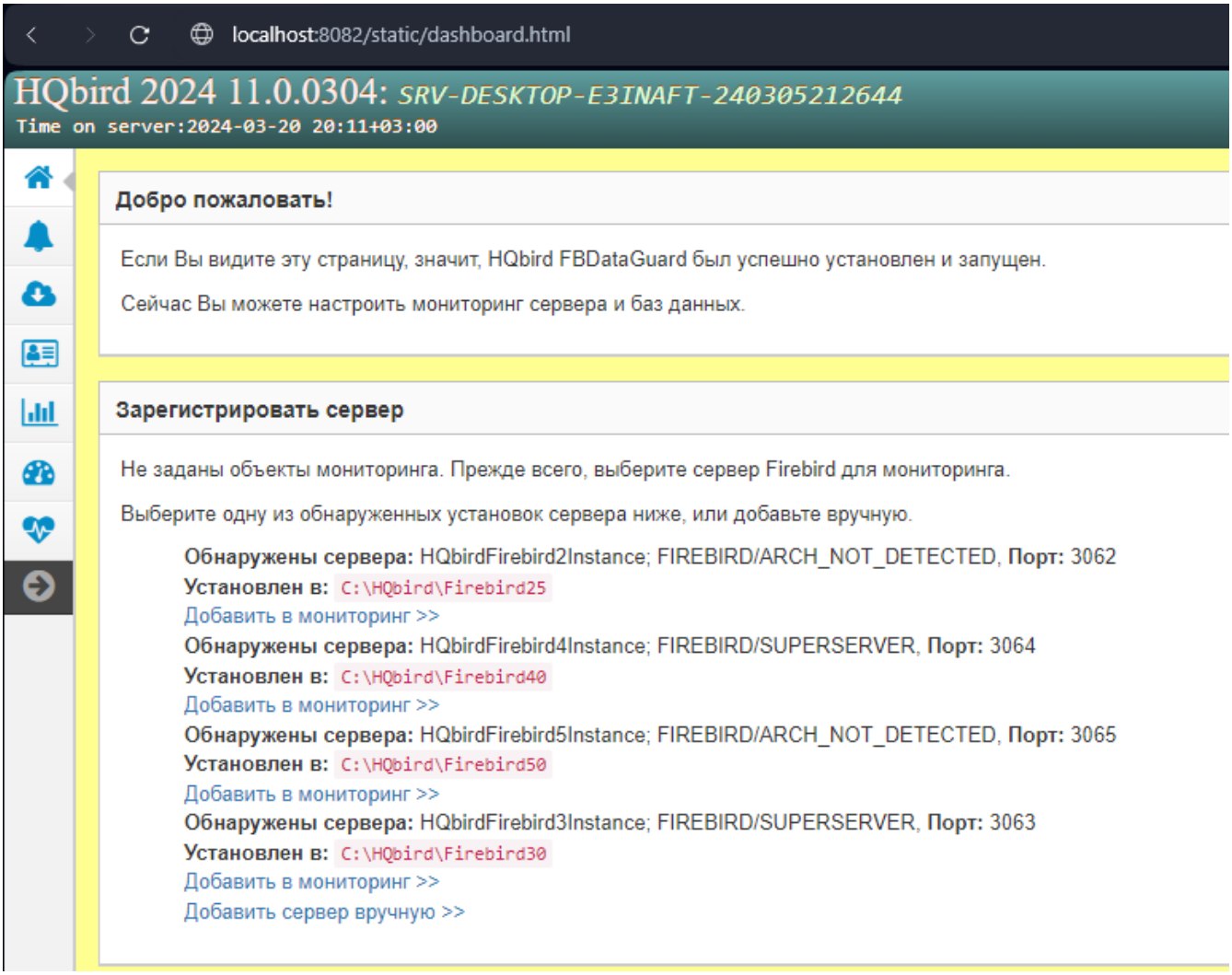
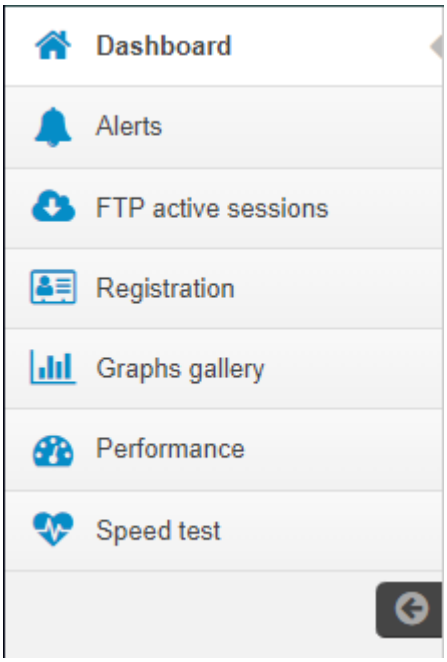



*Рисунок 27. Функция автоматического обнаружения HQbird.*

# 3.2. Обзор веб-консоли.

## 3.2.1. Части веб-консоли.




Веб-консоль FBDataGuard содержит 7 вкладок (в левой части экрана, обычно они свернуты):

- *Dashboard* — это основная вкладка, где администратор может настроить HQbird FBDataGuard и просмотреть статусы сервера и баз данных.
- *Alerts* — содержит полный список оповещений, созданных FBDataGuard.
- *FTP active sessions* — отображать информацию об активных FTP-сессиях.
- *Registration* — информация о лицензии и регистрации/активации.
- *Graphs gallery* — графики производительности.
- *Performance* — настройки мониторинга производительности и отчеты о производительности.
- *Speed test* — тест скорости

## 3.2.2. Задачи

Веб-консоль предназначена для удобной настройки действий (называемых **“задачи”**), которые выполняет HQbird FBDataGuard.

Почти все задания FBDataGuard имеют две цели: первая — отслеживать некоторые значения и при необходимости выдавать оповещения, а вторая — сохранять исторические значения в журналах, чтобы позже можно было увидеть динамику важных параметров сервера Firebird и базы данных.

В этом разделе мы рассмотрим общую настройку параметров заданий, а не анализ собранных логов.

### 3.2.3. Виджет задачи

Общий подход следующий: каждое действие представляется “виджетом”, который состоит из следующих частей:

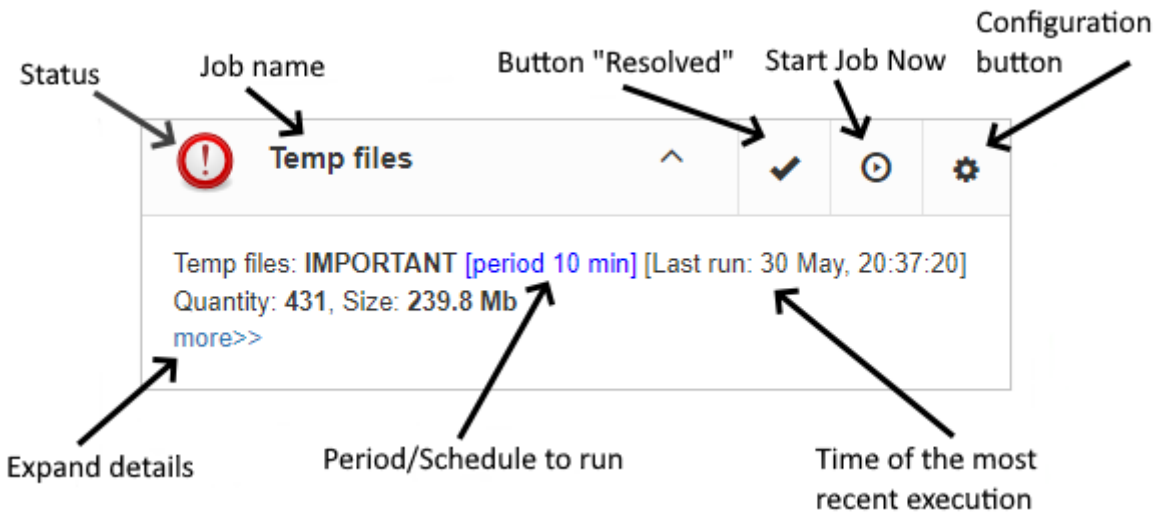



*Рисунок 28. Элементы виджета веб-консоли FBDataGuard.*

Статус — обозначается цветным значком и названием. Статус базы данных представляет собой сводку всех включенных заданий уровня сервера и статусов баз данных, и, соответственно, статус базы данных представляет собой сводку всех заданий уровня базы данных.

### 3.2.4. Типы статусов

**CRITICAL** означает проблемы, **OK** означает “Все в порядке”, **WARNING** означает некоторые проблемы, требующие внимания, **MAJOR** означает серьезную проблему, **MINOR** – незначительная проблема, **MALFUNCTION** означает, что задание не удалось выполнить (что-то препятствует его выполнению), **NOT_AVAILABLE** означает, что задание недоступно в этой версии сервера или базы данных.

**OFF** означает, что задание не активно, **UNKNOWN** означает, что задание активно, но еще не запущено, поэтому фактические результаты неизвестны.

**Job name** — имя задачи.

- ⚙ Кнопка конфигурации открывает диалог настройки, индивидуальный для каждого задачи.
- ⊙ Запускает задачу немедленно.
- ✔ Resolved — это ссылка для сброса статуса на UNKNOWN и забывания ошибок, которые были обнаружены ранее. Статус будет обновлен в соответствии с текущей ситуацией после следующего выполнения задания.

**Last run** показывает время после последнего запуска этого задания.

**Period/Schedule to run** показывает, как часто и когда будет запускаться задание.

**More** >> — это ссылка, которая открывает виджет и показывает более подробную информацию и предлагаемые действия администратору для разрешения ситуации.

Все задания в FBDataGuard имеют настройки по умолчанию, которые очень близки к рекомендуемым значениям для 80% установок Firebird, поэтому после первоначальной настройки сервер и база данных будут защищены на довольно хорошем уровне по сравнению с установкой по умолчанию, однако мы рекомендуем дополнительную настройку для каждой задачи. В следующих разделах мы рассмотрим каждое задание и его настройку.

# 3.3. Конфигурация сервера Firebird в FBDataGuard

## 3.3.1. Регистрация сервера Firebird

Чтобы зарегистрировать автоматически обнаруживаемый сервер, вам нужно нажать **[ Добавить в мониторинг >> ]**, а затем настроить параметры автоматического обнаружения.

Примечание. Чтобы использовать доверенную аутентификацию Windows (по умолчанию она отключена), вам необходимо быть уверенным, что библиотеки `jaybird30.dll` и `fbclient.dll` (из соответствующей версии Firebird) находятся в доступных для поиска путях Windows.

При установке под Windows, если выбрана опция автоматической регистрации master/replica, то сервер будет добавлен автоматически. В этом случае этот шаг можно пропустить. Если выбран вариант автоматической регистрации реплики, то база данных будет добавлена дополнительно.

Давайте рассмотрим, что вы можете увидеть в диалоге Server (как правило, вам не нужно ничего менять):

| | |
|---|---|
| **Установлен в папке** | Папка установки Firebird |
| **Папка Bin для Firebird** | Папка исполняемых файлов Firebird (для Firebird 3 и выше в Windows она совпадает с папкой установки) |
| **Log** | расположение `firebird.log` |
| **Configuration file** | расположение `firebird.conf` |
| **Aliases** | расположение `aliases.conf` или для Firebird 3 и старше `databases.conf` ( **пожалуйста, измените его вручную, если необходимо** ) |
| **Хост** | имя или IP-адрес сервера, обычно `localhost` |
| **Порт** | порт для Firebird, согласно настройке `firebird.conf` |
| **Использовать trusted auth** | use trusted authentication, by default it is off |
| **Логин** | имя пользователя (администратора), обычно это SYSDBA |
| **Пароль** | пароль пользователя (пароль SYSDBA) |
| **User for Services API** | имя пользователя для служб Firebird. Обычно совпадает с **Логин**. |
| **Password for Services API user** | пароль пользователя для служб Firebird. |
| **Server authentication plugin list** | список плагинов аутентификации. |

**Папка результатов** — Папка, в которой будут храниться резервные копии, статистика и собранные журналы.

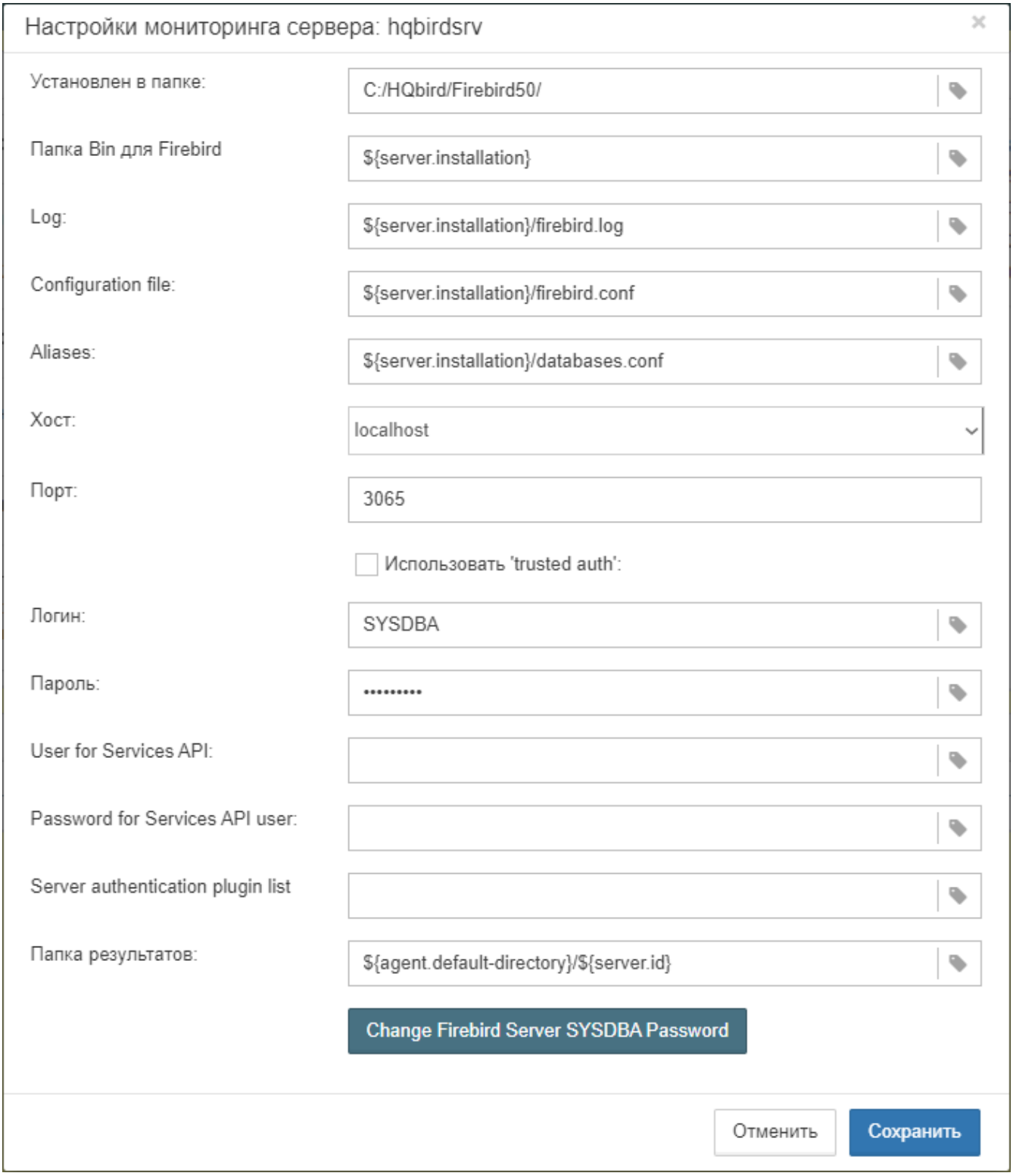



*Рисунок 29. Регистрация сервера в HQbird FBDataGuard.*

По умолчанию “Папка результатов” для сервера Firebird — `${agent.default-directory}/${server.id}`, она соответствует `C:\HQbirdData` в случае установки по умолчанию.

Это может быть не очень удобно, поэтому мы рекомендуем указать выходной каталог FBDataGuard по более простому пути, обычно расположенному на диске, где предполагается хранить резервные копии, например `F:\myserverdata`.

После нажатия кнопки “Сохранить” FBDataGuard заполнит файлы конфигурации по умолчанию и немедленно начнет анализ `firebird.log`. Это может занять некоторое время (например, 1 минута для `firebird.log` размером 100 МБ). После этого вы увидите начальную веб-консоль с зарегистрированным сервером Firebird:

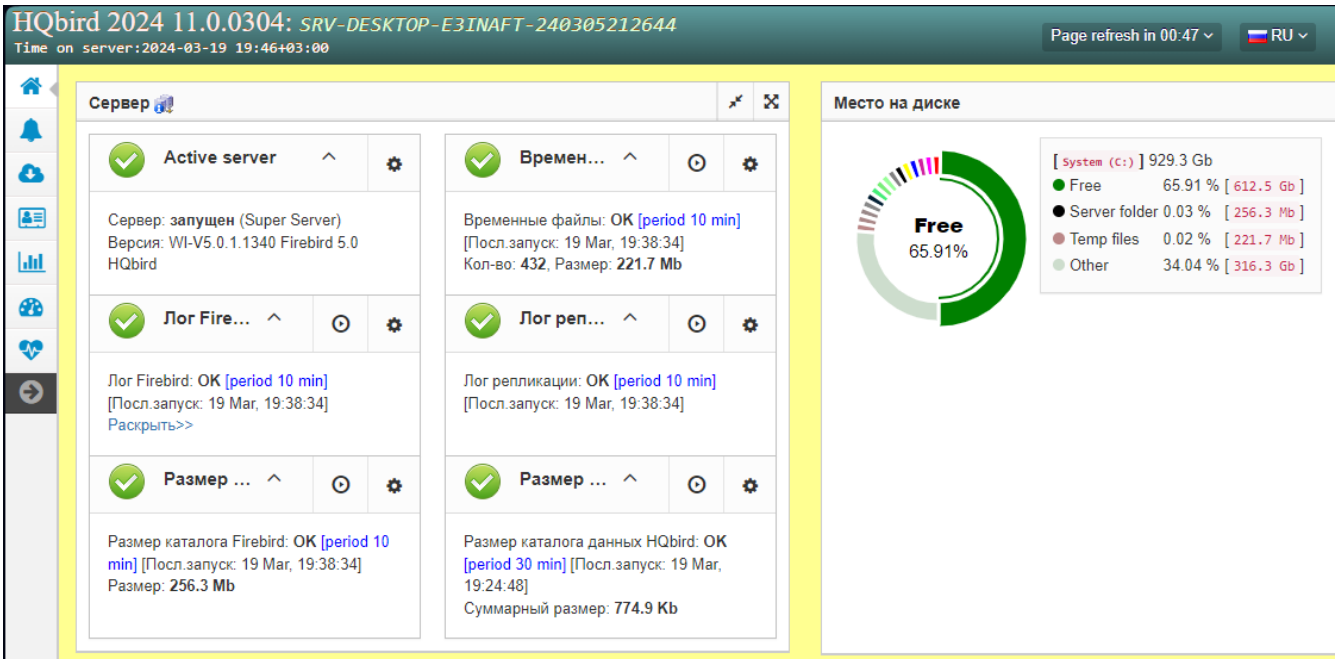



*Рисунок 30. HQbird FBDataGuard с зарегистрированным сервером Firebird.*

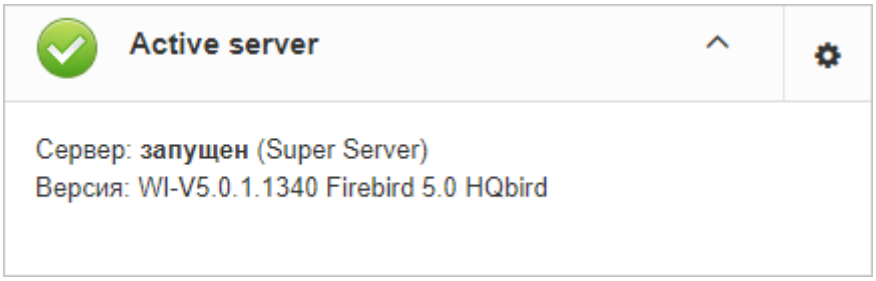
FBDataGuard показывает оповещения и статусы контролируемых объектов: если все в порядке, он показывает зеленые знаки, в противном случае будут желтые или красные уведомления.

Ниже мы подробно рассмотрим каждый контролируемый объект и его настройки.

Примечание: вы не можете удалить зарегистрированный сервер Firebird в веб-консоли FBDataGuard. Единственный способ отменить регистрацию сервера — удалить его файлы конфигурации. В общем, нет смысла удалять зарегистрированный сервер, пока вы не захотите полностью удалить FBDataGuard.

## 3.3.2. Сервер: Active server

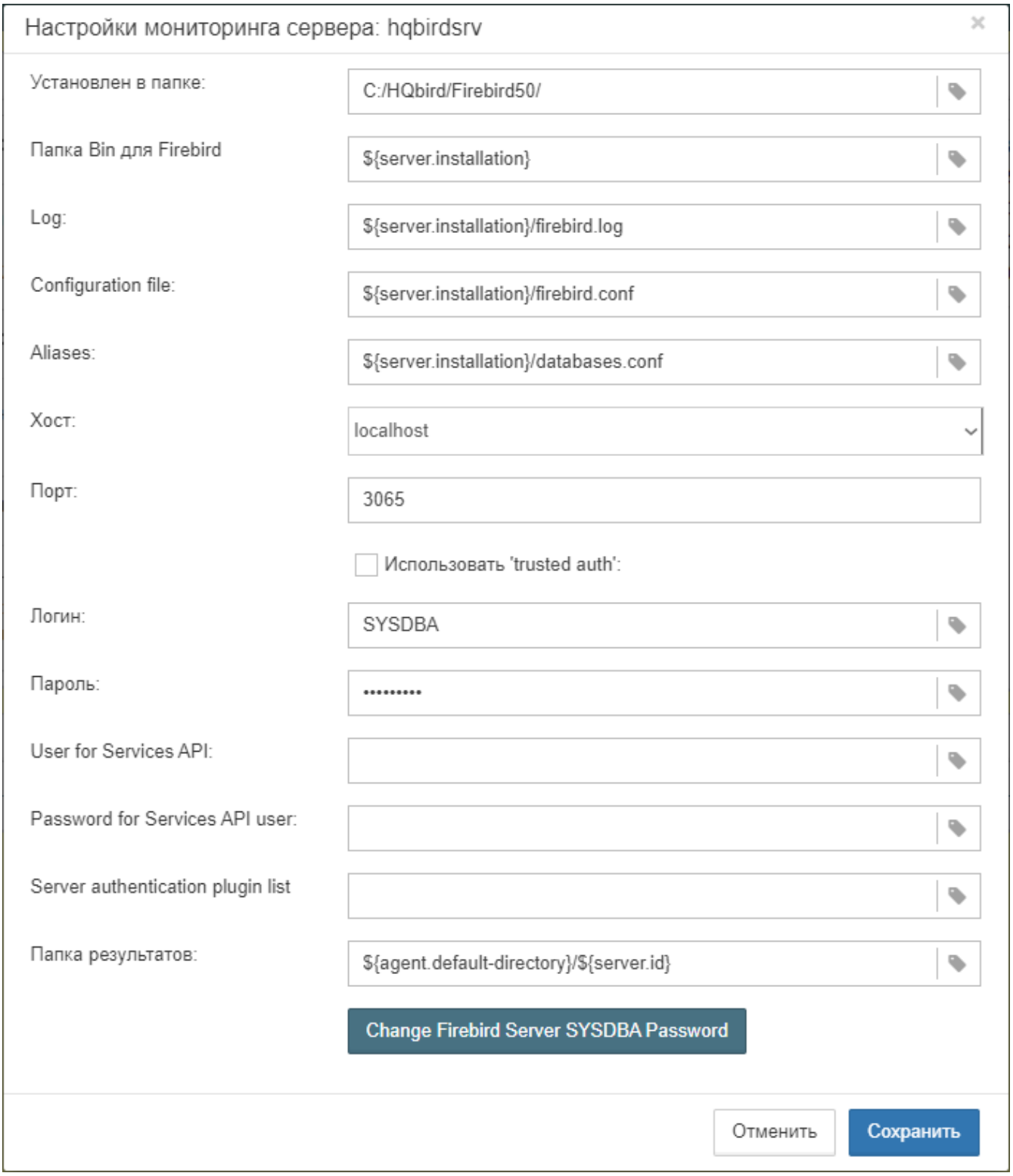
Сервер: Виджет Active server отображает сводный статус всех заданий уровня сервера и статусы отслеживаемых баз данных.




**Сервер: Active server** также указывает, работает ли Firebird в данный момент или нет, и

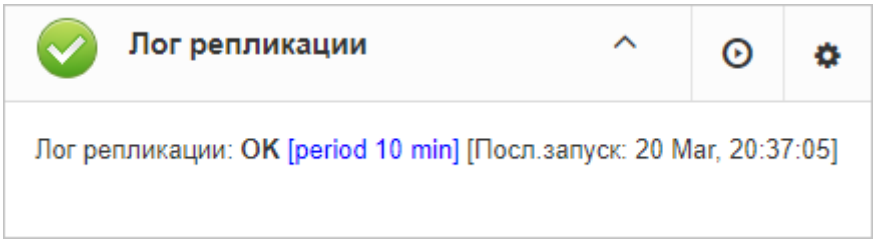
показывает подробную версию Firebird и HQbird.

Если вы нажмете ссылку **Настройка**, то увидите тот же диалог, который мы использовали для регистрации экземпляра Firebird в FBDataGuard, и теперь его можно использовать для изменения свойств экземпляра Firebird:



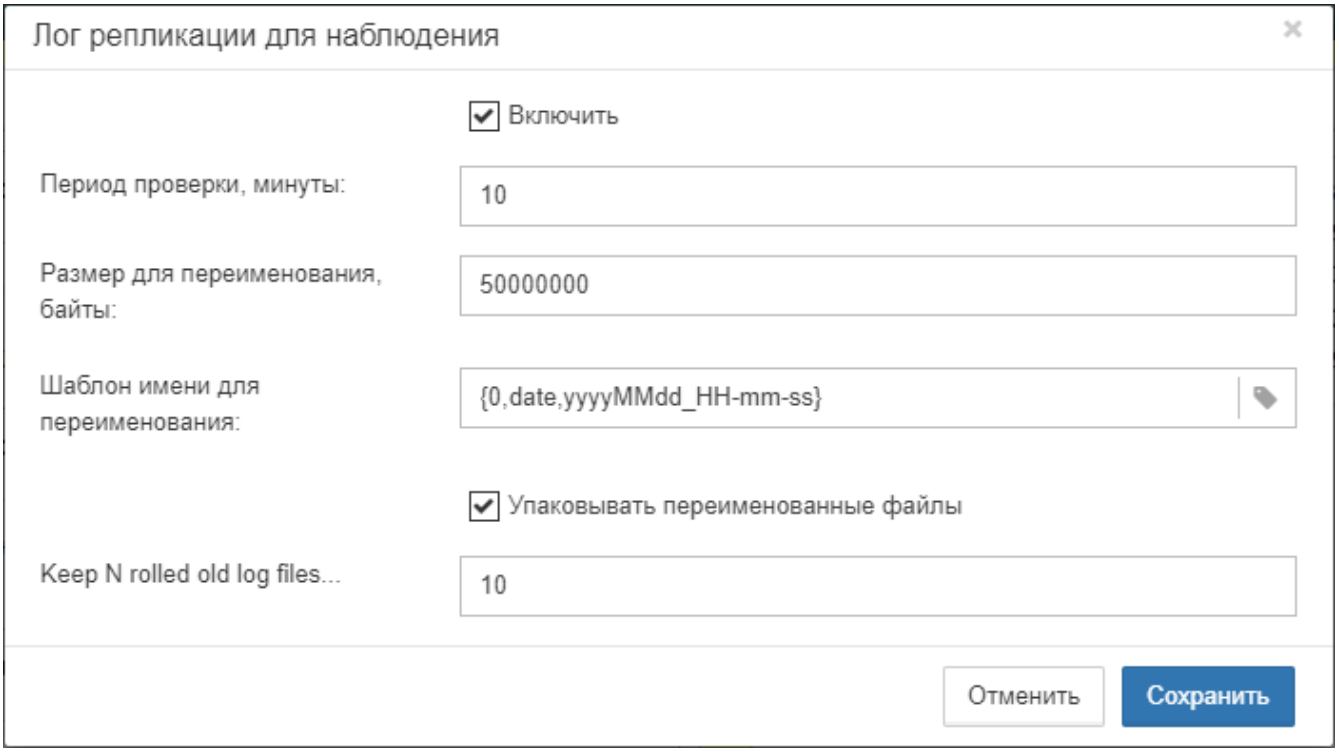

В общем, нет необходимости редактировать данные сервера Firebird после регистрации, пока вы не переустановите Firebird — но в этом случае мы рекомендуем также переустановить HQBird.

### 3.3.3. Сервер: Лог репликации



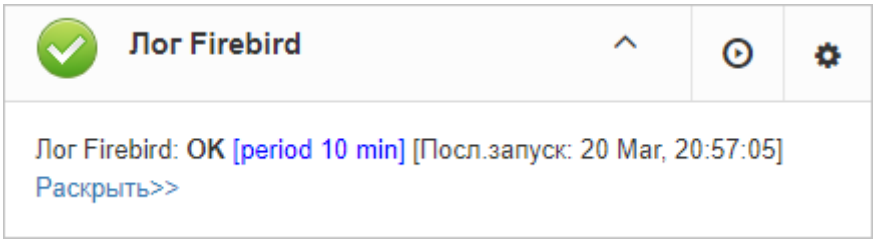

FBDataGuard проверяет `replication.log` на наличие ошибок. В случае ошибки он отправляет соответствующее предупреждение (по электронной почте) администратору.

Чтобы включить это задание, установите флажок “Включить”.




- “Период проверки, минуты” — как часто проверять файл `replication.log` на наличие изменений.
- “Размер для переименования, байты” — если `replication.log` превысит значение, он будет переименован в соответствии с шаблоном даты и времени.
- “Шаблон имени для переименования” — как переименовать `replication.log`
- “Keep N rolled old log files” — сколько ошибок будет храниться в списке последних ошибок.

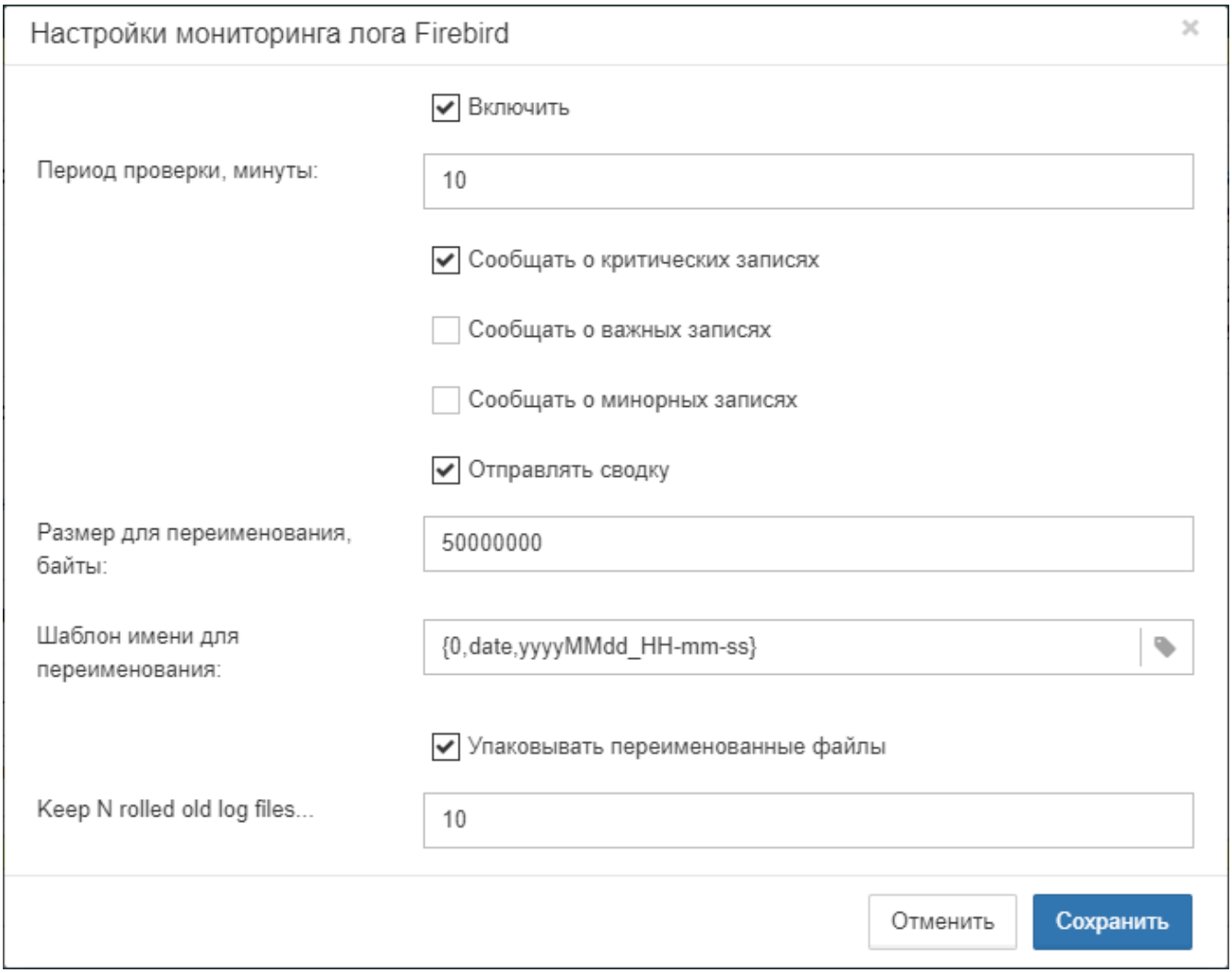
### 3.3.4. Сервер: Лог Firebird



Задание “Лог Firebird” периодически проверяет `firebird.log`, и если обнаруживает, что файл был изменен, начинается анализ журнала. Встроенный аналитический механизм проверяет каждую запись в файле `firebird.log` и классифицирует их по нескольким категориям с разными уровнями серьезности. В зависимости от серьезности сообщений назначается статус задания и генерируются соответствующие оповещения.

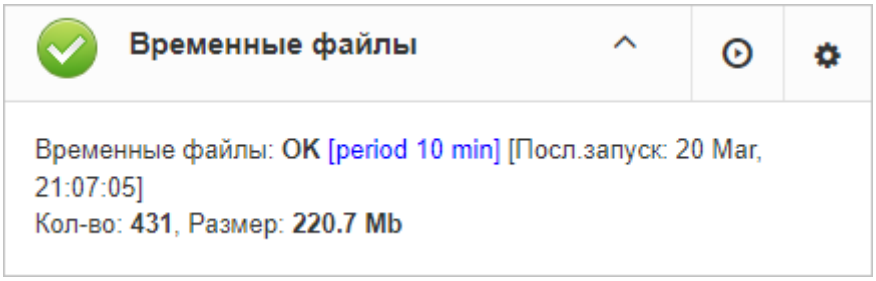
После того, как администратор просмотрит ошибки и предупреждения (и выполнит необходимые действия для устранения причины ошибки), ему необходимо нажать на ссылку **“Исправлено”**, и FBDataGuard забудет старые сообщения об ошибках в `firebird.log`.

В диалоге настройки “Лог Firebird” вы можете включить/отключить это задание и установить период проверки (в минутах).




Также это задание отслеживает размер `firebird.log`, и если его размер превышает "Размер для переименования", FBDataGuard разделит `firebird.log` и переименует его в соответствии с шаблоном даты и времени.

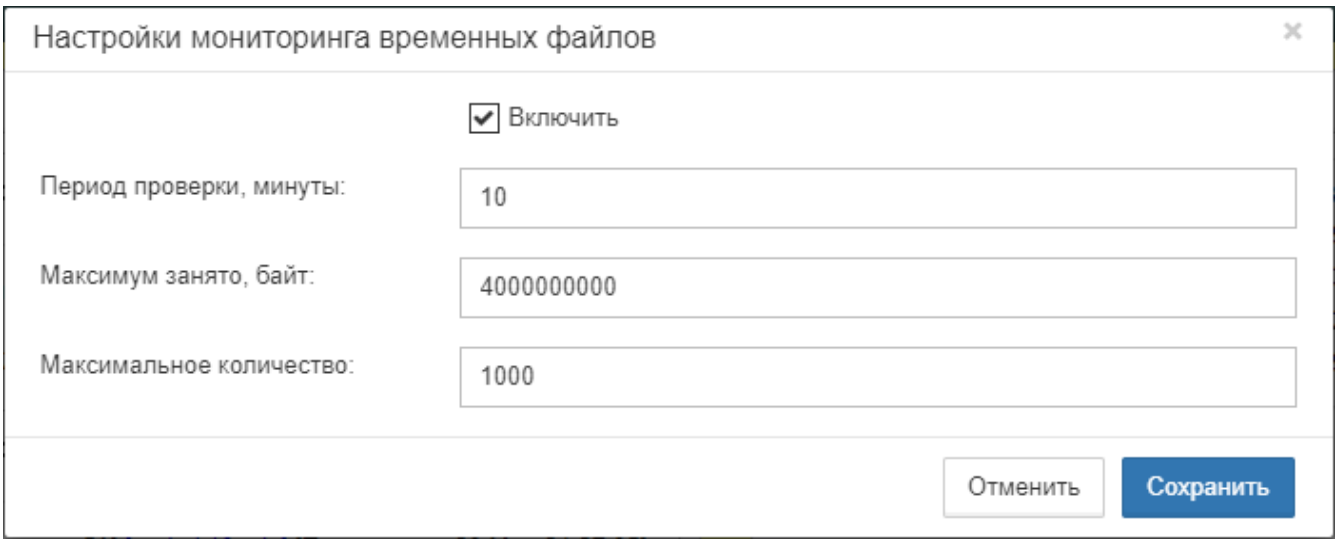
### 3.3.5. Сервер: Временные файлы




Задание “Server: Temp files” полезно для обнаружения и решения проблем с производительностью базы данных Firebird.

При выполнении SQL-запросов Firebird сохраняет промежуточные результаты сортировки и объединения потоков данных во временных файлах, которые размещаются в папках `TEMP`. FBDataGuard показывает в виджете “Сервер: Временные файлы” информацию о количестве и размере временных файлов.

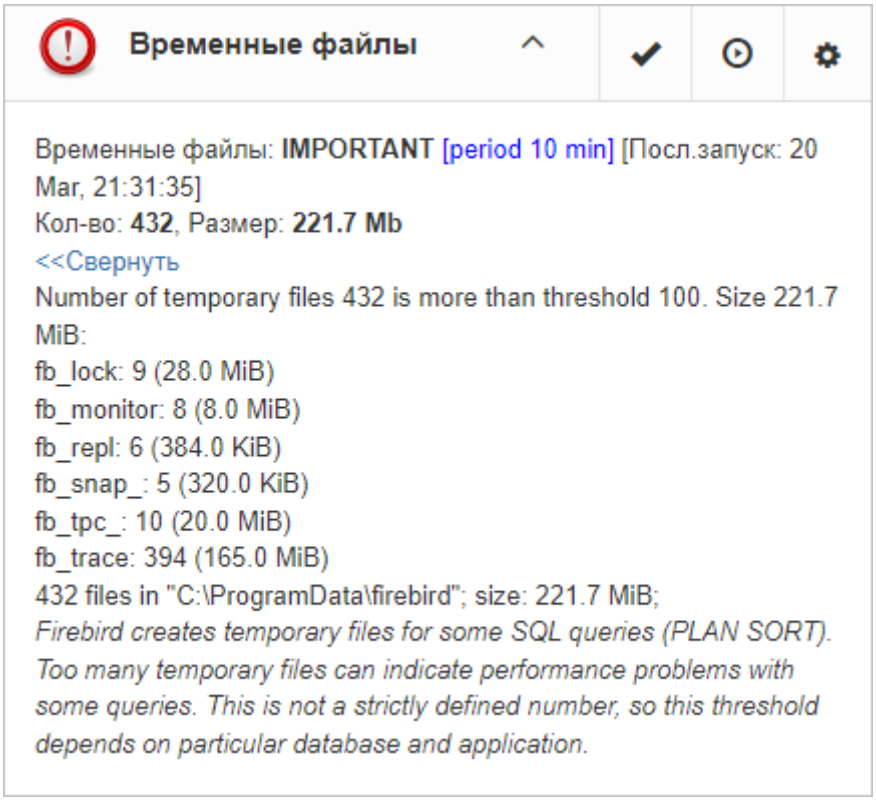
FBDataGuard распознает расположение папок `TEMP` и отслеживает количество и размер временных файлов. Нехватка места может привести к проблемам с производительностью или более серьезным ошибкам, слишком большое количество (или слишком большие) временных файлов может указывать на проблемы с качеством SQL-запросов.




Используя диалог конфигурации, вы можете включить/отключить это задание, установить период проверки и пороговые значения для максимального размера временных файлов (размера всех файлов) и количества.

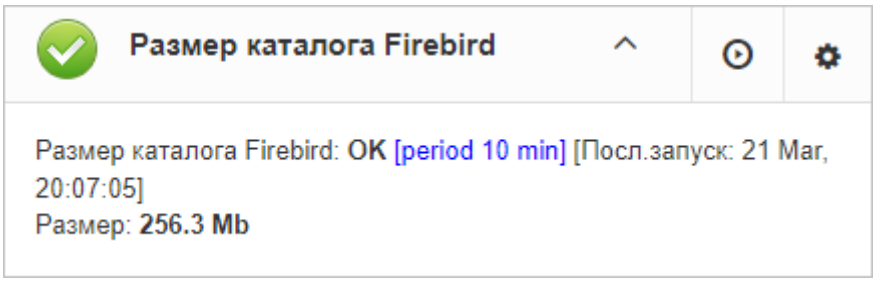
Если вы видите, что размер временных файлов слишком велик и на сервере достаточно оперативной памяти, увеличьте параметр `TempCacheLimit` в `firebird.conf`, чтобы все временные таблицы поместились в оперативную память.

Кроме того, HQbird проверяет другие временные файлы, используемые Firebird — если вы видите экстремальные значения (несколько ГБайт) для трассировки или мониторинга, хорошей идеей будет проверить папку `FIREBIRD_TMP` на наличие устаревших файлов (со старыми метками времени модификации). Обратите внимание: снимок экрана ниже не является настоящим предупреждением (т. е. значение OK), он был создан для демонстрации вывода в случае больших временных файлов.




### 3.3.6. Сервер: Размер каталога Firebird

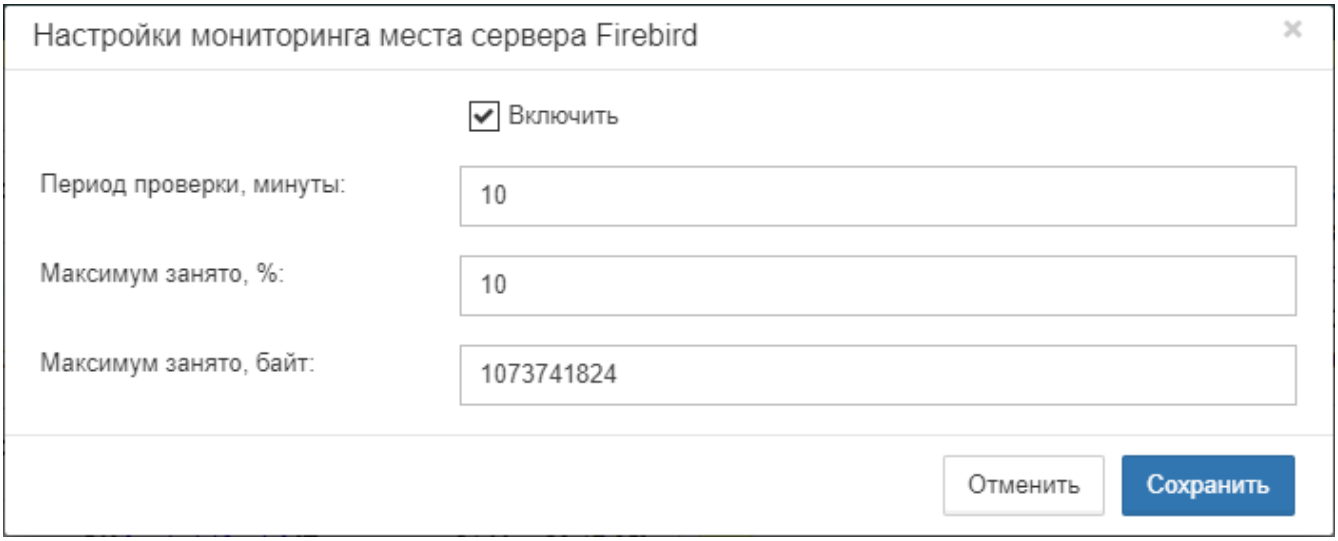
Задание “Размер каталога Firebird” отслеживает размер, занимаемый установкой Firebird. Оно включен по умолчанию.




Это задание предотвращает несколько угроз: проблемы неправильного администрирования при создании томов базы данных или внешних таблиц в папке `%Firebird%\Bin`, очень большой файл `firebird.log`, который может исчерпать все места на диске с установленным Firebird, и некоторые другие проблемы.

Также это задание отслеживает и анализирует информацию, собранную всеми заданиями, связанными со свободным пространством (включая задания на уровне базы данных). На рисунке ниже вы можете увидеть краткое представление анализа пространства для всех дисков, на которых хранятся базы данных Firebird и резервные копии.

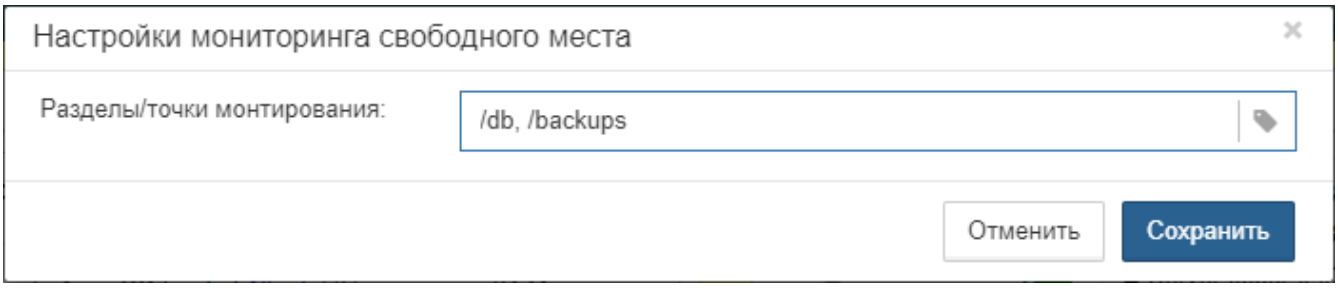
С помощью диалога конфигурации вы можете включить/отключить это задание, установить период проверки и пороговые значения размера папки сервера.




По умолчанию мы используем 1 Гб — это стандартная настройка для установки Firebird.

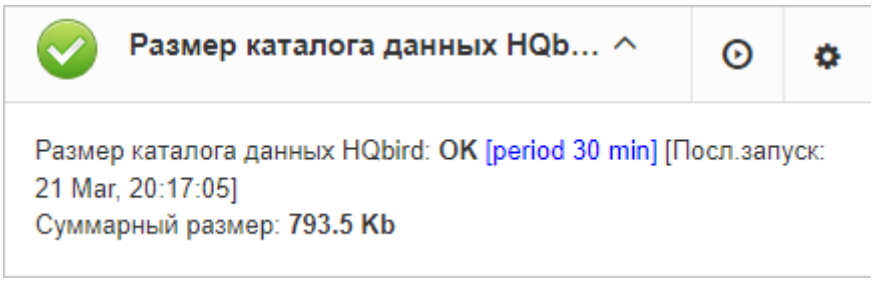
Если размер вашего Firebird больше, рассмотрите возможность очистки старых журналов и других нежелательных артефактов или увеличьте параметр **Максимум занято** (в байтах), чтобы предотвратить ложные оповещения.

**Примечание для пользователей Linux**: если вы видите красное предупреждение с противоречивой информации о свободном пространстве, добавьте каталоги с базой данных и резервными копиями в виджете “Место на диске”:




Вы можете получить представление о том, где находится ваша база данных и резервная копия, с помощью команды `df -h`.

### 3.3.7. Сервер: Размер каталога данных HQbird




Мониторинг “Размер каталога данных HQbird” предназначен для наблюдения за пространством, занимаемым отчетами, журналами, статистикой, хранилищем метаданных и другими данными, собранными и сгенерированными HQbird — по умолчанию это папка `C:\HQbirdData\output`.

Для баз данных, находящихся без присмотра в течение длительного времени (1-2 года),

возможно, что журналы FBDataGuard займут слишком много места, а нехватка места может привести к сбою базы данных. Чтобы наверняка это предотвратить, задание “Размер каталога данных HQbird” отслеживает занятое место.

По умолчанию задание “Размер каталога данных HQbird” включено.

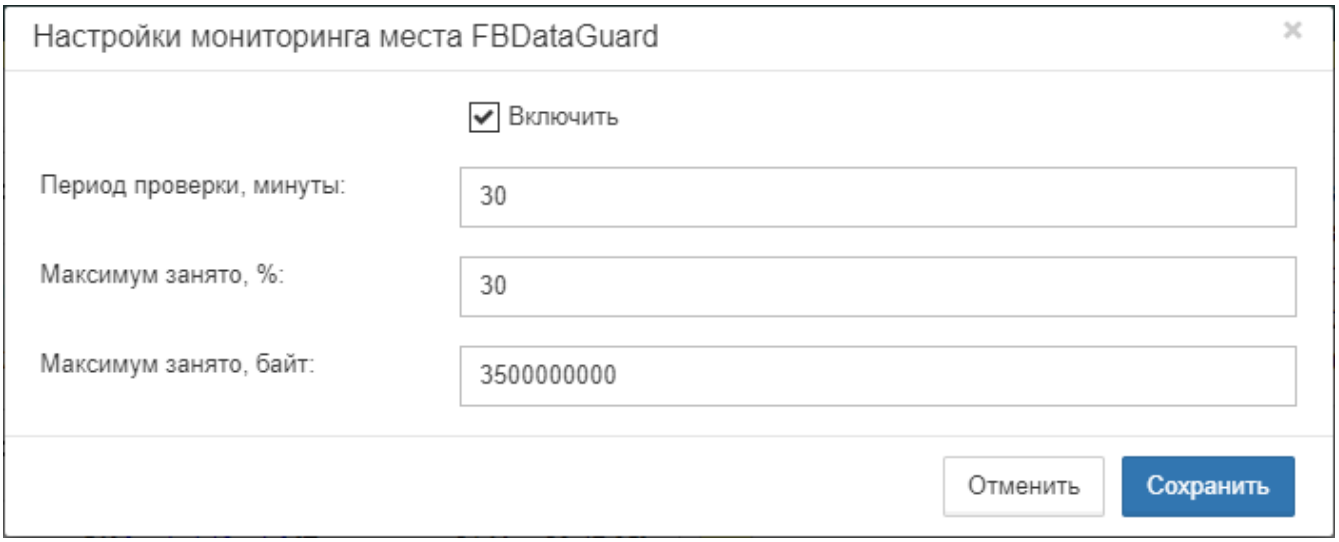
Кроме того, если кто-то проигнорировал рекомендации по размещению папок резервных копий в определенных местах, вполне возможно, что резервная копия базы данных будет создана внутри папки Агента. В этом случае вы сразу увидите CRITICAL статус — FBDataGuard распознает это и предупредит вас о неправильной конфигурации.

Это задание полезно для связок FBDataGuard и сторонних приложений.

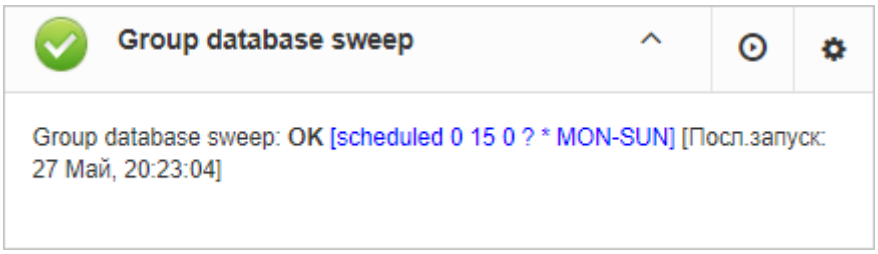
В диалоге конфигурации вы можете включить/отключить это задание, установить период проверки (по умолчанию 10 минут) и установить пороговые значения для оповещений.

Пороговые значения могут быть установлены в % от максимального размера, занимаемого журналом, или с указанием явного размера в байтах.

FBDataGuard проверяет оба значения и выдает предупреждение для первого порога. Если вы хотите установить только %, вам нужно установить -1 в качестве значения «Максимум занято, байт».

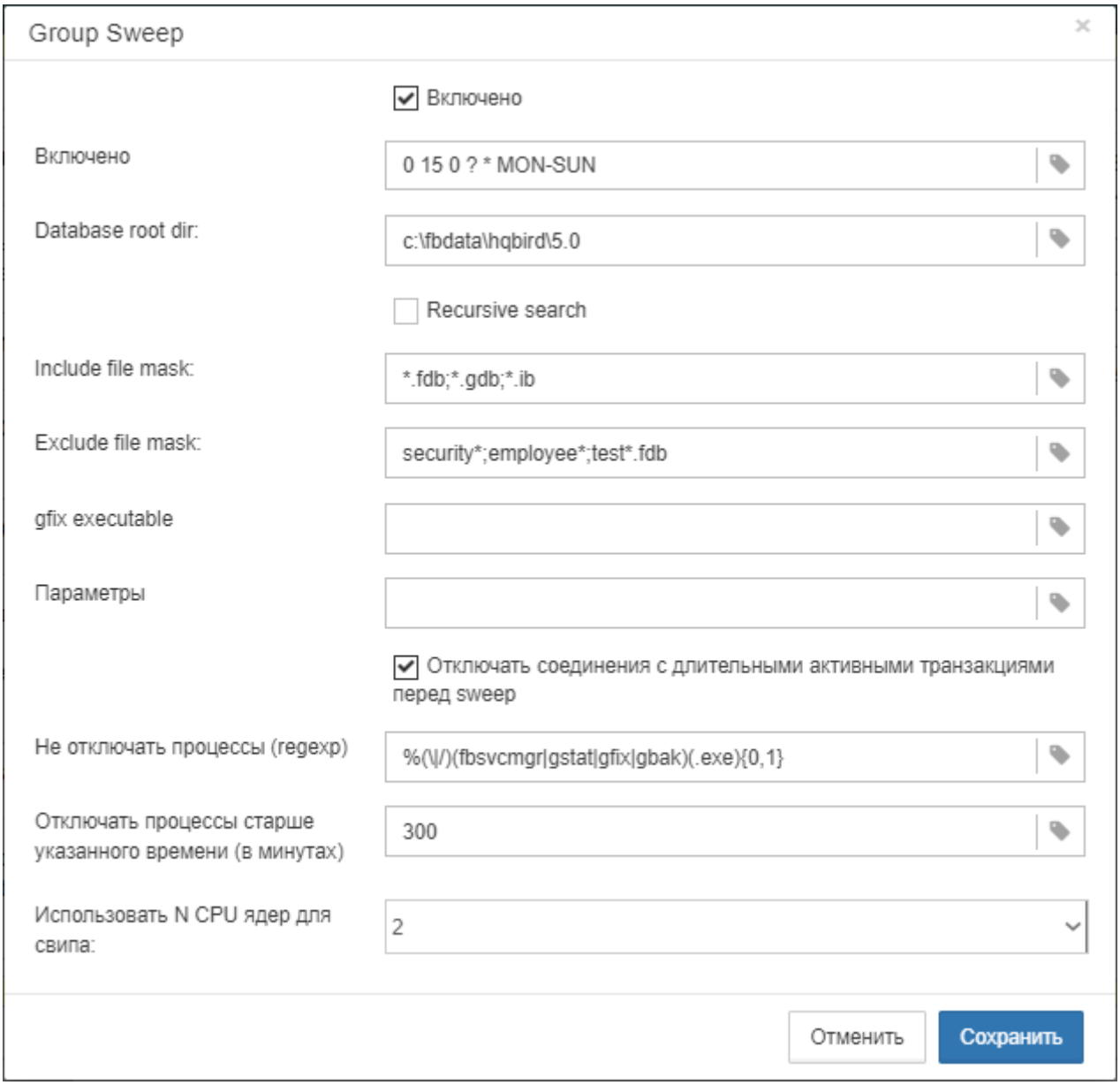



### 3.3.8. Сервер: Group sweep




Задание “Group sweep” полезно в том случае, если на одном сервере работает много баз данных. В этом случае задание “Group sweep” позволяет позволяет выполнять интеллектуальный Sweep для всех баз данных расположенных по указанному пути включая подпапки.

В диалоге настройки “Group sweep” можно указать, где расположены базы данных, настроить маску включения и/или исключения для выполнения sweep только для определённых баз данных, а также настроить время запуска процесса.




Задание “Group sweep” имеет те же параметры, что и База данных: Sweep, а также следующие дополнительные параметры:

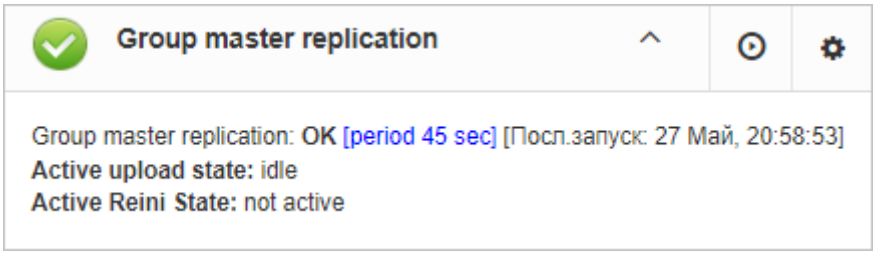
- “Database root dir” — указывает корневую директорию в которой производится поиск баз данных для выполнения Sweep.
- “Recursive search” — если этот флажок установлен, то поиск базы данных происходит рекурсивно во всех подпапках корневой папки.
- “Include file mask” — регулярное выражения для включения файла БД в список группового Sweep.
- “Exclude file mask” — регулярное выражения для исключения файла БД из списка группового Sweep.

### 3.3.9. Сервер: Group master replication

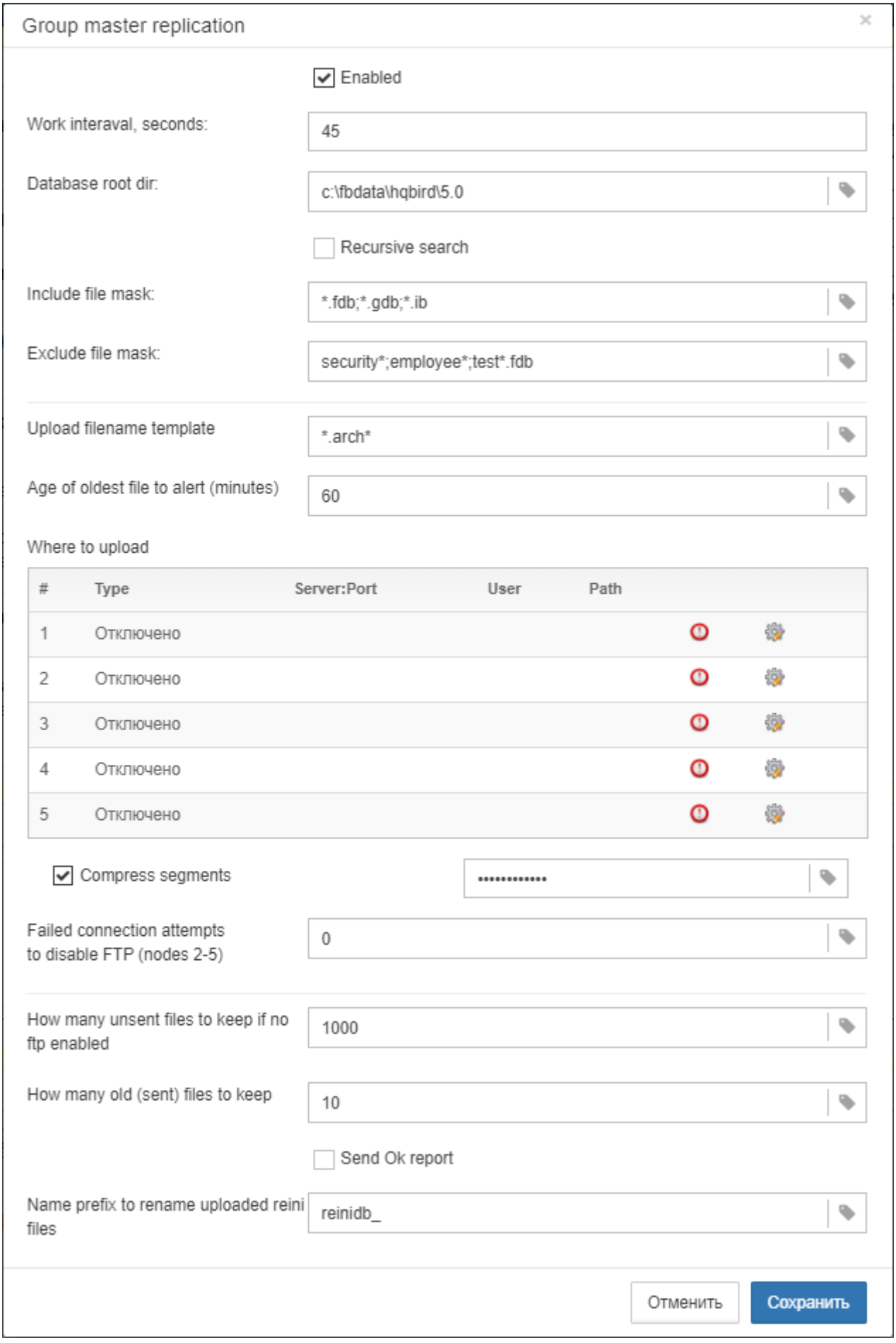
Задача “Group master replication” предназначена для упрощения настройки репликации множества баз данных (может достигать несколько сотен). Это множество баз (в виде группы) интегрировано в DataGuard, с сохранением действующего функционала DataGuard и его индивидуальных настроек баз данных (их индивидуальных настроек репликации и задач обслуживания). Зарегистрированные в DataGuard “индивидуальные” базы данных, подразумеваются как “очень важные базы данных”, а для баз данных зарегистрированные в “Group master replication” выполняются только задачи по переносу сегментов репликации на другие сервера.

Одна из причин создания отдельной задачи “Group master replication” состоит в том, что в текущей реализации DataGuard есть ограничения с количеством обрабатываемых баз данных. Это ограничение в основном связано с ограничениями браузера, обрабатывающего логику веб-консоли. Для “Group master replication” такого ограничения не обнаружено. Ещё одна причина заключается в сложности настроек множества баз при их индивидуальной регистрации.

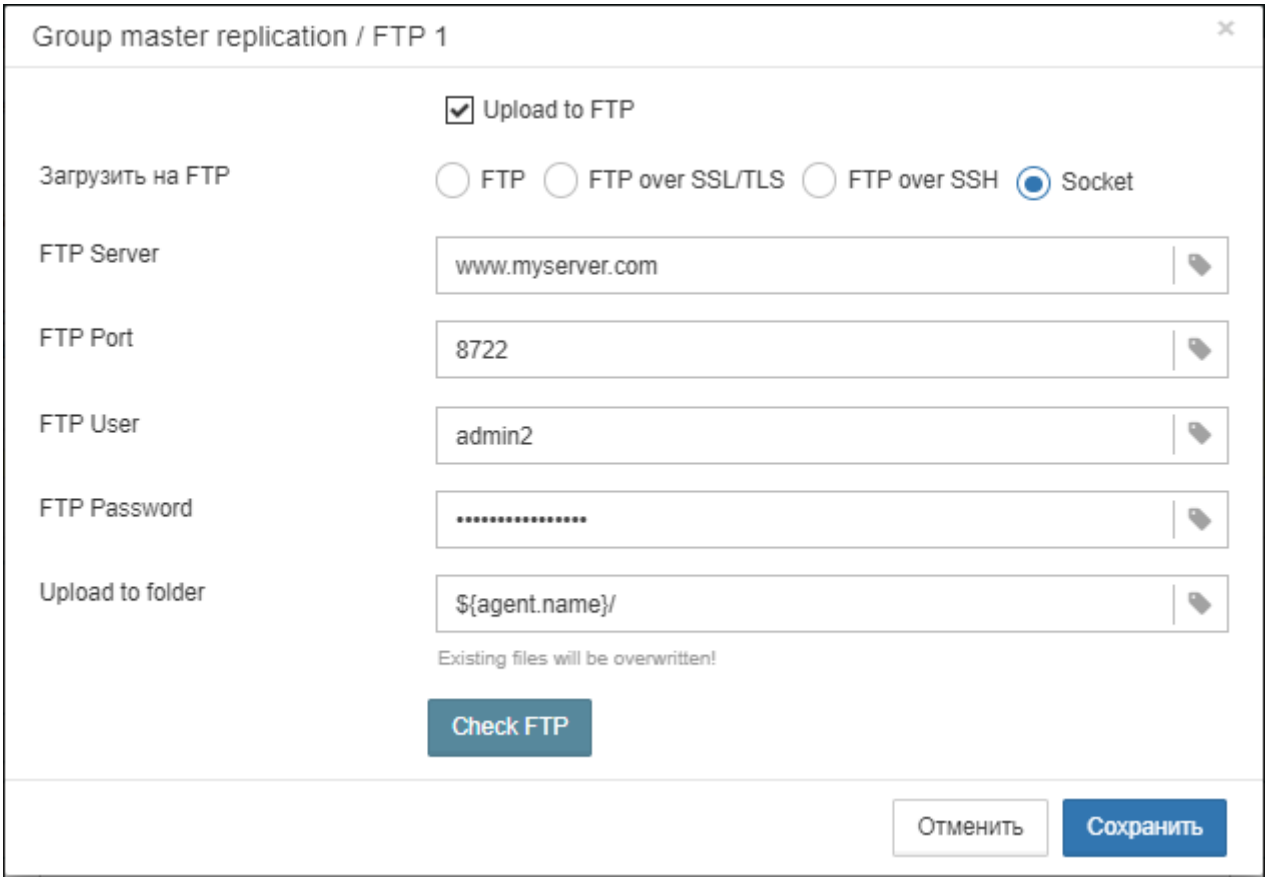
Настройка задачи “Group master replication” состоит из нескольких частей.




В первую очередь необходимо задать общие параметры для всех задач: где лежат базы, по какому шаблону искать подходящие базы, куда и как отправлять их сегменты (протокол отправки, адреса, шифрование) и как часто сканировать каталоги с сегментами.




Можно указать до пяти разных узлов — для каждого узла указываются типовые (как в задаче отправки сегментов) параметры:

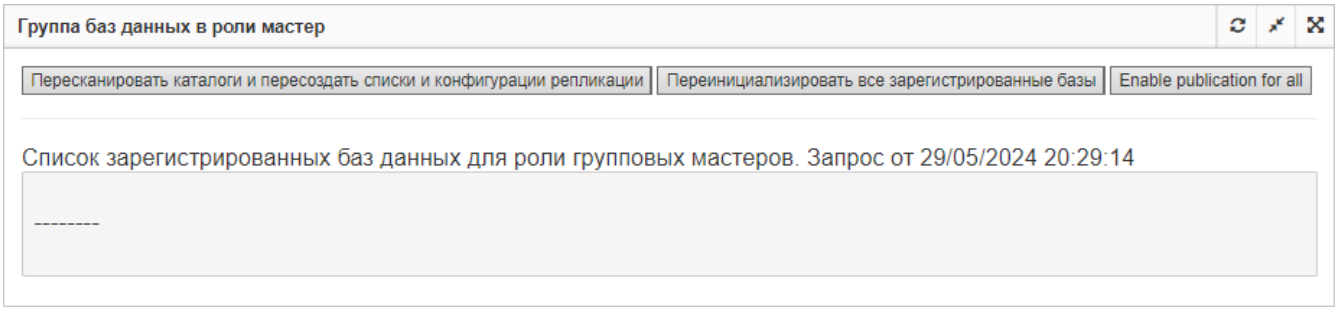



Замечания:

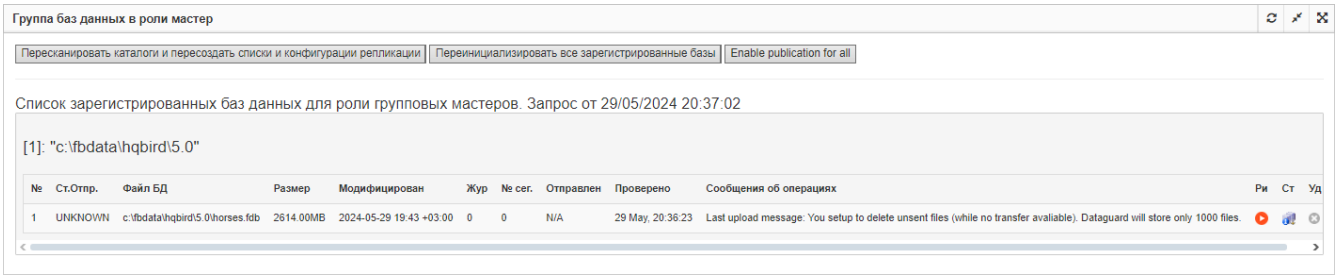
- Узел реплики может быть один, а серверов мастеров — несколько. Поэтому чтобы избежать перезаписи сегментов с абсолютно одинаковыми путями и именами баз, но приходящих разных узлов мастеров на реплике корневая папка загрузки для узла мастера выбрана и создаётся (по умолчанию) в виде имени агента (макроподстановка). Администратор волен её поменять, но должен учесть необходимость сохранения уникальности, если узлов мастеров будет несколько.
- DataGuard мульти-реплика умеет сам “разруливать” множественные источники на своём ФТП/Сокет каталоге и умеет втягивать и автоматически регистрировать вновь появившиеся базы (для облегчения последующего обслуживания также оперируя идентификатором имени узла отправки).
- DataGuard мульти-мастер для каждой отправляемой базы будет создавать файлы на узлах реплик в соответствии с расположением баз источников, и если файл базы данных на мастере называется одинаковым именем, но лежит в разных каталогах, то это не создаст проблемы.

Как и у обычной базы, если FTP отгрузка не настроена (но база уже в мастере, а значит генерит сегменты), то в действие вступает фильтр ограничение — сколько “неотправленных” сегментов разрешено хранить на диске. Сегменты сверх лимита будут удаляться на каждой итерации задачи. Так сделано, чтобы можно было без боязни переполнения диска включать базу мастером, а потом настраивать выгрузку (и в финале проводить реинициализацию для отправки на реплику).

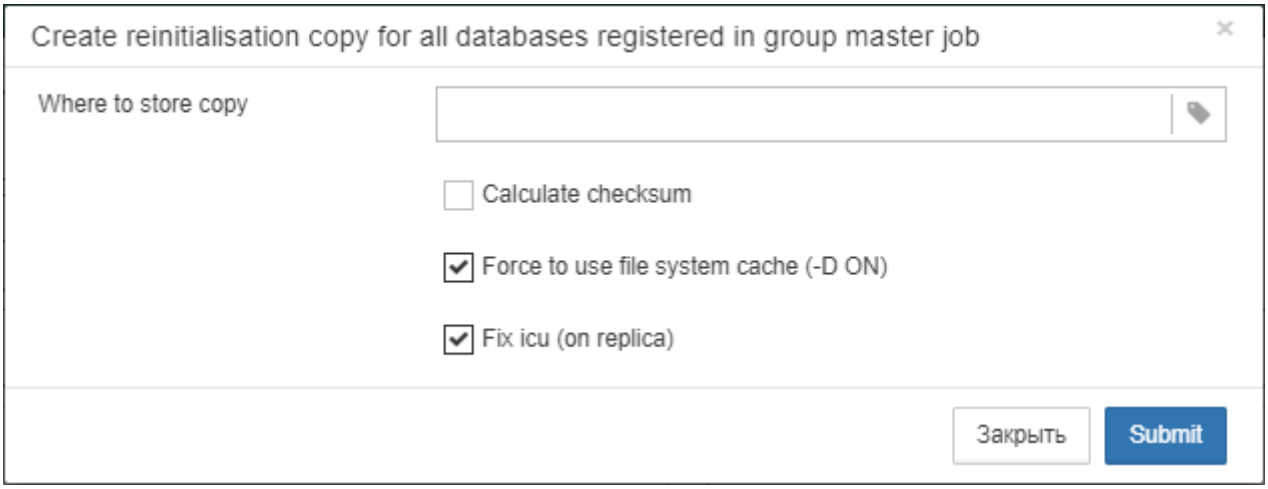
Для получения начального списка баз данных для групповой обработки необходимо нажать кнопку “Пересканировать каталоги и пересоздать списки и конфигурации репликации”.




DataGuard выполняет обход каталога и реконструирует список баз (в соответствии с настройками задачи — шаблоны, исключения, рекурсия). В список попадают только те базы, которые не зарегистрированы в DataGuard как “индивидуальные”. Если пользователь зарегистрировал в DataGuard базу данных, то её не нужно вписывать в шаблон исключения задачи, она будет исключена автоматически.




В этом же разделе присутствуют дополнительные кнопки для групповой реинициализации **всех** зарегистрированных баз и для установки флагов публикации для всех баз (актуально для Firebird версии 4.0 и выше).




# 3.4. Конфигурация базы данных в FBDataGuard

## 3.4.1. Регистрация базы данных Firebird

Список баз данных, отслеживаемых FBDataGuard, находится в разделе “Базы данных”.

Чтобы зарегистрировать базу данных в FBDataGuard, необходимо нажать на символ “Плюс” в правом углу “Базы данных” (появится подсказка “Добавить БД”) и заполнить следующую форму:

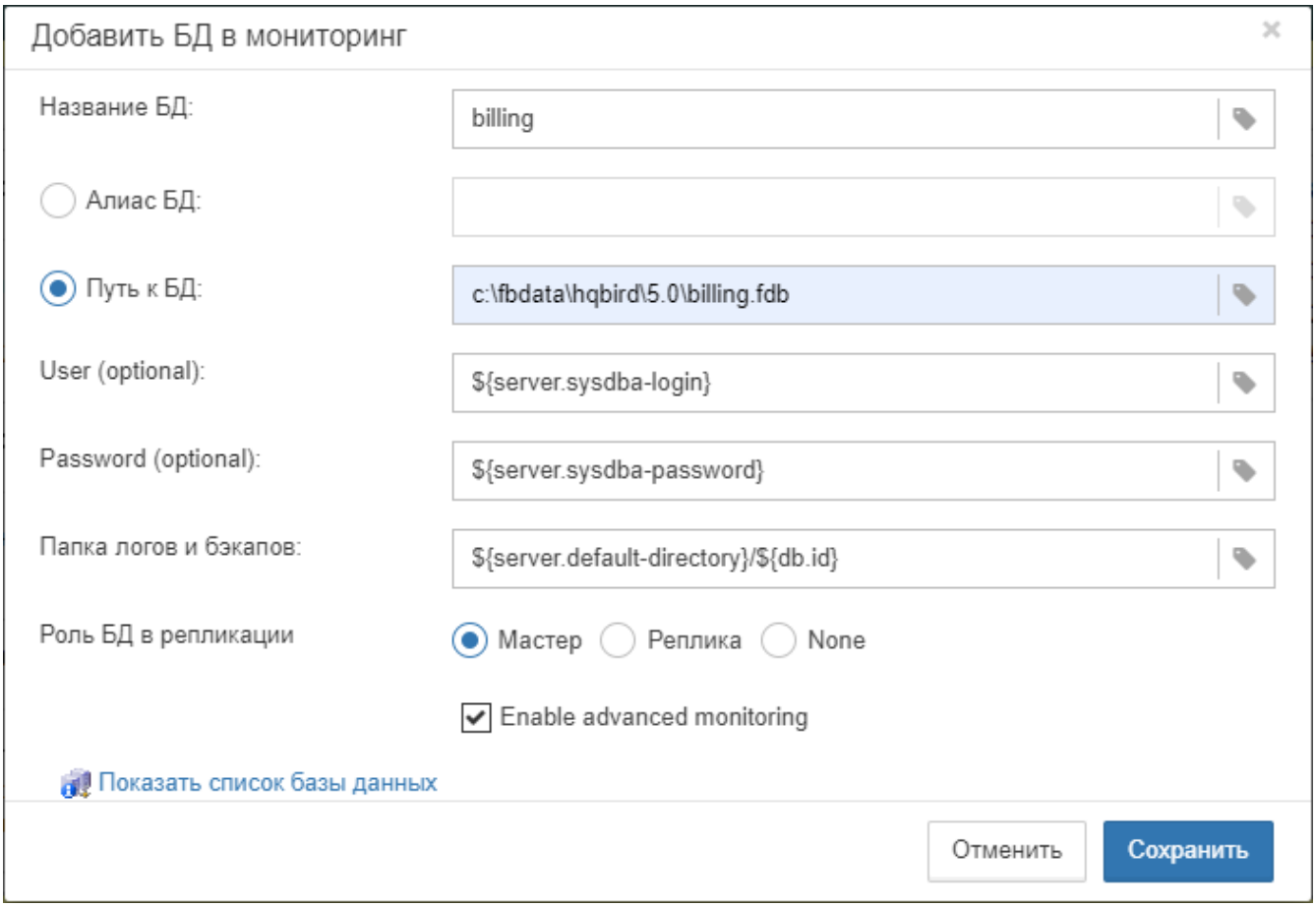



*Рисунок 31. Добавление базы данных в мониторинг*

- “**Название БД**” предназначен для вашего удобства и используется для ссылки на эту базу данных в оповещениях и сообщениях электронной почты.
- “**Алиас БД**” — это псевдоним базы данных из `aliases.conf` или `databases.conf`. Если вы укажете и “Алиас БД”, и “Путь к БД”, то будет использоваться “Алиас БД”.
- “**Путь к базе данных**” — это локальный путь к базе данных (помните, что FBDataGuard должен работать на одном компьютере с Firebird). Если вы помещаете базу данных на внешний диск, может возникнуть ошибка “File… has unknown partition”. Чтобы это исправить, вам нужно нажать “Настроить” в виджете “Сервер” и нажать “Сохранить”, чтобы FBDataGuard перечитал разделы.
- “**Папка логов и бэкапов**” — это папка, в которой FBDataGuard будет хранить резервные копии, журналы и статистику для этой базы данных. Если вы не выбрали папку HQbirdData во время установки и не указали выходную папку для сервера, рекомендуется указать “Папка логов и бэкапов” в каком-то явном месте, например `F:\mydatabasedata`.
- “**Enable advanced monitoring**” — см. Advanced Monitor Viewer

Вы можете указать точные абсолютные местоположения для резервных копий и статистики позже в соответствующих диалоговых окнах.

Список доступных для регистрации баз данных или их псевдонимов вы можете просмотреть, нажав на ссылку **Показать список базы данных**.

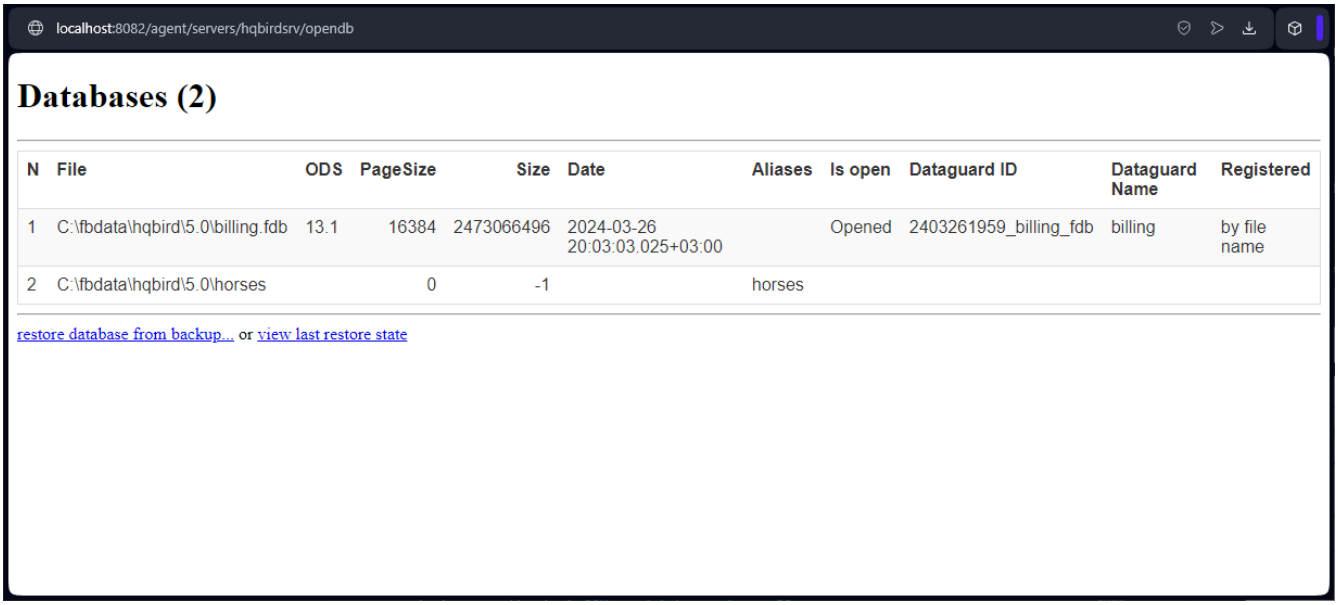



*Рисунок 32. Available database aliases.*

После регистрации FBDataGuard заполнит конфигурацию базы данных значениями по умолчанию, а затем отобразит веб-консоль с зарегистрированной базой данных:

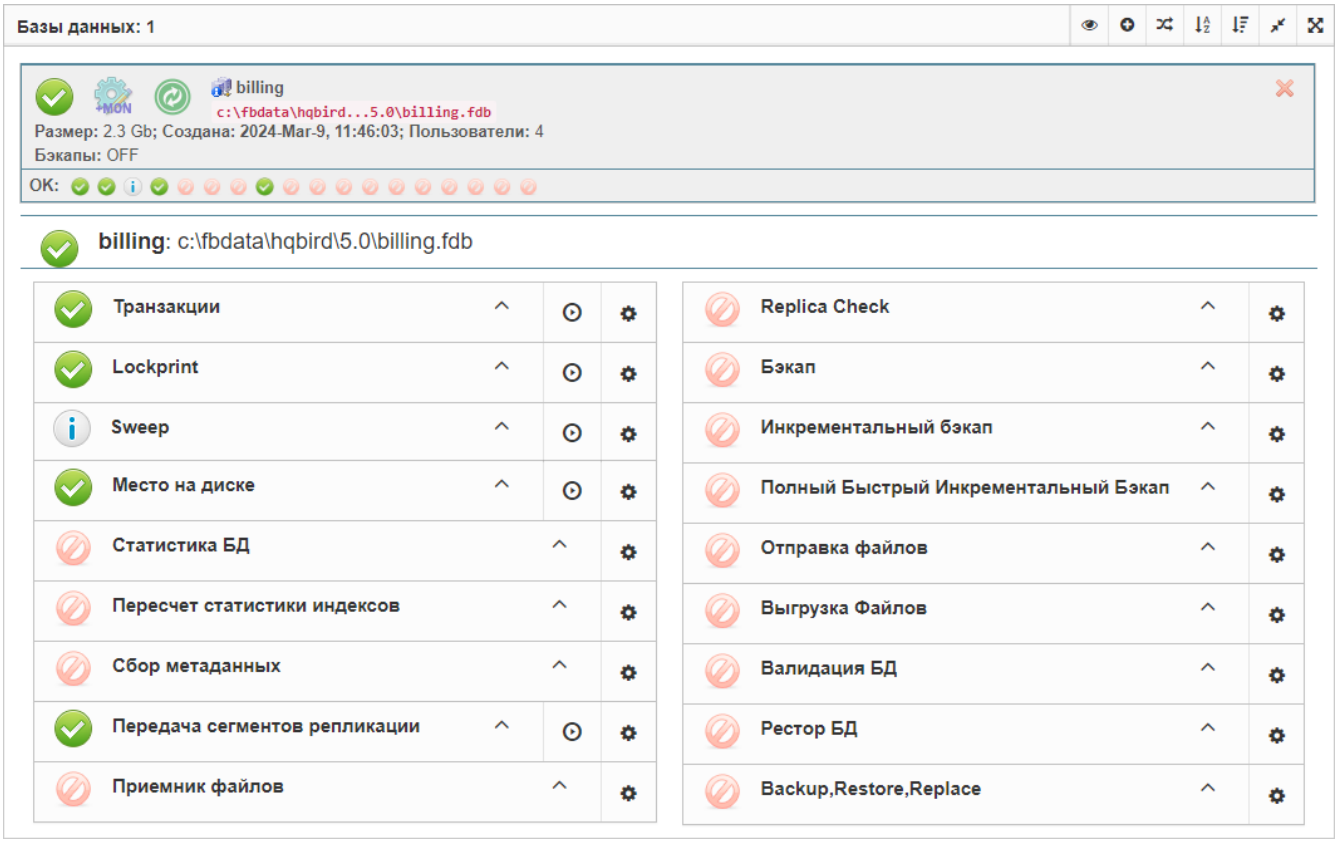
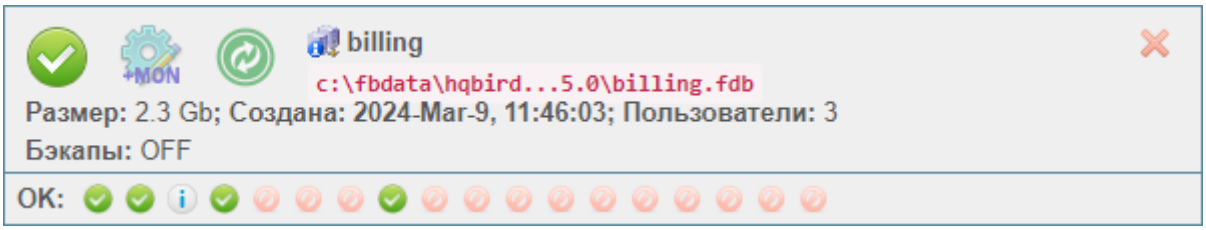



*Рисунок 33. HQbird FBDataGuard веб консоль после регистрации базы данных.*

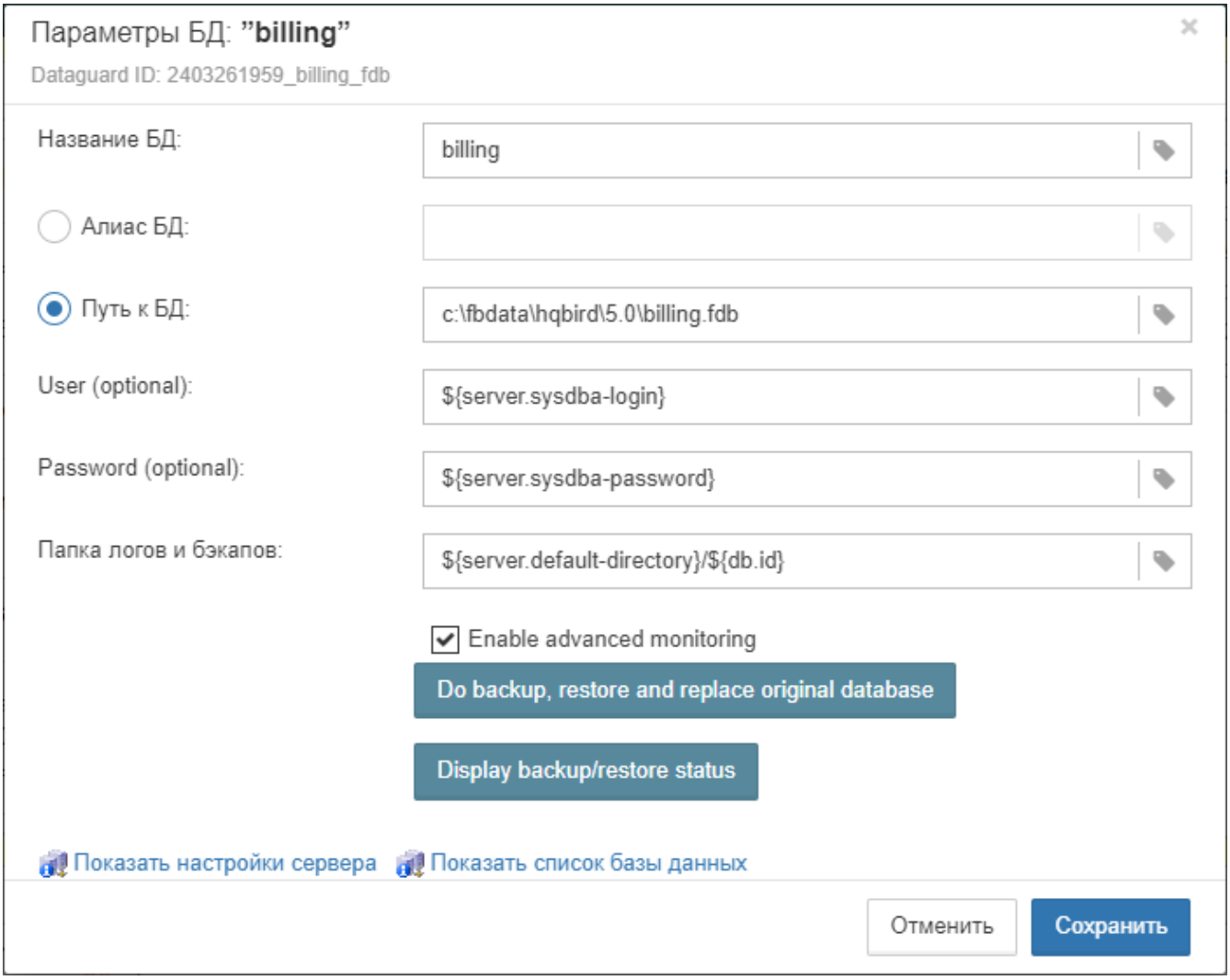
Вы можете изменить настройки базы данных позже; теперь приступим к настройке оповещений.

### 3.4.2. База данных: Общие настройки

FBDataGuard может контролировать несколько баз данных на одном сервере (до 80 баз данных). Для каждой базы данных создается отдельный виджет. Вверху виджета отображается состояние базы данных, никнейм базы данных (задается при добавлении базы данных и может быть изменен). Также виджет базы данных показывает полный путь к базе данных, ее размер, состояние резервных копий и количество подключенных в данный момент пользователей.




Используя диалог конфигурации, вы можете установить имя базы данных, путь к базе данных и папку вывода для базы данных (для хранения журналов и результатов заданий).




FBDataGuard проверяет корректность пути к базе данных и не позволяет указать неправильный путь.

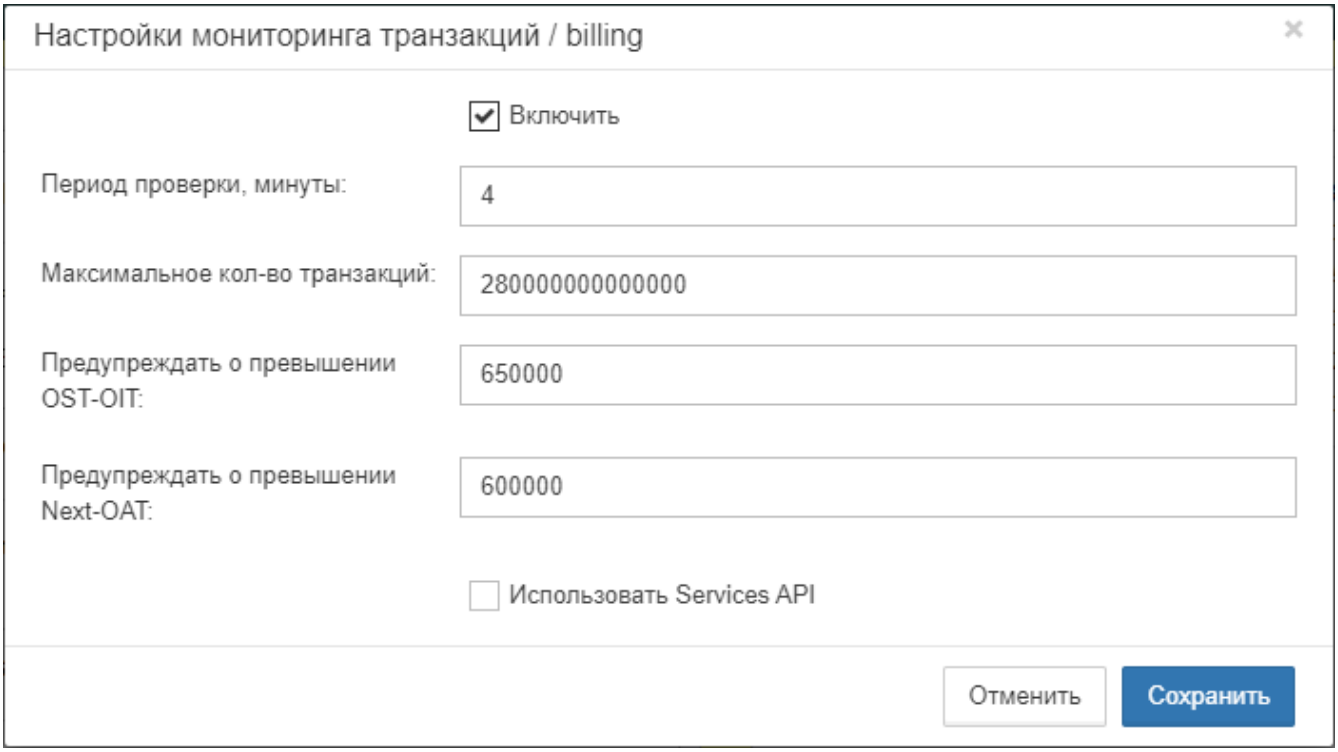
Также для HQbird в виджете базы данных можно увидеть статус репликации и настроить репликацию, нажав на иконку. Подробности читайте в разделе конфигурации репликации.

Виджет базы данных в HQbird также показывает статус шифрования базы данных.

### 3.4.3. База данных: Транзакции

Задание “База данных: Транзакции” предназначено для регистрации активности транзакций. Оно отслеживает два важных интервала: разницу между Oldest Active Transaction и Next transaction, а также разрыв между Oldest Snapshot и Oldest Interesting.

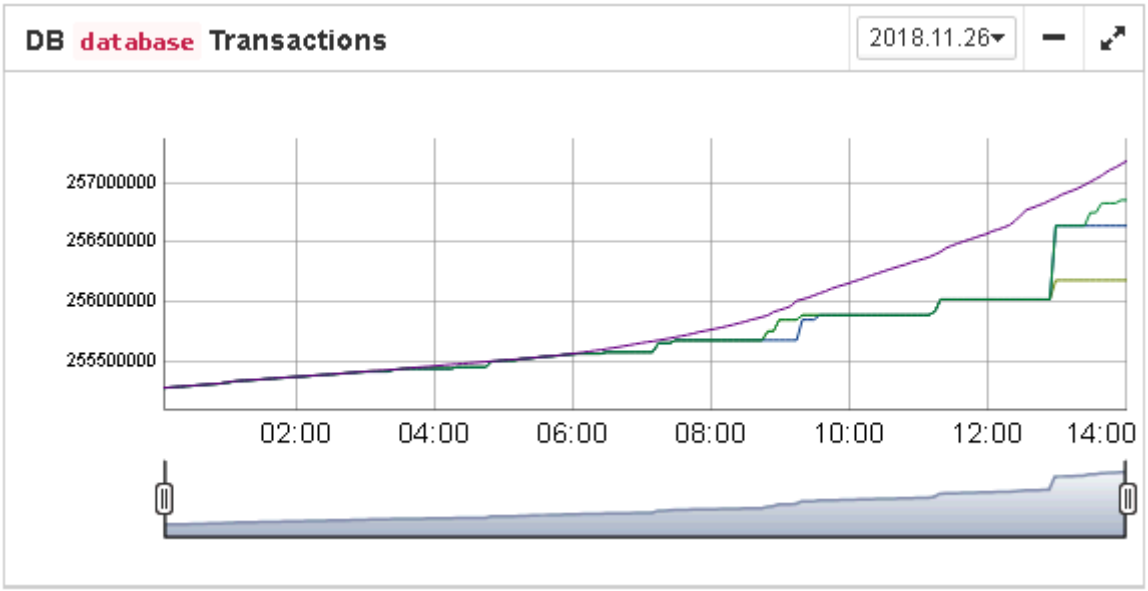
Если эти интервалы выходят за рамки указанного порога, то это означает проблемы с управлением транзакциями.




Эти журналы можно проанализировать, чтобы получить полезную информацию о производительности базы данных и качестве приложений (дополнительную информацию см. здесь http://ib-aid.com/en/articles/ibanalyst-what-you-can-see-at-summary-view/).

Это задание также отслеживает ограничение реализации в Firebird: максимальное количество транзакций в версиях Firebird до 3.0 должно быть меньше $2^{31}$-1. При приближении к этому значению необходимо выполнить резервное копирование и восстановление базы данных. Оно выдаст предупреждение, если номер транзакции будет близок к ограничениям.

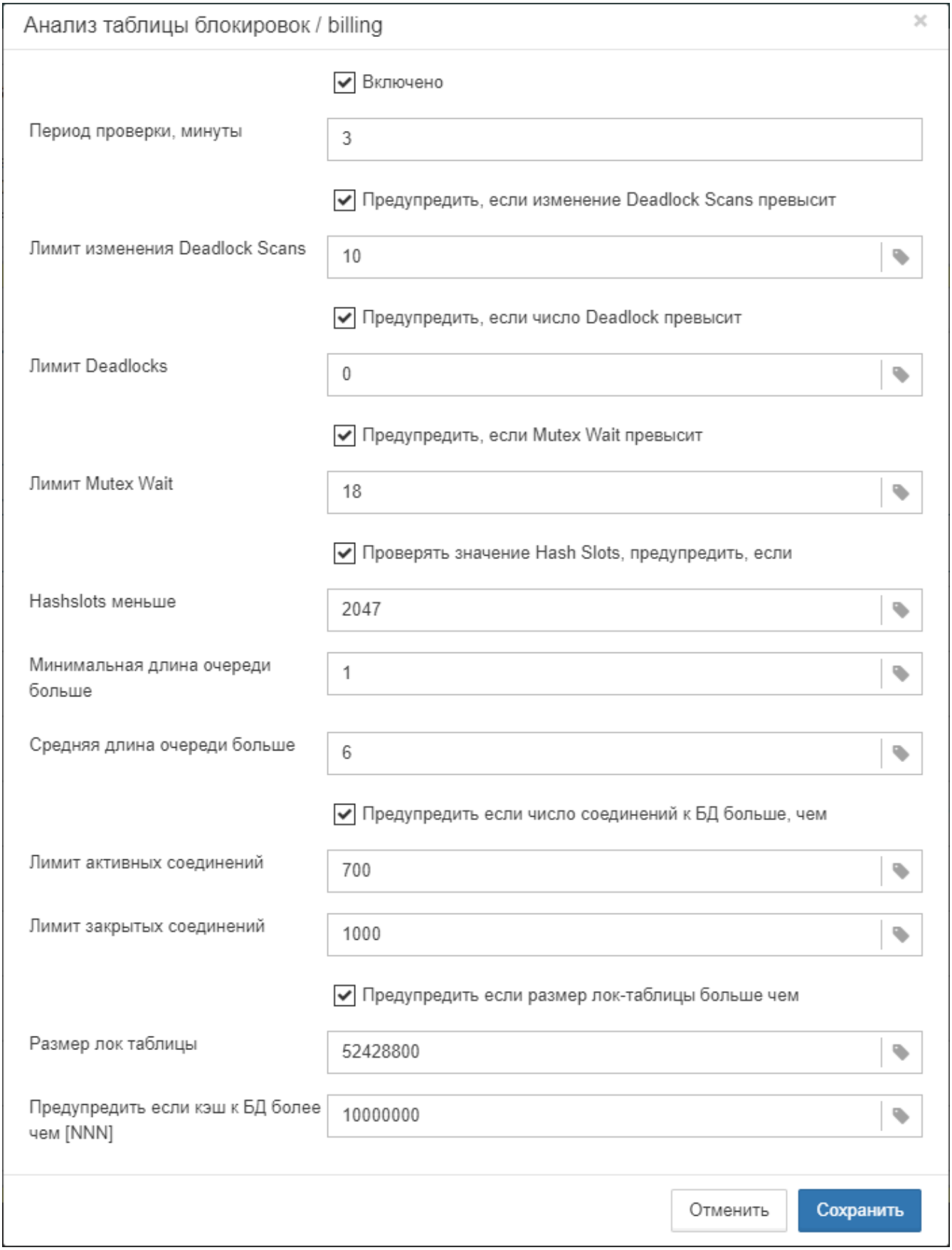
Также динамика транзакций отображается на вкладке “Graphs gallery”:




### 3.4.4. База данных: Lockprint

Задание “Lockprint” отслеживает информацию из таблицы блокировок Firebird. Это очень важно для архитектур Classic/SuperClassic и полезно для SuperServer.

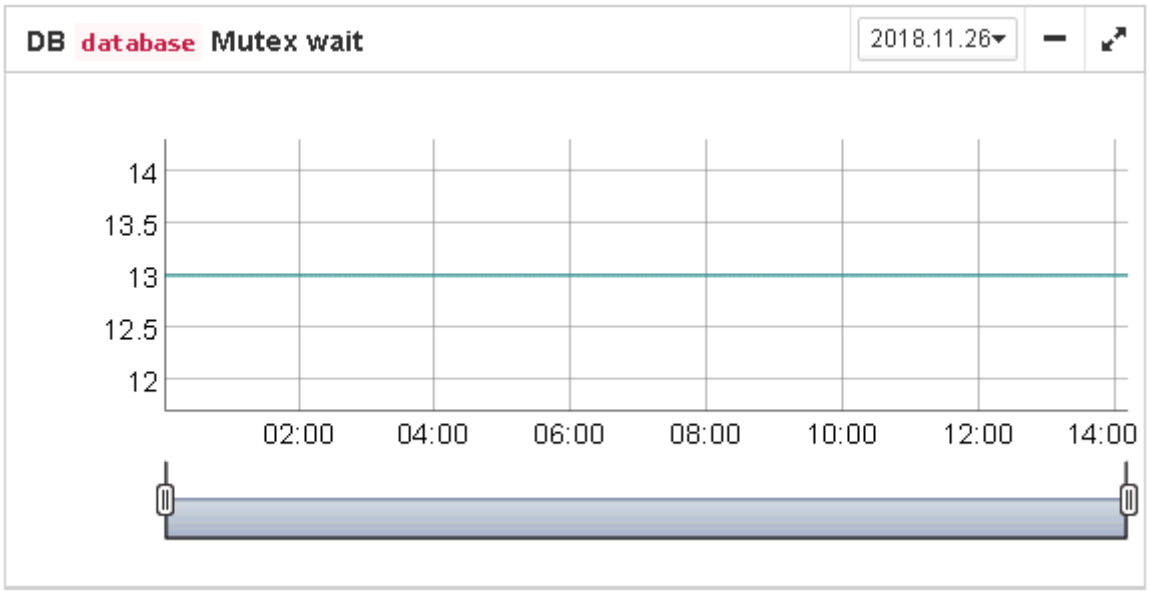
Таблица блокировок — это внутренний механизм Firebird для организации доступа к объектам внутри движка Firebird. HQbird отслеживает важные параметры таблицы блокировок:



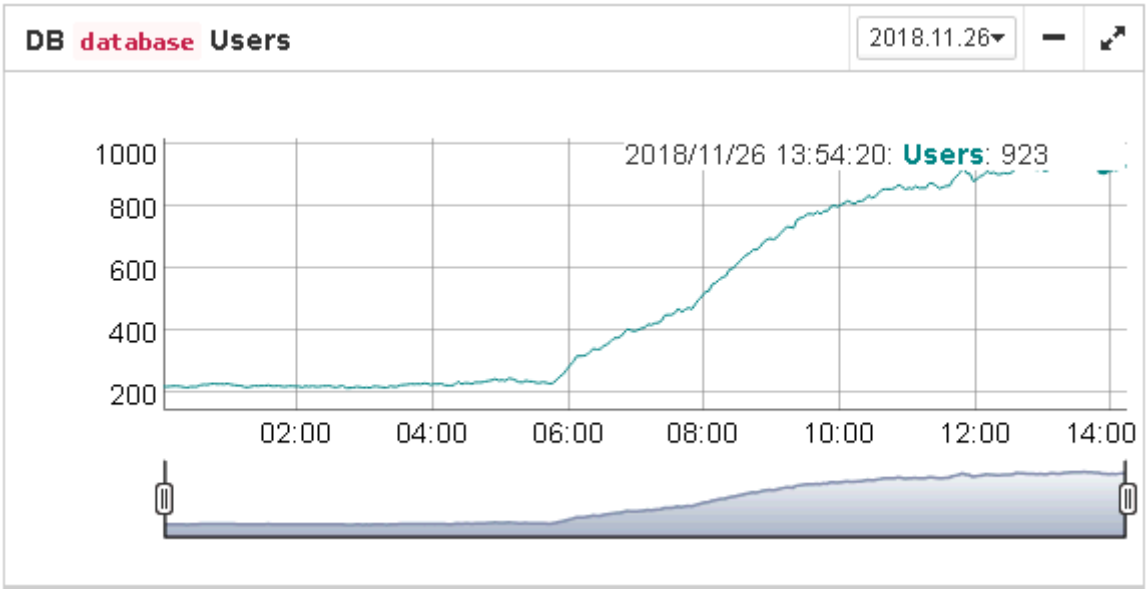

- **Период проверки, минуты** — как часто HQbird анализирует таблицу блокировок. 3 минуты — оптимальный интервал.
- **Лимит изменения Deadlock Scans** — сканирование взаимоблокировок — это процесс, запускаемый движком Firebird в случае длительной задержки ответа от одного из потоков. Если количество взаимоблокировок велико, это означает, что Firebird сильно

загружен. Значение накапливается с момента запуска движка Firebird. Значение по умолчанию довольно велико — 12345, поэтому его превышение означает низкую производительность базы данных.

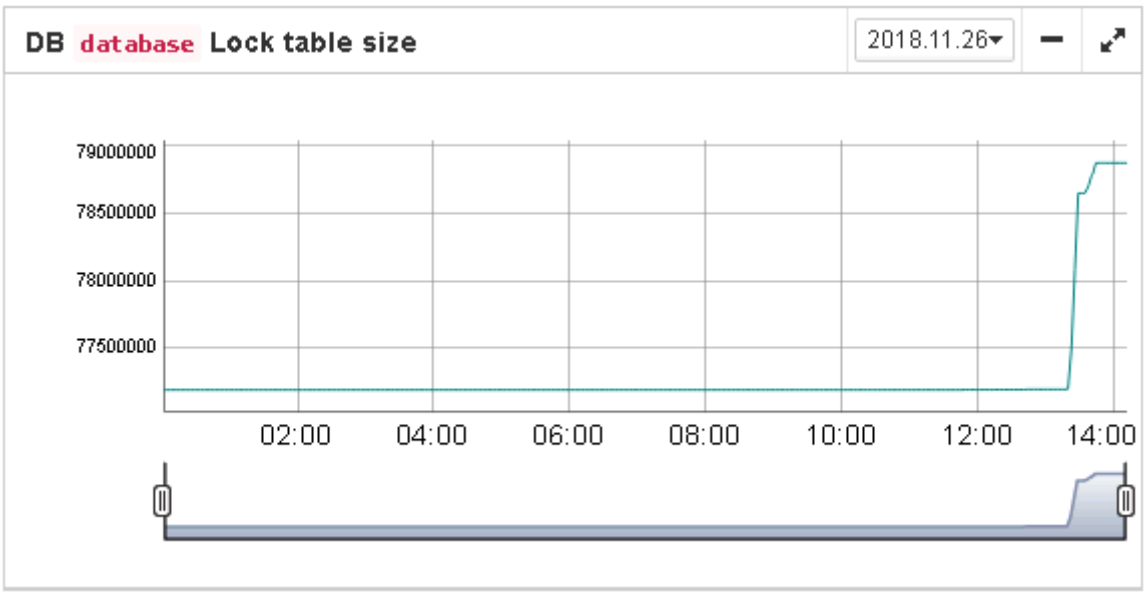
- **Лимит Deadlocks** — если движок Firebird обнаруживает истинную взаимоблокировку во время сканирования взаимоблокировок, он увеличивает это значение. Обратите внимание: настоящие взаимоблокировки случаются очень редко. Не путайте их с конфликтами транзакций (“deadlock. Lock conflict on nowait transaction” и другое).
- **Лимит Mutex Wait**. Mutex Wait — это параметр таблицы блокировок, который неявно указывает на конфликты ресурсов. Чем выше время ожидания мьютекса, тем выше конкуренция внутри движка за ресурсы. По умолчанию порог ожидания мьютекса установлен на 18%, но это значение не является универсальным для всех баз данных. Хороший подход — следить за значениями мьютексов в течение 1–2 недель, а затем устанавливать самое высокое значение, наблюдаемое за этот период. График ожидания мьютекса доступен в галерее Mutex Wait.



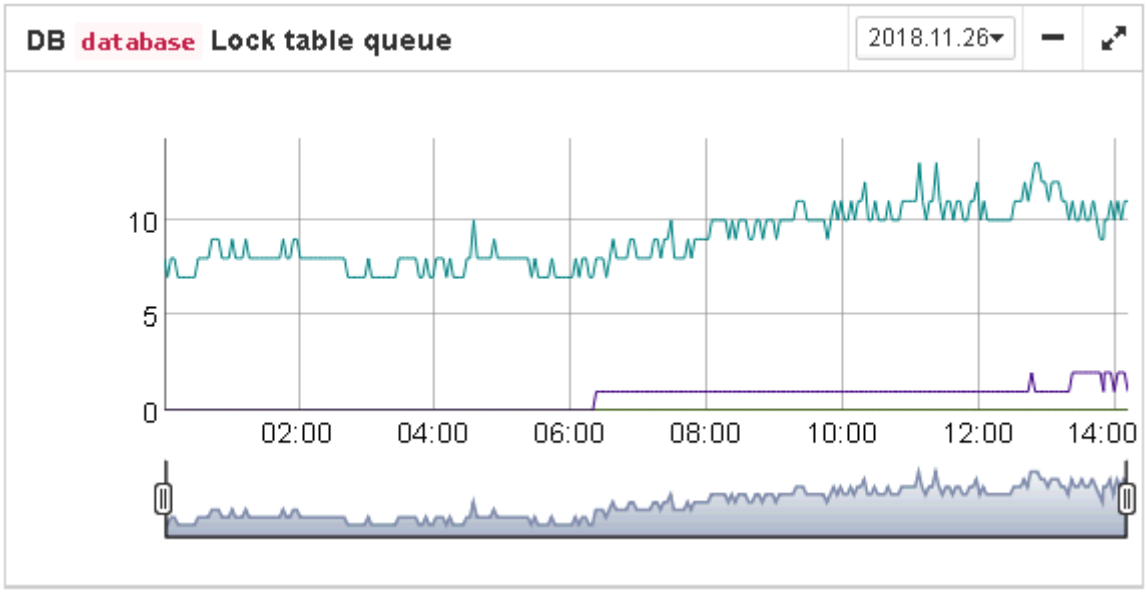

- **Проверять значение Hash Slots**. В заголовке таблицы блокировок есть параметр “Hash lengths (min/avg/max): 0/0/4”, он отображает длины цепочек коллизий в хеш-таблице блокировок. Важно сохранять эти значения как можно меньшими, поэтому HQbird отслеживает их и подсказывает, как улучшить ситуацию, если длина цепочки коллизий в хеш таблице больше, чем указано в этом задании.
- **Лимит активных соединений** “Owners” — количество подключений, установленных к указанной базе данных. Фактически, это самый быстрый способ получить фактическое количество подключений к базе данных с минимальной нагрузкой на базу данных — другие способы, такие как запрос к `MON$ATTACHMENTS` или `isc_tpb_database`, имеют различные недостатки. Ограничение здесь должно быть установлено в соответствии с фактическим пиковым количеством подключений. Например, если вы уверены, что пиковое количество подключений к вашей базе данных составляет 500, установите в качестве лимита 550, и если в какой-то момент нагрузка увеличится, вы не пропустите этот момент.




- **Лимит закрытых соединений**. “Free owners” — это соотношение между пиковым количеством соединений и текущим количеством соединений. Если вы видите `Free owners = 0`, то это означает, что количество подключений постоянно растет с момента запуска Firebird. Если вы видите большое количество Free owners, то это может означать, что многие соединения недавно были отключены.
- **Размер таблицы блокировок**. Размер таблицы блокировок является неявным индикатором нагрузки на систему. Обычно размер таблицы блокировок должен быть стабильным. Кроме того, рекомендуется установить первоначальный размер таблицы блокировок равным значению, которое она имеет после некоторого периода активной работы — хотя таблица блокировок увеличивается по требованию, процесс перераспределения является тяжелой операцией и может привести к микрозависаниям в ответах базы данных. График таблицы блокировки полезен для определения правильного начального значения.




- **Очередь к таблице блокировок**. Очередь таблиц блокировки не имеет явного порога в задании Lockprint, но ее значения собираются и отображаются в “Graphs gallery”. Очередь таблицы блокировок является индикатором общей загрузки.




### 3.4.5. База данных: Пересчет статистики индексов

“База данных: Пересчет статистики индексов” — важная задача, которая помогает поддерживать производительность индексов на оптимальном уровне, а также выполняет дополнительную проверку работоспособности базы данных.

“Пересчет статистики индексов” позволяет запустить пересчет значений селективности индексов. Во время этой процедуры Firebird быстро просматривает листовые страницы индексов и обновляет статистику избирательности. Посещая эти страницы, Firebird также проверяет их целостность, и если индекс поврежден, то будет выдано предупреждение.

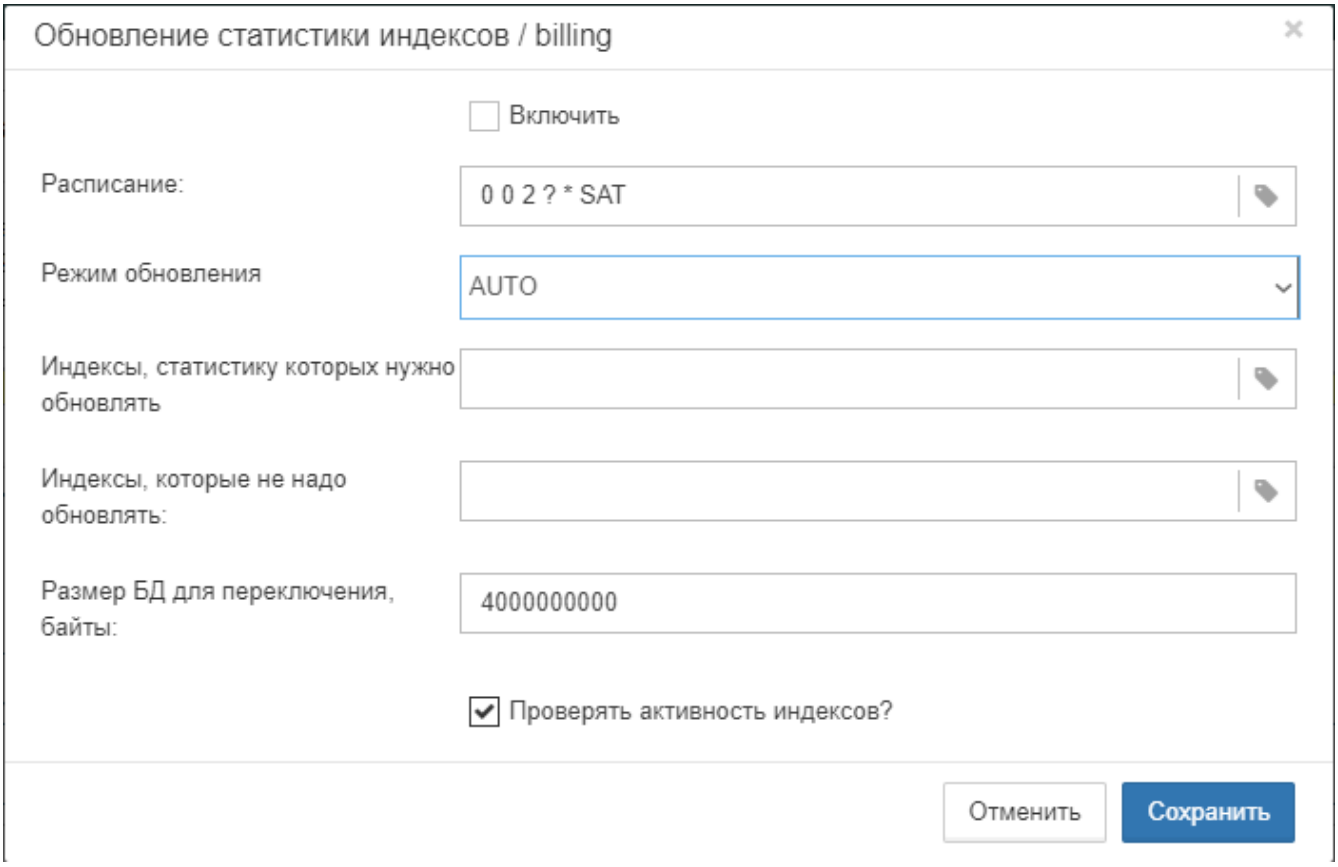
Кроме того, это задание проверяет, активны ли все индексы в базе данных. Неактивные или неактивированные индексы обычно указывают на повреждения и приводят к снижению производительности.

По умолчанию это задание отключено, но мы рекомендуем включить его после тщательного выбора показателей для пересчета.

В этом задании есть три режима: AUTO, ALL, SELECTED.

ALL — режим, в котором будут проверены все индексы.

AUTO — режим по умолчанию. Он очень похож на ALL, но также проверяет размер базы данных и не трогает индексы, если база данных больше 3,6 ГБ.




SELECTED — рекомендуемый режим. Это позволяет выбрать индексы, которые следует пересчитывать, или те, которых следует избегать.

Для включения индексов в список пересчитываемых необходимо указать названия индексов (через запятую), а для исключения – выполнить то же самое в соответствующем поле.

Как вы можете видеть на снимке экрана диалогового окна конфигурации, здесь есть поля для включения/отключения задания, установки режима обновления, а также включения или исключения индексов. “Размер БД для переключения, байты” — устанавливает предел, при котором работает режим AUTO. Переключатель “Проверять активность индексов” должен быть включен всегда, пока вы не проведете специальные манипуляции с неактивными индексами.

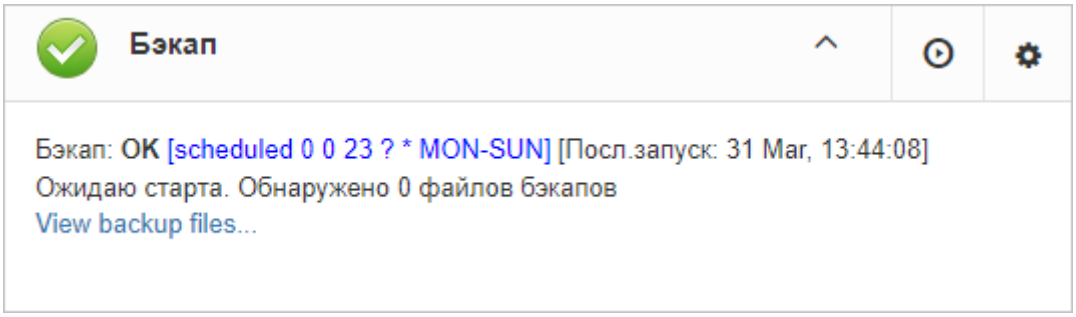
### 3.4.6. База данных: Бэкап

“База данных: Бэкап” — одно из ключевых заданий, гарантирующих сохранность данных, хранящихся в защищенной базе данных. При разработке HQbird мы учитывали определенный сценарий восстановления, и этот сценарий подразумевает, что ключевой целью защиты базы данных является минимизация потенциальных потерь данных. Если у нас есть работоспособная резервная копия, восстановление может быть сконцентрировано на сохранении самых последних данных (только что введенных в базу данных), и это значительно сокращает время общего простоя.

Как вы увидите ниже, “База данных: Бэкап” — это не просто оболочка для стандартных функций `gbak` и планировщика, это умное задание, в котором есть множество встроенных правил для предотвращения проблем с резервным копированием и предоставления

подходящего интерфейса для управление резервными копиями.

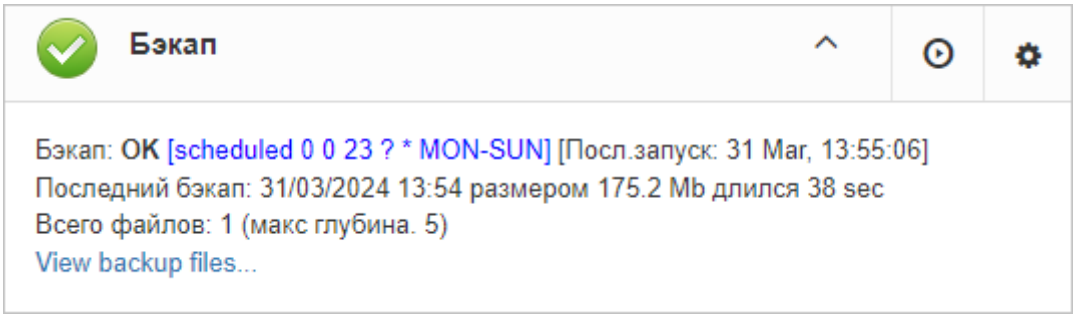
Задание “База данных: Бэкап” отключено **по умолчанию**, но мы настоятельно рекомендуем изменить его настройки сразу после установки HQbird.




Первоначально задание “База данных: Бэкап” отображается как OK, хотя попытка резервного копирования не выполнялась. В этом случае OK означает, что резервное копирование как минимум запланировано.

Также это задание распознает файлы по шаблону имен (см. ниже информацию о конфигурации) и показывает общее количество резервных копий.

После завершения резервного копирования информация в виджете изменится: будет показано время создания последней успешной резервной копии, а также время, затраченное на фактическое выполнение резервного копирования (всего 1 минута 12 секунд на скриншоте с примером).




Кроме того, подробное оповещение будет отправлено на вашу электронную почту и/или в HQbird Control Center:

| Name | Desc |
|---|---|
| Regular backup was done successfully. | Backup 'C:\HQBirdData\output\output\hqbirdsrv\2403261959_billing_fdb\backup\backup_20240331_13-53.zbk' 1.2 GiB was created at 2024-03-31 13:53:56.286+03:00 took total 01m:09s.874 to complete. Test restore is omitted.<br><br>Backup:<br>[OK] Backup 'C:\HQBirdData\output\output\hqbirdsrv\2403261959_billing_fdb\backup\backup_20240331_13-53.fbk' 1.2 GiB done successfully at 2024-03-31 13:54:35.305+03:00, taking 00m:38s.964.<br><br>Other:<br>[OK] Packing 'C:\HQBirdData\output\output\hqbirdsrv\2403261959_billing_fdb\backup\backup_20240331_13-53.zbk' 175.2 MiB done successfully at 2024-03-31 13:55:06.160+03:00, taking 00m:30s.784. |

“База данных: Бэкап” проверяет свободное место на диске с местом назначения резервной

копии, и если обнаруживает, что на диске недостаточно свободного места, будет отправлено CRITICAL предупреждение, а текущая резервная копия будет отменена (при необходимости).

Будьте осторожны: по умолчанию время резервного копирования установлено ***23-00 Понедельник-Воскресенье**.

По умолчанию резервные копии базы данных будут храниться в выходной папке, указанной вами на этапе установки! По умолчанию это `C:\HQbirdData\output...`

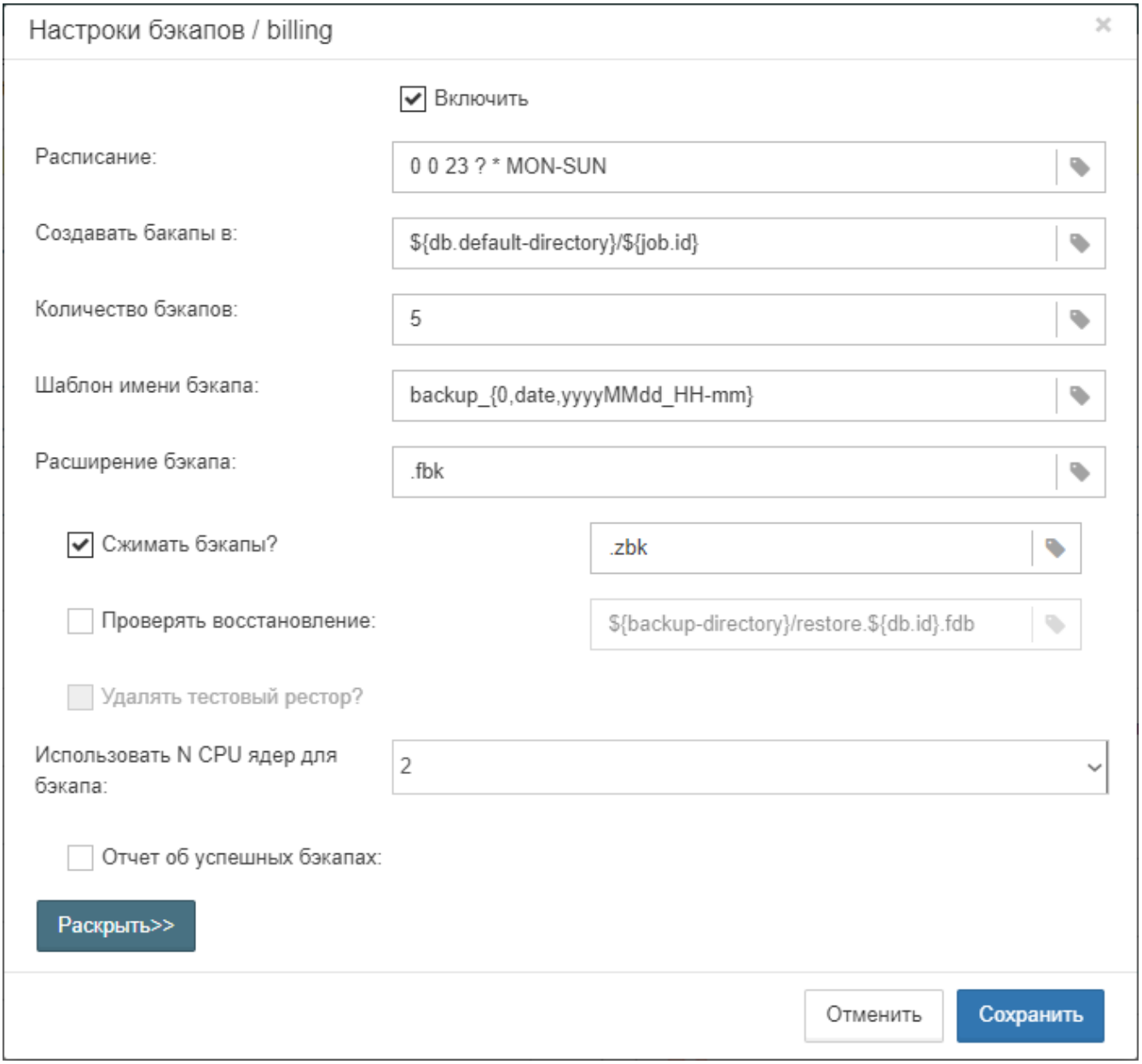
Очень важно внимательно просмотреть настройки резервного копирования базы данных и настроить их в соответствии с локальной конфигурацией!

Рассмотрим диалог настройки резервного копирования подробнее:

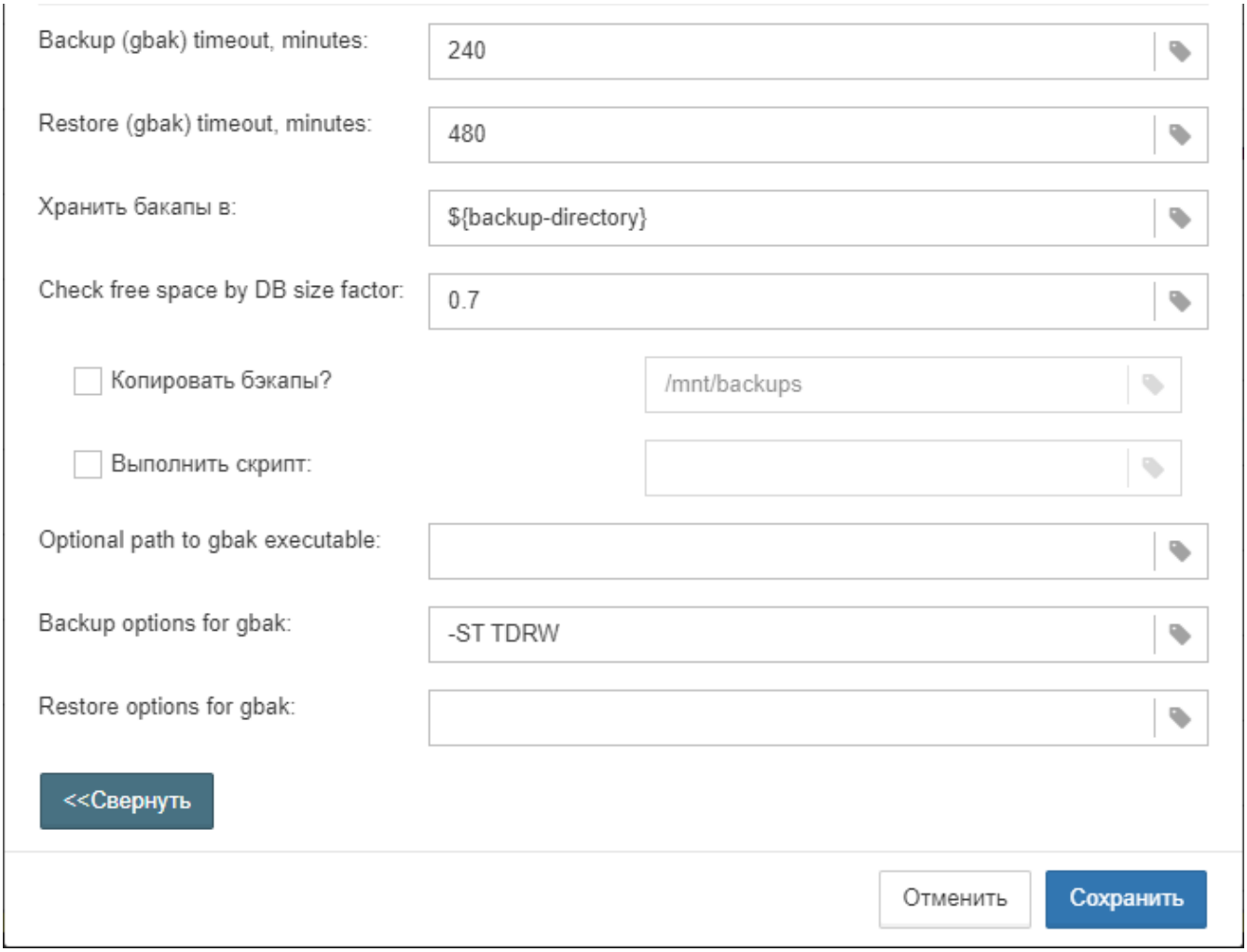
- **“Включить”** — включает или отключает задание резервного копирования.
- В поле **“Расписание”** вы можете установить время, когда должно запускаться резервное копирование. Планировщик использует выражение CRON (см. Выражения CRON).
- **“Создавать бакапы в”** указывает папку для хранения резервных копий. Эта папка должна находиться на том же компьютере, где находится база данных. По умолчанию она расположен в каталоге базы данных по умолчанию. Обычно рекомендуется указать явный путь к папкам с резервными копиями.
- **“Количество бэкапов”** указывает, сколько предыдущих резервных копий должно храниться. FBDataGuard хранит резервные копии в револьверном порядке: когда будет достигнуто максимальное количество (т. е. будет создано 5 резервных копий), FBDataGuard удалит самую старую резервную копию и создаст новую резервную копию. В сочетании с выражениями CRON это дает мощную возможность создавать необходимую историю резервных копий.
- **“Шаблон имени бэкапа”** определяет, как будут именоваться файлы резервных копий. Кроме того этот шаблон имени позволяет FBDataGuard распознавать старые резервные копии с тем же шаблоном имени.
- **“Расширение бэкапа”** — по умолчанию `.fbk`.
- **“Сжимать бэкапы”** указывает, должен ли FBDataGuard архивировать резервные копии после обычного резервного копирования Firebird. По умолчанию эта опция включена, но вы должны знать, что FBDataGuard будет архивировать файлы резервных копий размером менее **100 ГБ**. После достижения этого размера сжатие резервной копии будет автоматически отключено. Мы рекомендуем включать эту функцию только для небольших баз данных.
- **“Проверить восстановление”** — важная опция. Если она включена, то FBDataGuard выполнит тестовое восстановление новой резервной копии, чтобы проверить её. Это гарантирует качество созданной резервной копии и уведомляет администратора в случае возникновения проблем с тестовым восстановлением.
- **“Удалять тестовый рестор”** указывает, должен ли FBDataGuard удалить восстановленную базу данных. По умолчанию она ВЫКЛЮЧЕНА, поэтому вы можете ВКЛЮЧИТЬ её, но вам нужно внимательно подумать: действительно ли вам нужно

сохранять копию тестовой восстановленной базы данных. При каждом тестовом восстановлении эта копия будет перезаписана.

- **“Использовать N CPU ядер для бэкапа”** — эта функция позволяет выполнять резервное копирование базы данных и восстановление тестовой базы данных с использованием нескольких ядер ЦП, поэтому резервное копирование может выполняться в 3-5 раз быстрее. Мы рекомендуем выделить половину ядер вашего процессора.
- **“Отчет об успешных бэкапах”** — по умолчанию отключено, но настоятельно рекомендуется включить её и начать получать уведомления о успешном резервном копировании. Эта функция будет использовать настройки электронной почты из системы оповещений.




Если вы нажмете кнопку **[ Раскрыть>> ]**, то появятся расширенные параметры резервного копирования:




- **“Backup (gbak) timeout, minutes”** — максимальное время для выполнения только операции резервного копирования (`gbak -b`), в противном случае будет сгенерировано предупреждение.
- **“Restore (gbak) timeout, minutes”** — максимальное время для завершения тестовой операции восстановления.
- **“Хранить бэкапы в”** — если вам нужно сделать резервные копии в одну папку, а затем переместить созданную резервную копию в другую папку (например, для долговременного хранения), вы можете изменить значение этого параметра с `${backup-directory}` на папку где вы их будете хранить. Файлы резервных копий в обоих местах отслеживаются HQbird FBDataGuard и включаются в счетчик резервных копий, отображаемый в виджете.
- Переключатель **“Копировать бэкапы”** и путь. . Если у вас есть сетевое расположение или подключенный USB-накопитель для хранения базы данных, на котором вы хотите хранить копию резервной копии (помимо обычных резервных копий), FBDataGuard может скопировать туда последнюю резервную копию: просто включите переключатель: просто включите переключатель “Копировать бэкапы” и задайте путь. Скопированные файлы не отслеживаются и не включаются в число резервных файлов, отображаемых в виджете.
- Переключатель **“Выполнить скрипт”** и путь к скрипту. После завершения общей процедуры резервного копирования можно указать собственный сценарий или исполняемый файл. Скрипт получает в качестве параметра путь к новой резервной

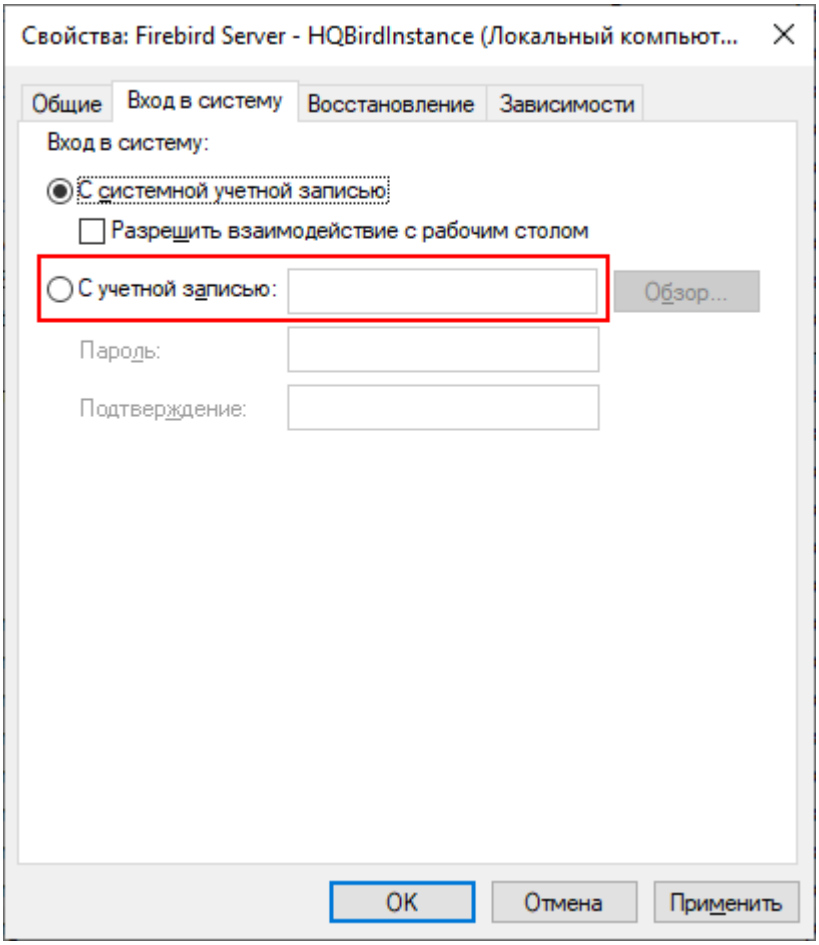
копии базы данных.

- **“Optional path to gbak executable”** — можно указать другой не стандартный `gbak`.
- **“Backups option for gbak”** — если вам нужно добавить какие-то конкретные опции резервного копирования, добавьте их сюда.
- **“Restore options for gbak”** — если вам нужно добавить какие-то конкретные опции тестового восстановления, добавьте их сюда.

Если вы отслеживаете более одной базы данных, настоятельно рекомендуется разделить время выполнения восстановления.

### Важное замечание: резервное копирование на сетевое хранилище

Обратите внимание, что для создания и копирования резервной копии в сетевые хранилища службы Firebird и FBDataGuard должны быть запущены под учетной записью с достаточными правами. По умолчанию Firebird и FBDataGuard запускаются под учетной записью LocalSystem, у которой нет прав доступа к сетевому расположению.




Итак, чтобы хранить резервные копии Firebird в сетевом расположении в Windows, запустите апплет “Службы” (`services.msc`) и на вкладке “Вход в систему” измените “С учётной записью” на соответствующую учетную запись (подойдет Администратор домена).

В Linux — добавьте необходимые права для пользователя `firebird`.

### 3.4.7. База данных: Инкрементальный бэкап

Инкрементальный бэкап — это задание по планированию и управлению инкрементным резервным копированием в Firebird.

Обратите внимание, что мы рекомендуем использовать инкрементное резервное копирование только в сочетании с проверяемым резервным копированием, поскольку при инкрементальном резервном копировании происходит копирование страниц базы данных, измененных с момента последнего резервного копирования (в случае многоуровневого инкрементного резервного копирования).

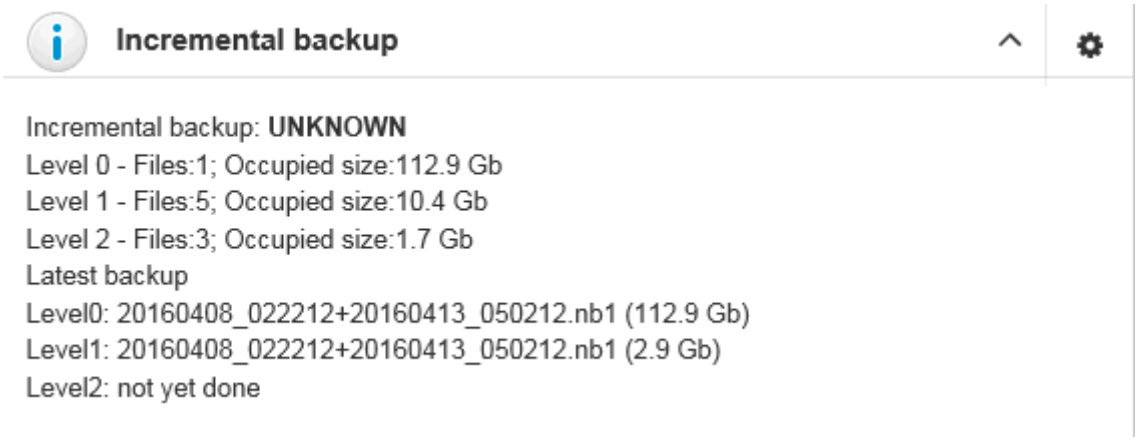
HQbird FBDataGuard реализует 2 типа многоуровневого инкрементного резервного копирования: Простое и Расширенное инкрементное копирование, а также полное инкрементное копирование (см. База данных: Полный Инкрементальный Бэкап).

Многоуровневое резервное копирование в Firebird должно выполнять следующие шаги:

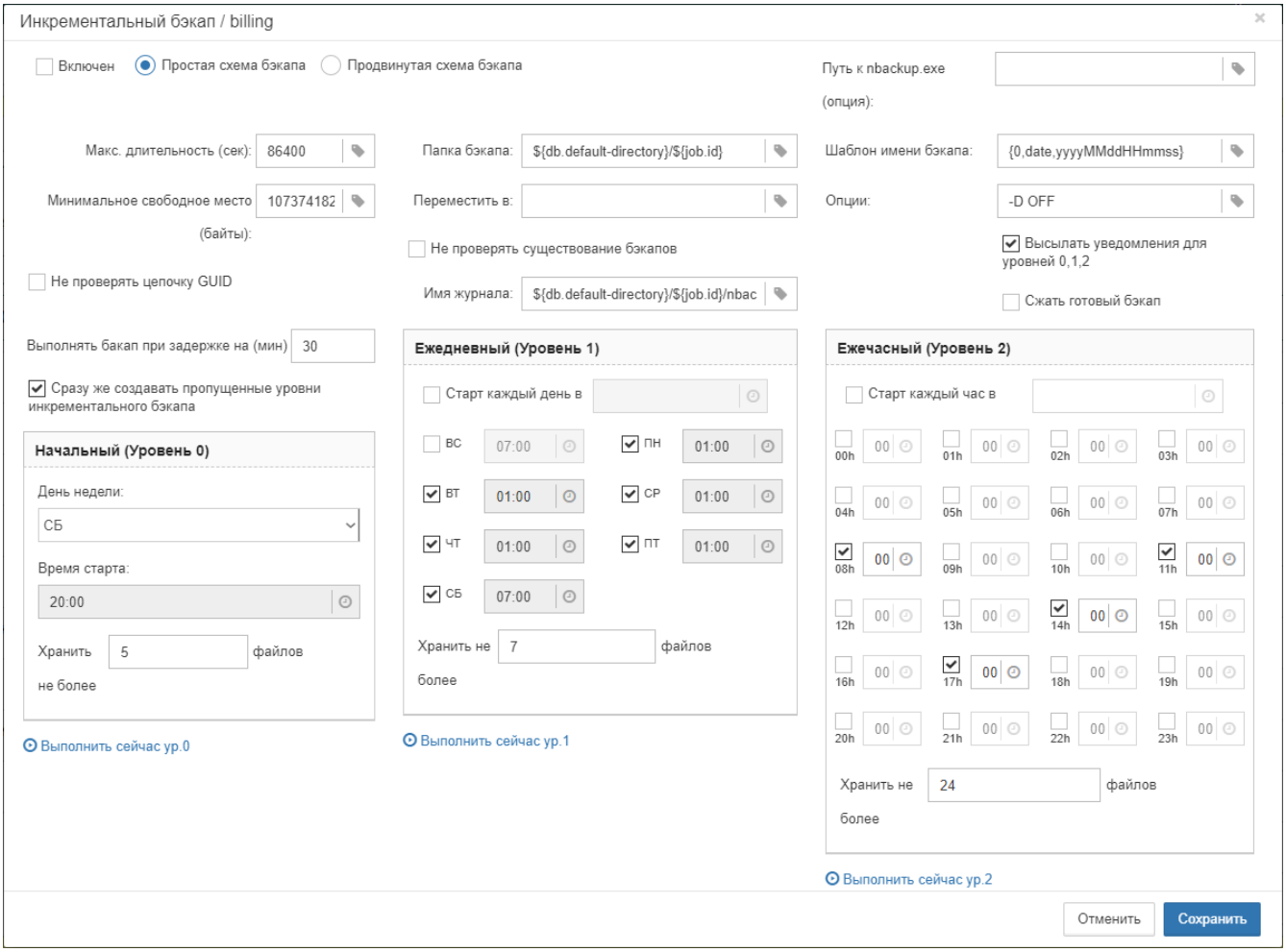
1. Создаётся начальная резервная копия (уровень 0), которая по сути является копией базы данных на момент запуска резервного копирования, и помечает её с помощью GUID резервной копии.
2. Поскольку Firebird при каждом изменении помечает каждую страницу данных определенным идентификатором, то можно найти страницы данных, изменившиеся с момента предыдущего резервного копирования, и скопировать только их, чтобы сформировать резервную копию уровня 1.
3. Возможно создание нескольких уровней резервных копий — например, начальная резервная копия (полная копия, уровень 0) создается каждую неделю, каждый день мы создаем копию уровня 1 (отличия от уровня 0), и каждый час мы создаем резервные копии уровня 2 (отличия от ежедневного уровня 1).

Инкрементальное резервное копирование с простым расписанием позволяет планировать 3 уровня резервного копирования: еженедельно, ежедневно и ежечасно.

Вы можете увидеть сводную информацию для такой конфигурации инкрементного резервного копирования на следующем снимке экрана ее виджета:

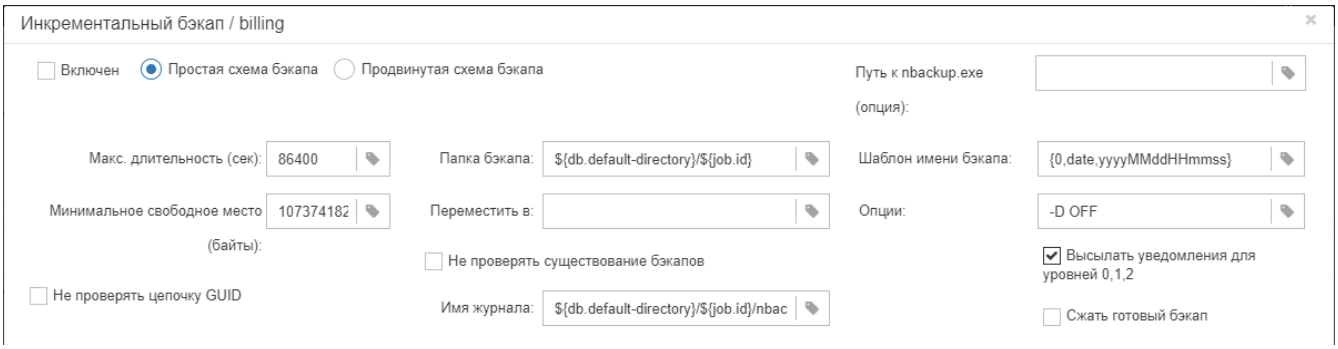



Чтобы настроить простое инкрементное резервное копирование, нажмите “шестеренку” настроек виджета и выберите “Простая схема бэкапа” (выбрано по умолчанию). Появится следующий диалог:




В этом диалоговом окне есть 4 основные области, давайте рассмотрим их одну за другой.

Верхняя область отведена под общие настройки инкрементального резервного копирования – они одинаковы для простого и продвинутой схемой бэкапа:




**Макс. длительность (сек)** — ограничение максимальной продолжительности процесса резервного копирования, по умолчанию составляет 1 день (86400 секунд).

**Минимальное свободное место (байты)** — минимальный размер свободного места на диске для предотвращения запуска резервного копирования, по умолчанию ~9Мб

**Папка бэкапа** — где будет храниться инкрементальная резервная копия выбранной базы данных. Необходимо хранить инкрементные резервные копии каждой базы данных отдельно от резервных копий других баз данных: т. е. в отдельной папке для каждой базы данных.

Необходимо указать папку резервных копий с достаточным количеством свободного места на диске для хранения резервных копий всех уровней!

**Имя журнала** — имя файла, который содержит информацию о файлах инкрементных резервных копий только для внутреннего использования.

**Путь к nbackup.exe** — можно указать другой инструмент nbackup, кроме стандартного `nbackup` (не рекомендуется).

**Шаблон имени бэкапа** — шаблон для файлов инкрементного резервного копирования (менять его не нужно).

**Опции** — дополнительные параметры для инструмента командной строки nbackup (менять их не нужно).

**Не проверять существование бэкапов** — этот параметр следует выбрать, если вы планируете удалить или создать дополнительные инкрементальные резервные копии в другом месте.

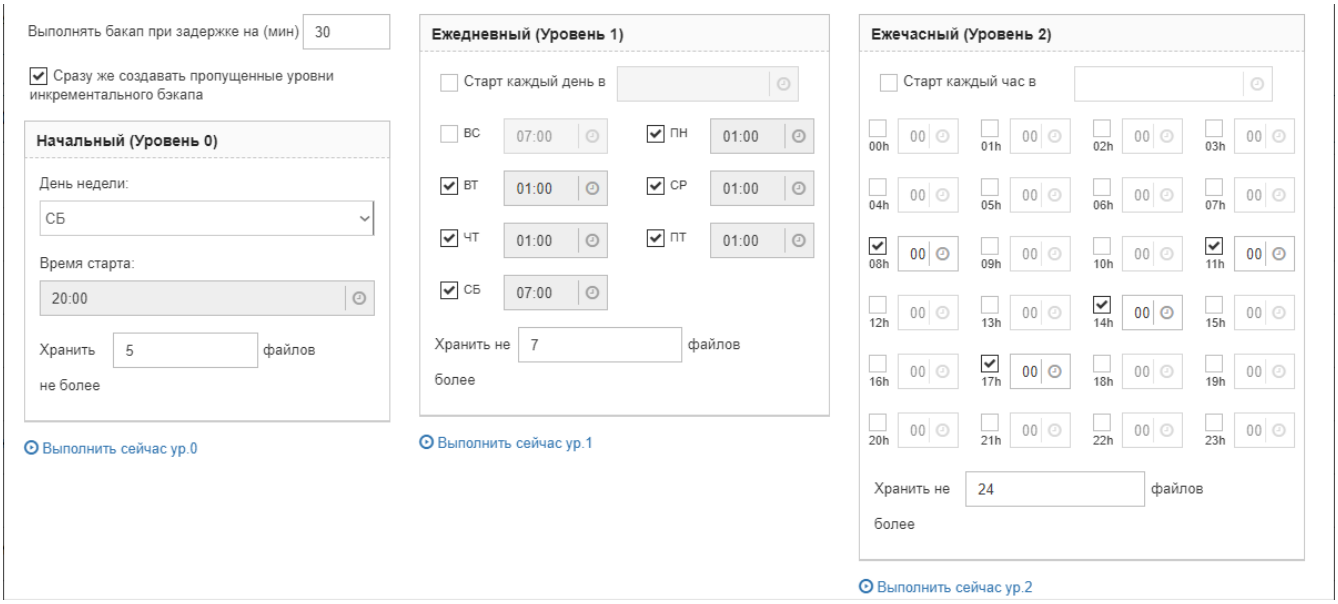
**Не проверять цепочку GUID** — эту опцию следует выбрать, если вы хотите пропустить проверку существования предыдущих уровней инкрементных резервных копий.

**Сразу же создавать пропущенные уровни инкрементального бэкапа** — по умолчанию эта опция включена. Это означает, что если вы запланировали начальный момент запуска резервного копирования уровня 1 раньше, чем начальный момент запуска резервного копирования уровня 0, DataGuard автоматически это исправит и создаст резервную копию уровня 0 непосредственно перед уровнем 1. Следующие резервные копии уровня 0 будут выполнены по обычному графику.

**Высылать уведомления для уровней 0,1,2** — включите эту опцию, чтобы получать уведомления об инкрементальных резервных копиях (*настоятельно рекомендуется*!)

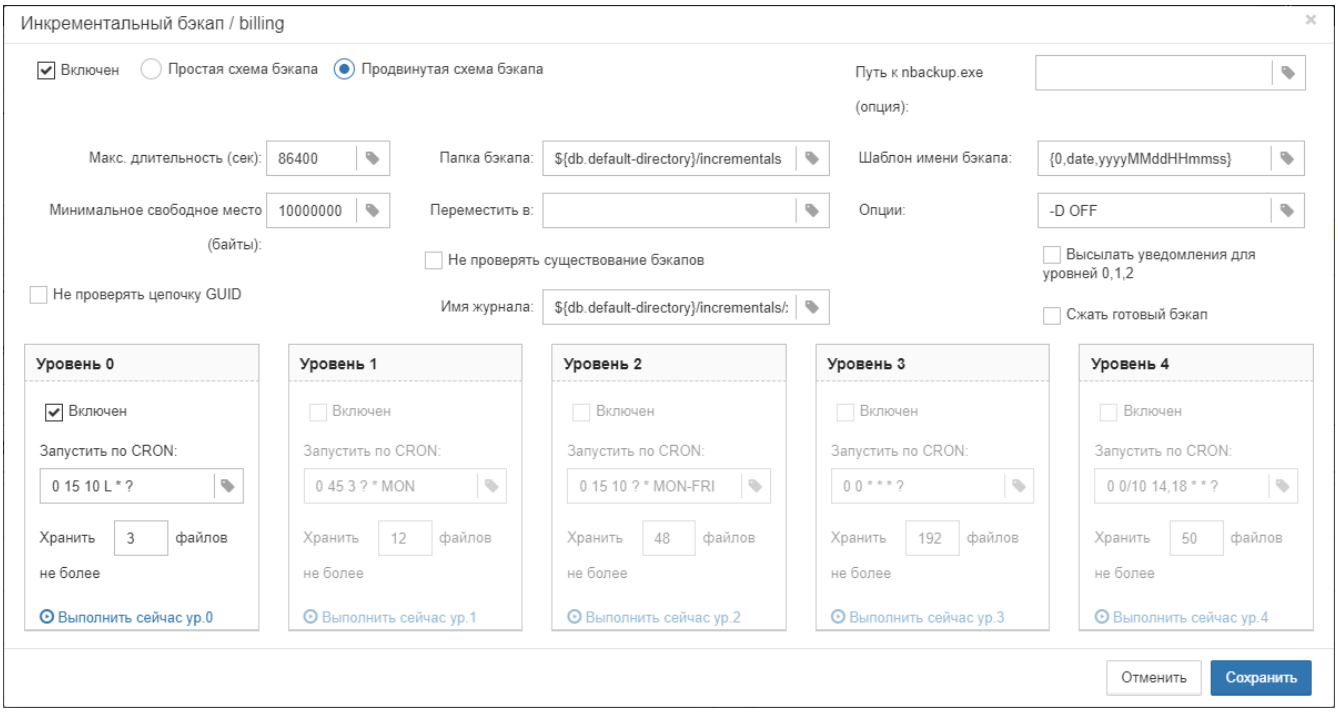
После настройки основного набора параметров необходимо настроить само расписание. Как вы можете видеть на скриншоте ниже, вам необходимо указать день недели и время для резервного копирования уровня 0 (еженедельного), дни недели и время для запуска резервного копирования уровня 1 (ежедневного), а также часы и минуты для резервной копии уровня 3 (ежечасного).

Для каждого уровня резервного копирования вы можете указать, сколько файлов хранить в истории.




По умолчанию настроено сохранение 5 резервных копий еженедельно, 7 ежедневных и 24-часовых резервных копий.

Однако иногда требуется более гибкое расписание, для этого в виджете “Инкрементальный бэкап” имеется “Продвинутая схема бэкапа”:




Как видите, верхняя часть экрана конфигурации такая же, как и в простом расписании, а разница заключается в способе планирования уровней резервного копирования.

Продвинутая схема бэкапа позволяет настроить до 5 уровней резервного копирования и гибко планировать их Выражения CRON.

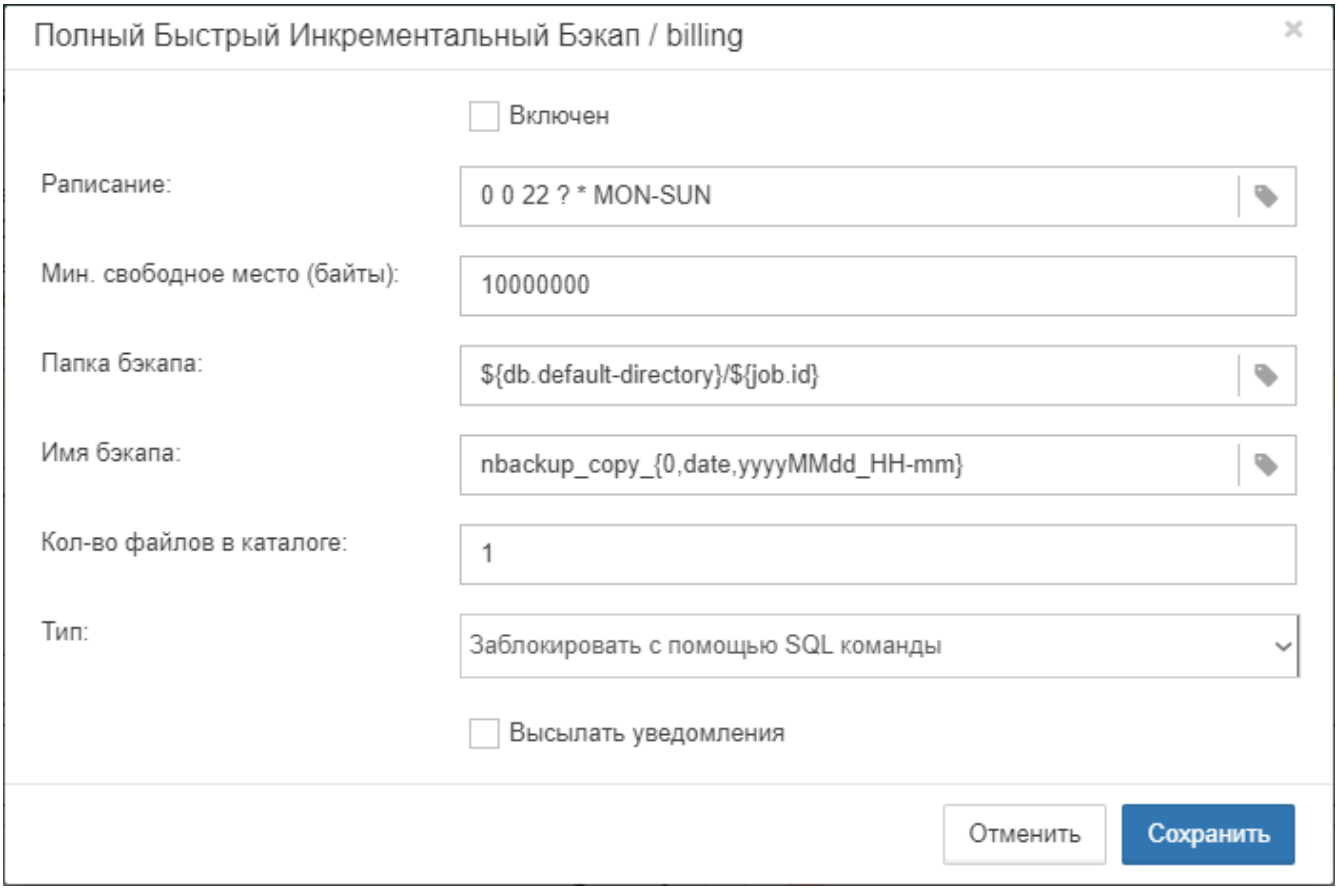
Например, вы можете настроить её на создание полной резервной копии (уровень 0) каждые 3 месяца, копии уровня 1 каждый месяц, уровня 2 — каждую неделю, уровня 3 — каждый день и уровня 4 — каждый час.

Если вы отслеживаете более одной базы данных, настоятельно рекомендуется разделить время выполнения резервных копий.

### 3.4.8. База данных: Полный Инкрементальный Бэкап

Это задание также использует функцию nbackup в Firebird, но, в отличие от многоуровневого резервного копирования, оно всегда выполняет полную копию (уровень 0) базы данных. Такая работа полезна для быстрого создания копии рабочей базы данных.

Конфигурация “База данных: Полный Инкрементальный Бэкап” тривиальна:




Вам просто нужно настроить, когда и куда DataGuard должен копировать полную копию (инкрементная резервная копия уровня 0) и сколько копий он должен хранить.

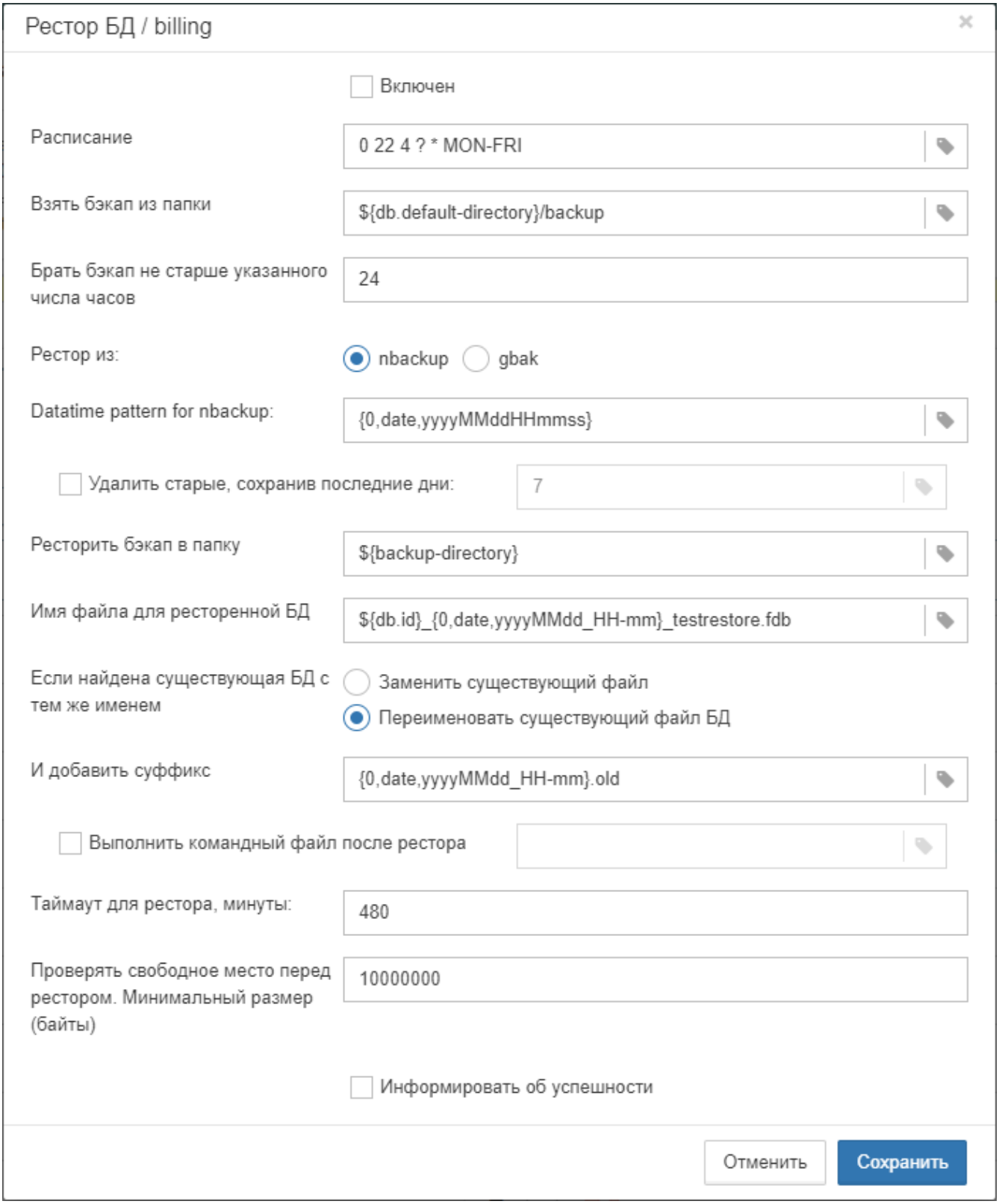
### 3.4.9. База данных: Рестор БД

Одной из частых задач администраторов баз данных является восстановление базы данных из резервной копии. Причин для восстановления может быть много, наиболее распространенными являются регулярная проверка сохраненных резервных копий и необходимость иметь свежую восстановленную копию для быстрого отката. HQbird FBDataGuard может автоматизировать восстановление резервных копий (которые были созданы с помощью gbak или “База данных: Бэкап”) с помощью задания **База данных: Рестор БД**. Рассмотрим варианты и параметры данного задания.

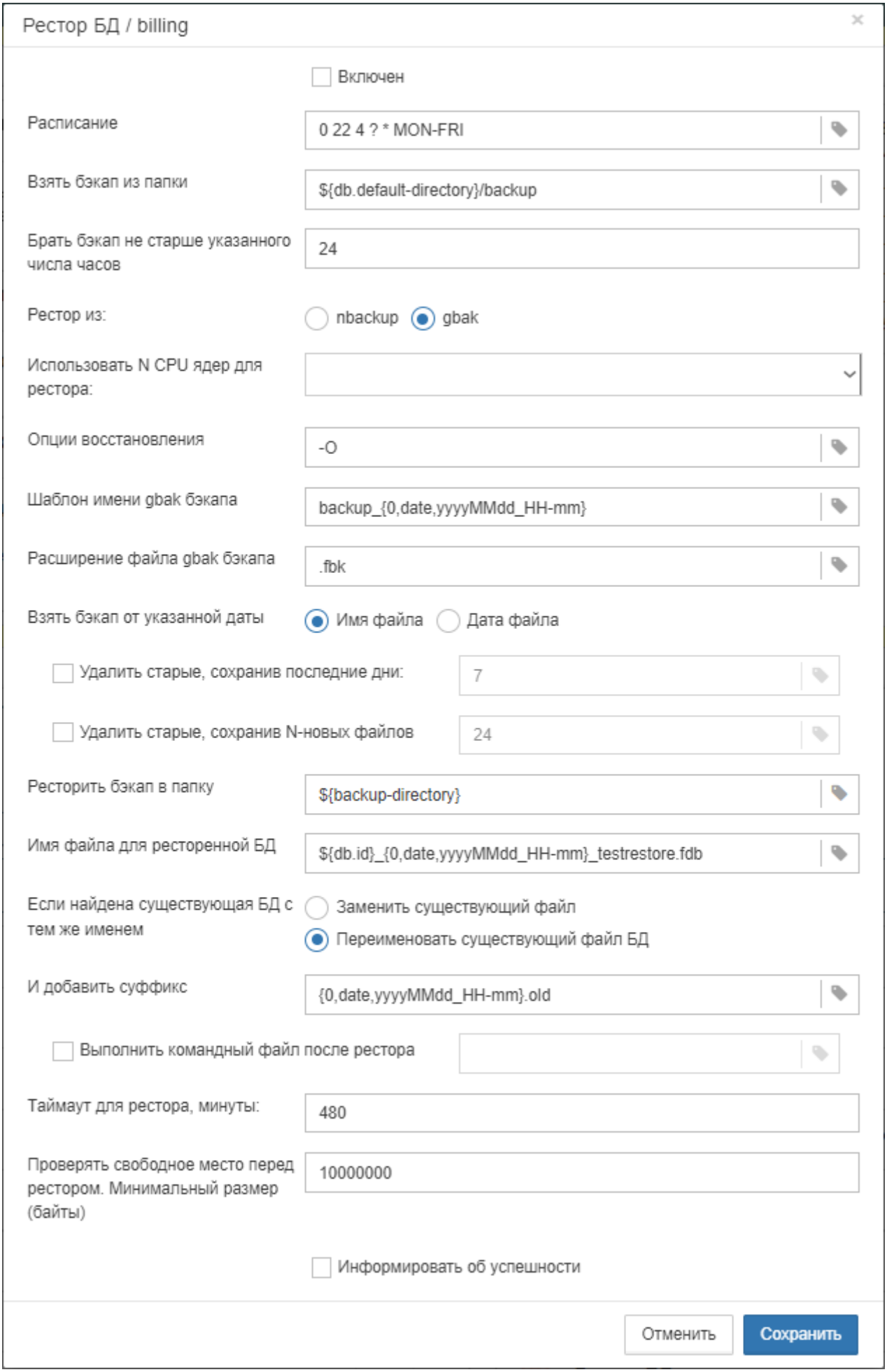

По умолчанию восстановление отключено. Поскольку восстановление может оказаться длительным и ресурсоемким, тщательно планируйте время восстановления.

Базу данных можно восстановить из резервных копий разных типов. Чтобы указать, какие типы резервных копий будут использоваться при восстановлении, используйте переключатель **Рестор из**.

Ниже вы можете увидеть диалог настройки **База данных: Рестор БД** в режиме **nbackup**:




В режиме **gbak** диалог настройки **База данных: Рестор БД** выглядит так:


Рестор БД / billing
Включен
Расписание
0 22 4 ? * MON-FRI
Взять бэкап из папки
${db.default-directory}/backup
Брать бэкап не старше указанного числа часов
24
Рестор из:
nbackup
gbak
Использовать N CPU ядер для рестора:
Опции восстановления
-O
Шаблон имени gbak бэкапа
backup_{0,date,yyyyMMdd_HH-mm}
Расширение файла gbak бэкапа
.fbk
Взять бэкап от указанной даты
Имя файла
Дата файла
Удалить старые, сохранив последние дни:
7
Удалить старые, сохранив N-новых файлов
24
Ресторить бэкап в папку
${backup-directory}
Имя файла для ресторенной БД
${db.id}_{0,date,yyyyMMdd_HH-mm}_testrestore.fdb
Если найдена существующая БД с тем же именем
Заменить существующий файл
Переименовать существующий файл БД
И добавить суффикс
{0,date,yyyyMMdd_HH-mm}.old
Выполнить командный файл после рестора
Таймаут для рестора, минуты:
480
Проверять свободное место перед рестором. Минимальный размер (байты)
10000000
Информировать об успешности
Отменить
Сохранить

- Поле “**Расписание**” содержит выражение CRON, которое определяет когда будет запускаться восстановление.
- “**Взять бэкап из папки**” — указывает расположение файлов резервных копий, которые необходимо восстановить. Если вы восстанавливаете резервные копии на том же компьютере, где они были созданы, укажите ту же папку, что и в задании “База данных: Бэкап”. Если вы восстанавливаете резервные копии с другого компьютера, укажите папку, в которой находятся эти резервные копии.
- “**Брать бэкап не старше указанного числа часов**” — этот параметр определяет максимальный возраст резервной копии, подлежащей восстановлению. Если последний файл резервной копии будет старше указанного количества часов, то задание “Рестор БД” отправит предупреждение о том, что резервная копия слишком старая. Это полезно для автоматической проверки резервных копий, созданных на удаленном компьютере.
- “**Рестор из**” указывает, какие типы резервных копий будут использоваться для восстановления базы данных.
- “**Datatime pattern for nbackup**” содержит шаблон для имен резервных копий, созданных с помощью nbackup. Оно должно быть таким же, как **Шаблон имени бэкапа** см. База данных: Инкрементальный бэкап.
- “**Шаблон имени gbak бэкапа**” содержит шаблон для имен резервных копий. Оно должно быть таким же, как **Шаблон имени бэкапа** см. База данных: Бэкап.
- “**Расширение файла gbak бэкапа**” — по умолчанию `.fbk`.
- “**Использовать N CPU ядер для рестора**” — доступно только в режиме gbak.
- “**Опции восстановления**” — доступно только в режиме gbak.
- “**Ресторить бэкап в папку**” — папка, в которую FBDataGuard будет восстанавливать резервные копии.
- “**Имя файла для ресторенной БД**” — шаблон восстанавливаемого файла базы данных. По умолчанию он содержит следующие части
  - `${db.id}_{0,date, yyyyMMdd_HH-mm}_testrestore.fdb`
  - `db.id` — внутренний идентификатор базы данных (GUID)
  - `0,date, yyyyMMdd_HH-mm` — дата и время
  - `testrestore.fdb` — описание (Вы можете указать там любое имя файла, которое вам нужно).
- “**Если найдена существующая БД**” — если FBDataGuard обнаружит в папке назначения файл с тем же именем, что и восстанавливаемая база данных, то по умолчанию он переименует существующий файл. Если вы хотите заменить старый восстановленный файл на новый, то выберите “Заменить существующий файл”.
- “**И добавить суффикс**” — если вы выбрали “Переименовать существующий файл БД”, то этот суффикс будет использоваться для его переименования.
- “**Выполнить командный файл после рестора**” — в этом поле вы можете указать необязательный путь к командному файлу или другой утилите, которая будет запускаться после восстановления. Будет передано 2 параметра: первый — путь к только что восстановленной резервной копии, второй — путь к восстановленному файлу.

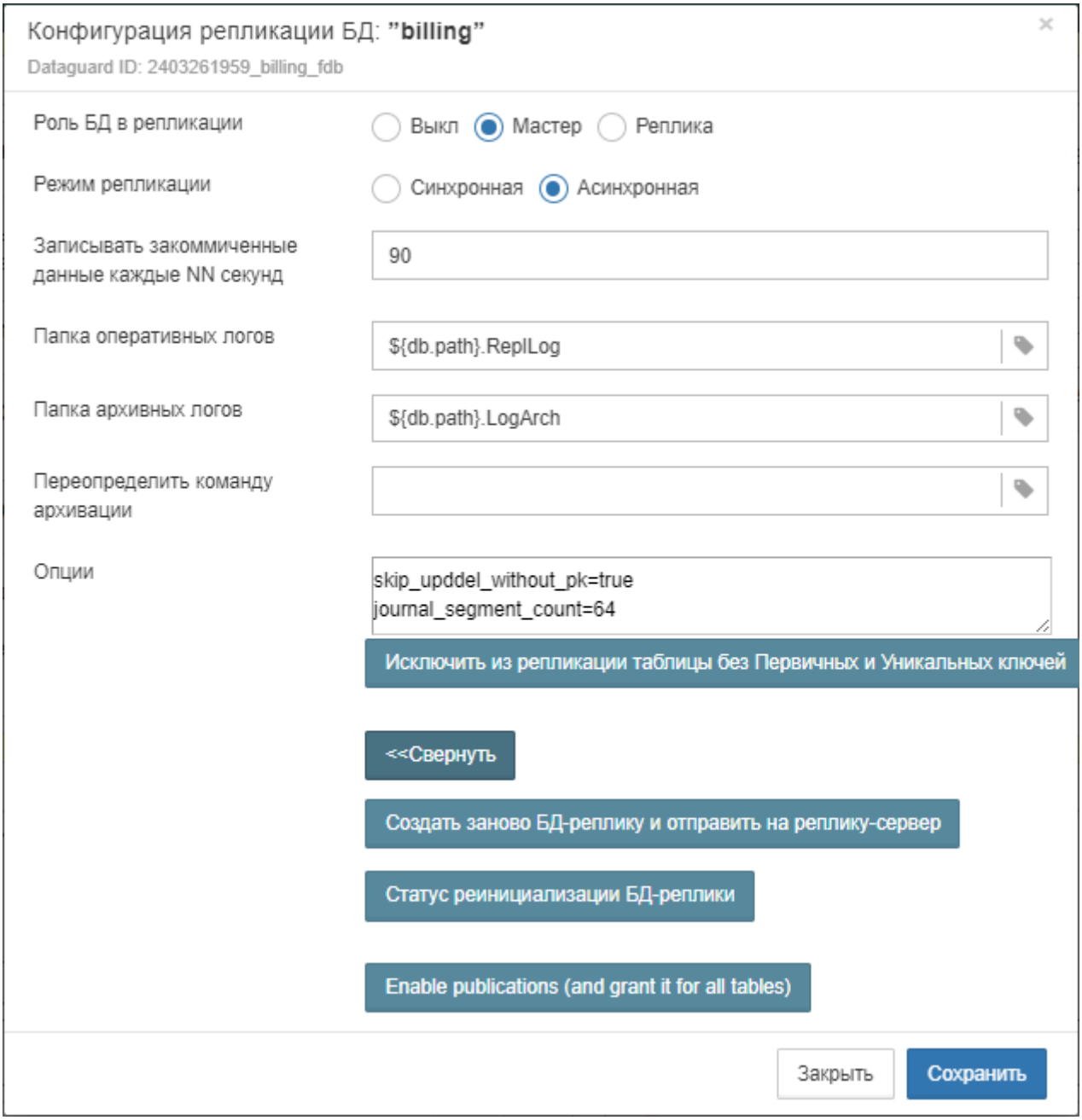
- “**Таймаут для рестора, минуты**” — здесь вы можете установить ограничение по времени для операции восстановления. Если этот лимит будет превышен, то будет отправлено предупреждение о том, что восстановление занимает слишком много времени.
- “**Проверять свободное место перед рестором. Минимальный размер (байты)**” — здесь вы можете установить ограничение на минимального доступное свободное места в папке назначения восстановления — если свободного места меньше, чем указано, то восстановление не начнется, и будет отправлено соответствующее предупреждение.
- “**Информировать об успешности**” — отправить электронное письмо об успешном восстановлении (по умолчанию отключено, будут отправляться только оповещения о проблемах).

### 3.4.10. База данных: Передача сегментов репликации

Целью задания “Передача сегментов репликации” является отправка сегментов репликации, созданных в результате асинхронной репликации, с главного сервера на сервер реплики. В случае распределенной среды асинхронной репликации, когда сетевое соединение между главным сервером и сервером-репликой нестабильно, с большой задержкой или когда серверы находятся в разных географических регионах, лучшим способом передачи сегментов репликации будет FTP или FTP через SSH.

Ниже мы рассмотрим, как настроить Cloud Backup для этой задачи.

Во-первых, мастер асинхронной репликации должен быть настроен на сохранение сегментов репликации в какой-нибудь локальной папке — по умолчанию это будет `${db.path}.LogArch` — как показано в примере ниже:


Конфигурация репликации БД: "billing"
Dataguard ID: 2403261959_billing_fdb
Роль БД в репликации
Выкл
Мастер
Реплика
Режим репликации
Синхронная
Асинхронная
Записывать закоммиченные данные каждые NN секунд
90
Папка оперативных логов
${db.path}.ReplLog
Папка архивных логов
${db.path}.LogArch
Переопределить команду архивации
Опции
skip_upddel_without_pk=true
journal_segment_count=64
Исключить из репликации таблицы без Первичных и Уникальных ключей
<<Свернуть
Создать заново БД-реплику и отправить на реплику-сервер
Статус реинициализации БД-реплики
Enable publications (and grant it for all tables)
Закрыть
Сохранить

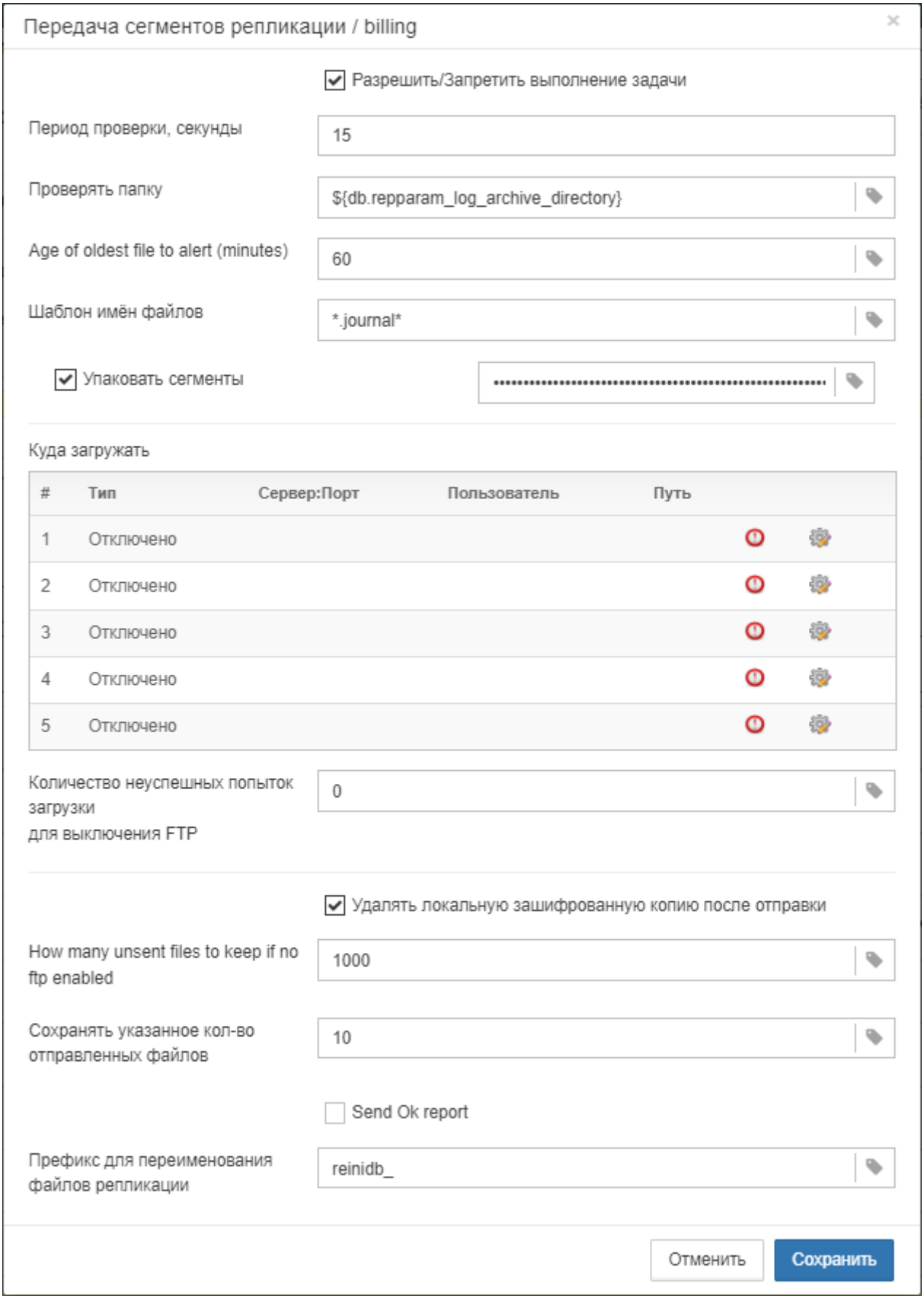



*Рисунок 34. Конфигурация передачи сегментов репликации*

Затем мы можем настроить задание **Передача сегментов репликации**, чтобы отслеживать в этой папке новые сегменты репликации и загружать их на удаленный FTP-сервер.

Как вы можете видеть на скриншоте выше, задание **Передача сегментов репликации**

проверяет папку, указанную в “**Проверять папку**” с интервалом, указанным в “**Период проверки, секунды**”.

Обратите внимание: **Передача сегментов репликации** отправляет файлы в порядке их имен, а не дат.

Чтобы проверить, что переданные файлы являются действительными сегментами репликации, а также для поддержки автоматической повторной инициализации баз данных реплик, необходимо включить флажок “**Разрешить/Запретить выполнение задачи**”.

По умолчанию задание сжимает и шифрует сегменты репликации перед их отправкой. Пароль по умолчанию — “**zipmasterkey**” (без кавычек), который можно указать в поле рядом с переключателем “**Упаковать сегменты**”. FBDataGuard создает сжатую и зашифрованную копию сегмента репликации и загружает ее на указанный целевой сервер.

Чтобы отключить упаковку и шифрование, снимите флажок “**Упаковать сегменты**”.

Последняя часть параметров в диалоговом окне Cloud Backup позволяет контролировать поведение облачного резервного копирования.

Флажок **Send Ok report** — включает отправку электронного письма на указанный в оповещениях адрес каждый раз при загрузке сегмента репликации. По умолчанию он выключен.

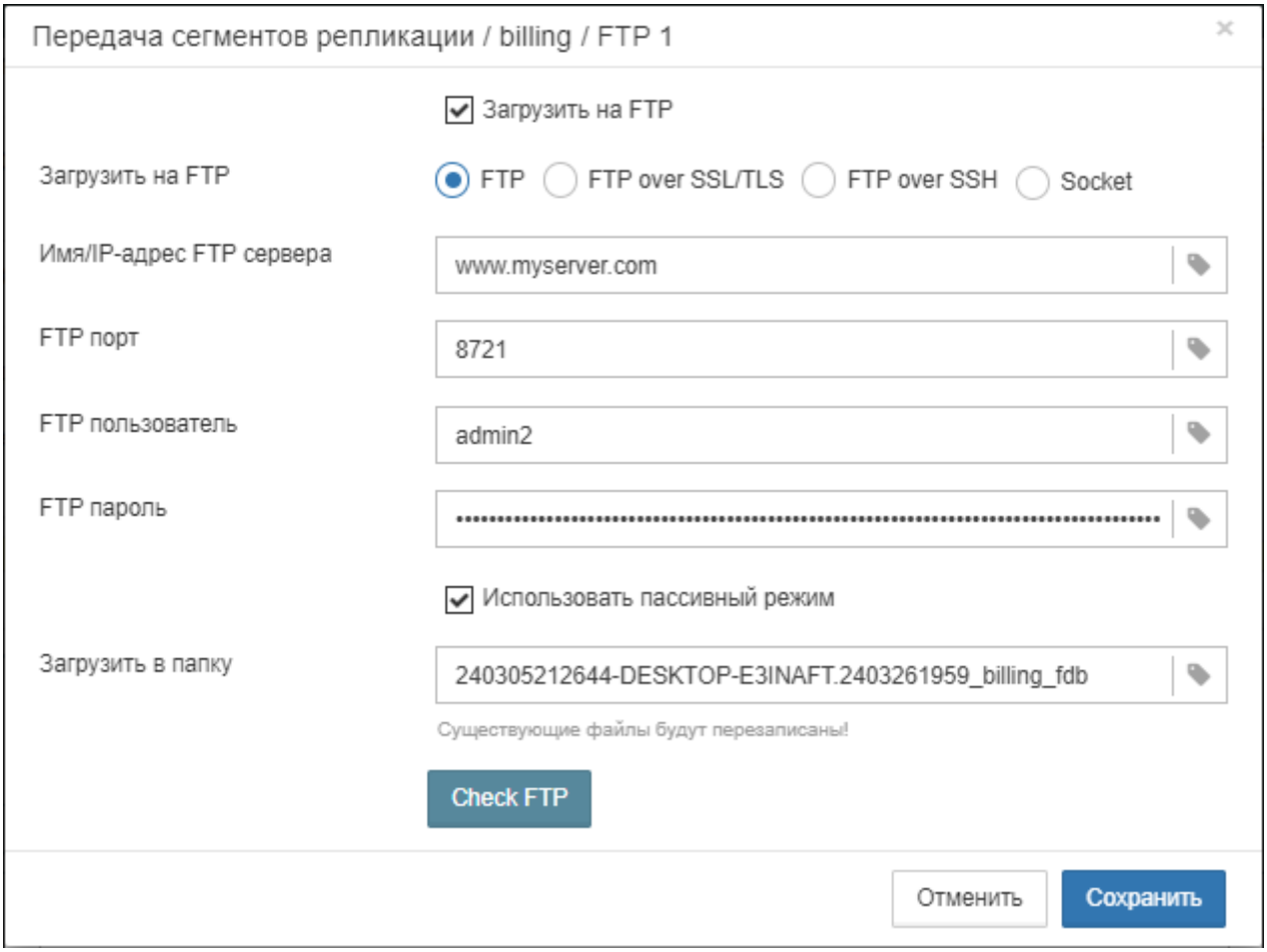
В результате FBDataGuard загрузит зашифрованные и сжатые сегменты репликации на удаленный сервер. Чтобы распаковать и расшифровать их в обычные сегменты репликации, на сервере реплики должен быть установлен еще один экземпляр HQbird FBDataGuard и настроено задание Cloud Backup Receiver — подробнее см. в разделе База данных: Приемник файлов.

Далее рассмотрим настройки каждого из протоколов передачи файлов.

### FTP/FTPS/FTPS через SSH

Существует несколько типов целевых серверов: FTP, FTP через SSL/TLS, FTP через SSH. При выборе необходимого типа в диалоговом окне отображаются обязательные для заполнения поля.

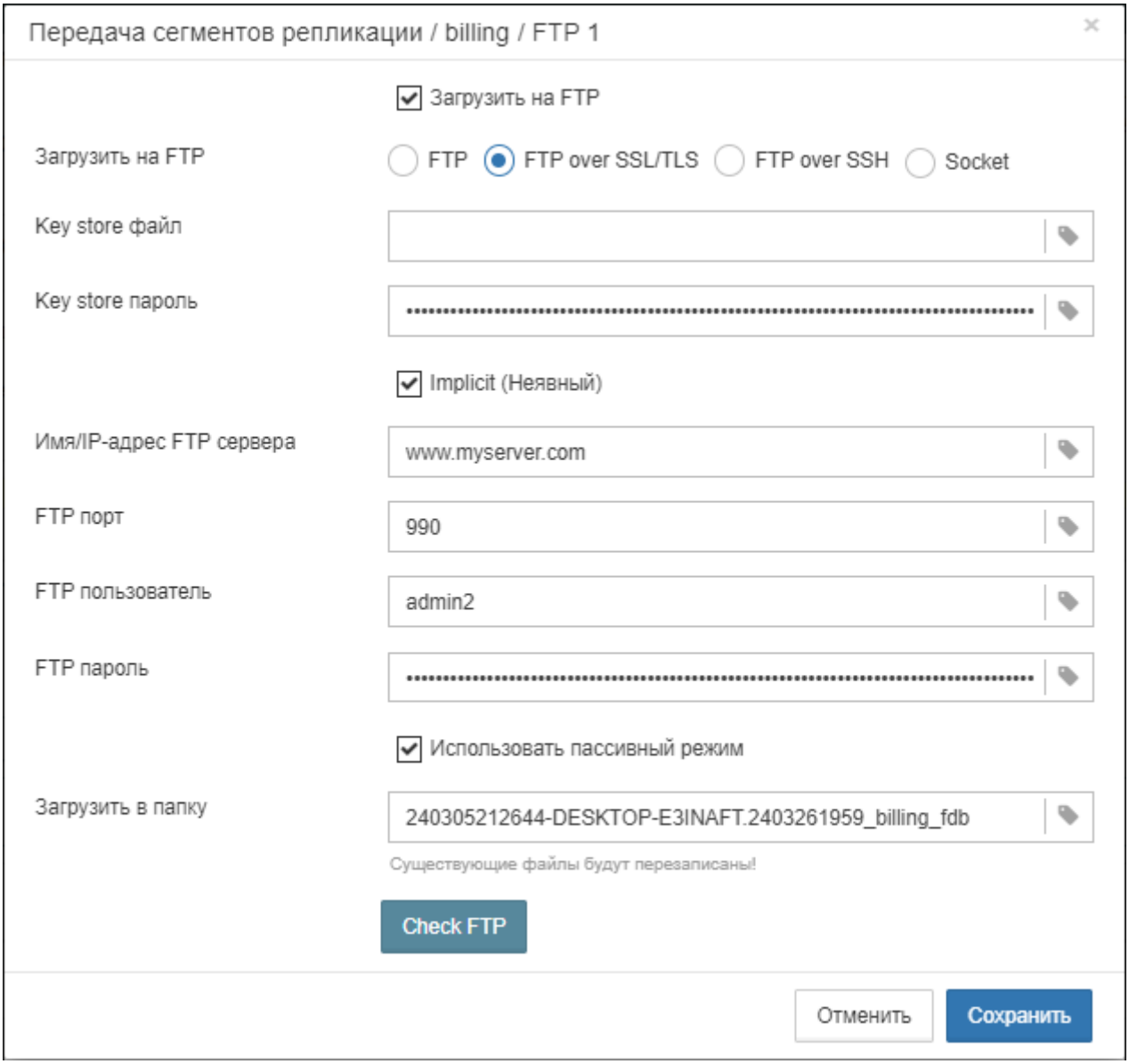
Вы можете выбрать до 5 одновременных удаленных серверов для загрузки резервных копий. Ниже вы можете увидеть диалог настройки FTP.




Если у вас не установлен FTP на целевом сервере с Windows, установите Filezilla — это очень популярный быстрый и легкий FTP-сервер для Windows.

Сегменты репликации будут загружены в подкаталог, указанный в поле “Загрузить в папку”. По умолчанию это `/dababase0/${db.id}`, где `db.id` — это идентификатор базы данных внутри DataGuard. Реплика об этом `db.id` ничего не знает, поэтому нужно прописать её вручную в “Распаковывать файлы в папку” (см. База данных: Приемник файлов).

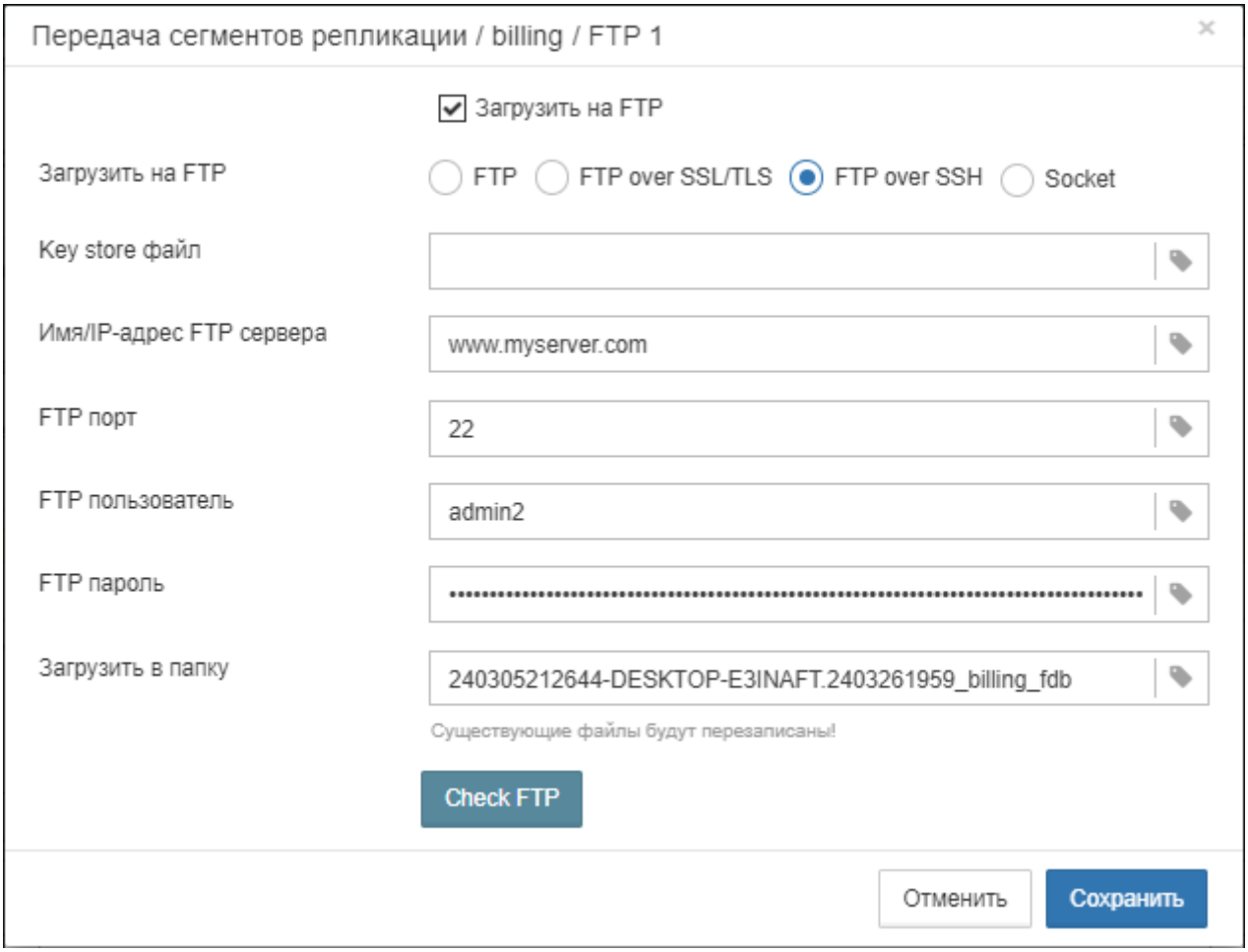
#### FTP через SSL/TLS




Для отправки файлов на FTPS необходимо создать jks-хранилище с файлом закрытого ключа, указать путь к нему в поле “Key store file” и пароль к нему в поле “Key store password”.

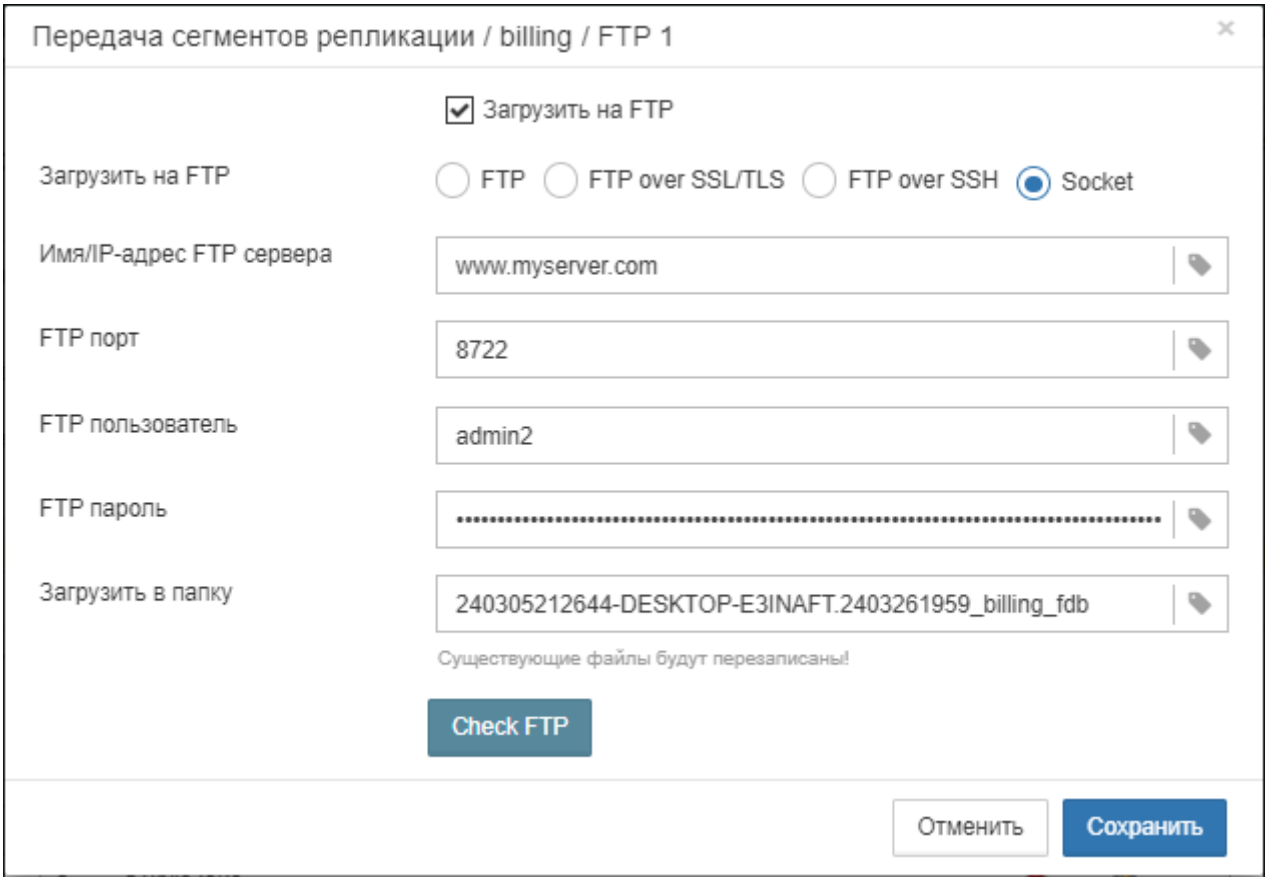
Подробности и пример создания файла jks и пароля см. здесь: http://xacmlinfo.org/2014/06/13/how-to-keystore-creating-jks-file-from-existing-private-key-and-certificate/

#### FTP через SSH




Чтобы использовать FTP через SSH с аутентификацией по закрытому ключу, укажите полный путь к нему в “Key store file”, остальные параметры аналогичны обычному FTP.

#### Socket




### 3.4.11. База данных: Отправка файлов

Целью задания “Отправка файлов” является отправка файлов резервных копий с главного сервера на сервер-реплику. В случае распределенной среды, когда сетевое соединение между главным сервером и сервером-репликой нестабильно или имеет высокую задержку, или когда серверы находятся в разных географических регионах, лучшим способом передачи файлов будет через FTP или FTP через SSH.

Ниже мы рассмотрим, как настроить “Отправка файлов” для этой задачи.

Во-первых, сервер базы данных должен быть настроен на сохранение файлов резервных копий в какой-либо локальной папке — по умолчанию это будет `${db.default-directory}/backup` — как показано в примере ниже:

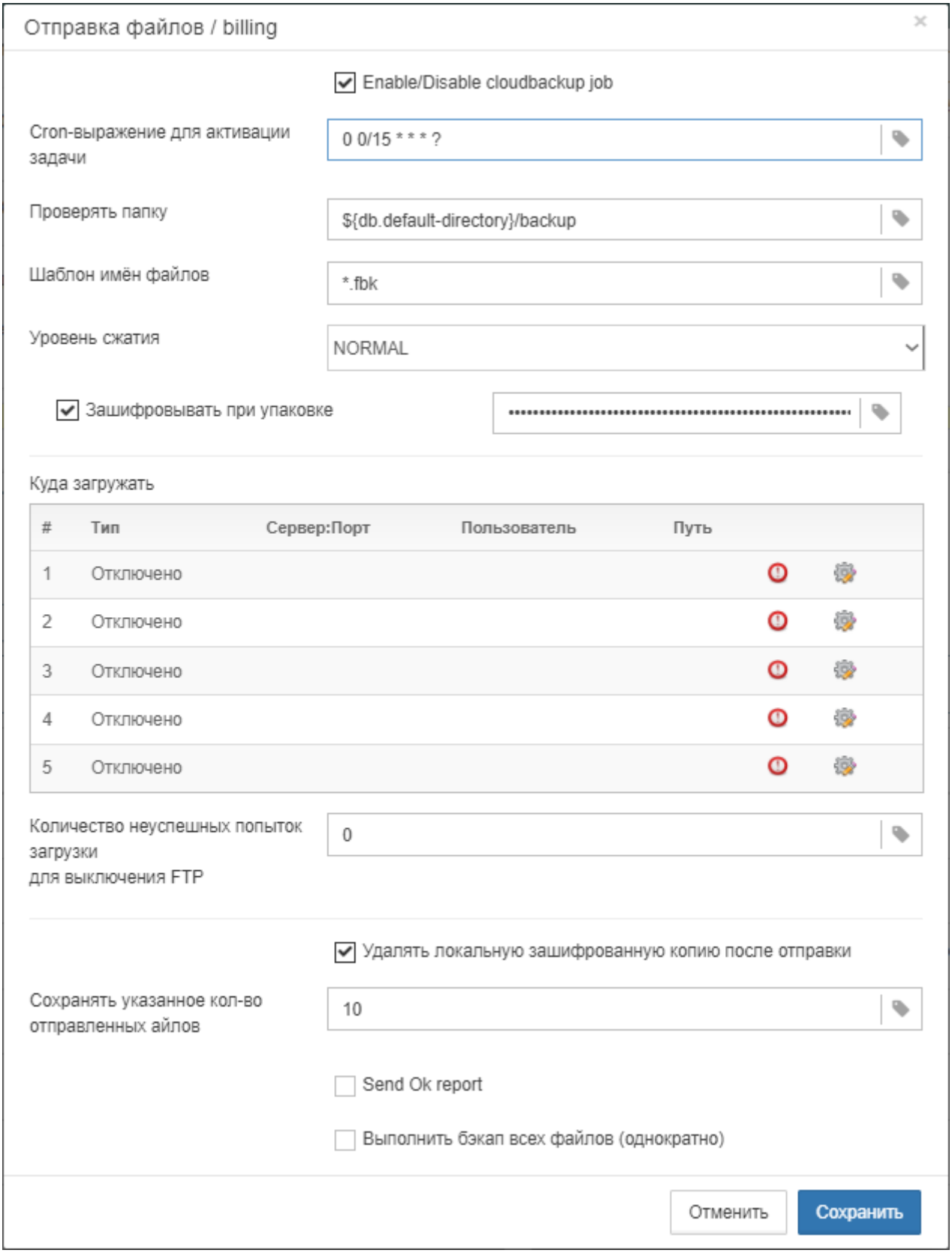



*Рисунок 35. Конфигурация отправки файлов*

Затем мы можем настроить задание **Отправка файлов**, чтобы отслеживать в этой папке новые файлы резервных копий и загружать их на удаленный FTP-сервер.

Как видно на скриншоте выше, задание **Передача файлов** проверяет папку, указанную в “Проверять папку”, согласно расписанию, указанным в “Cron-выражение для активации задачи”. Обратите внимание: **Передача файлов** отправляет файлы в порядке их имен, а не

дат.

По умолчанию задание **Отправка файлов** сжимает и шифрует файлы резервных копий перед их отправкой. Пароль по умолчанию — “**zipmasterkey**” (без кавычек), который можно указать в поле “**Зашифровывать при упаковке**”. FBDataGuard создает сжатую и зашифрованную копию резервной копии и загружает ее на указанный целевой сервер.

Чтобы отключить шифрование, снимите флажок “**Зашифровывать при упаковке**”.

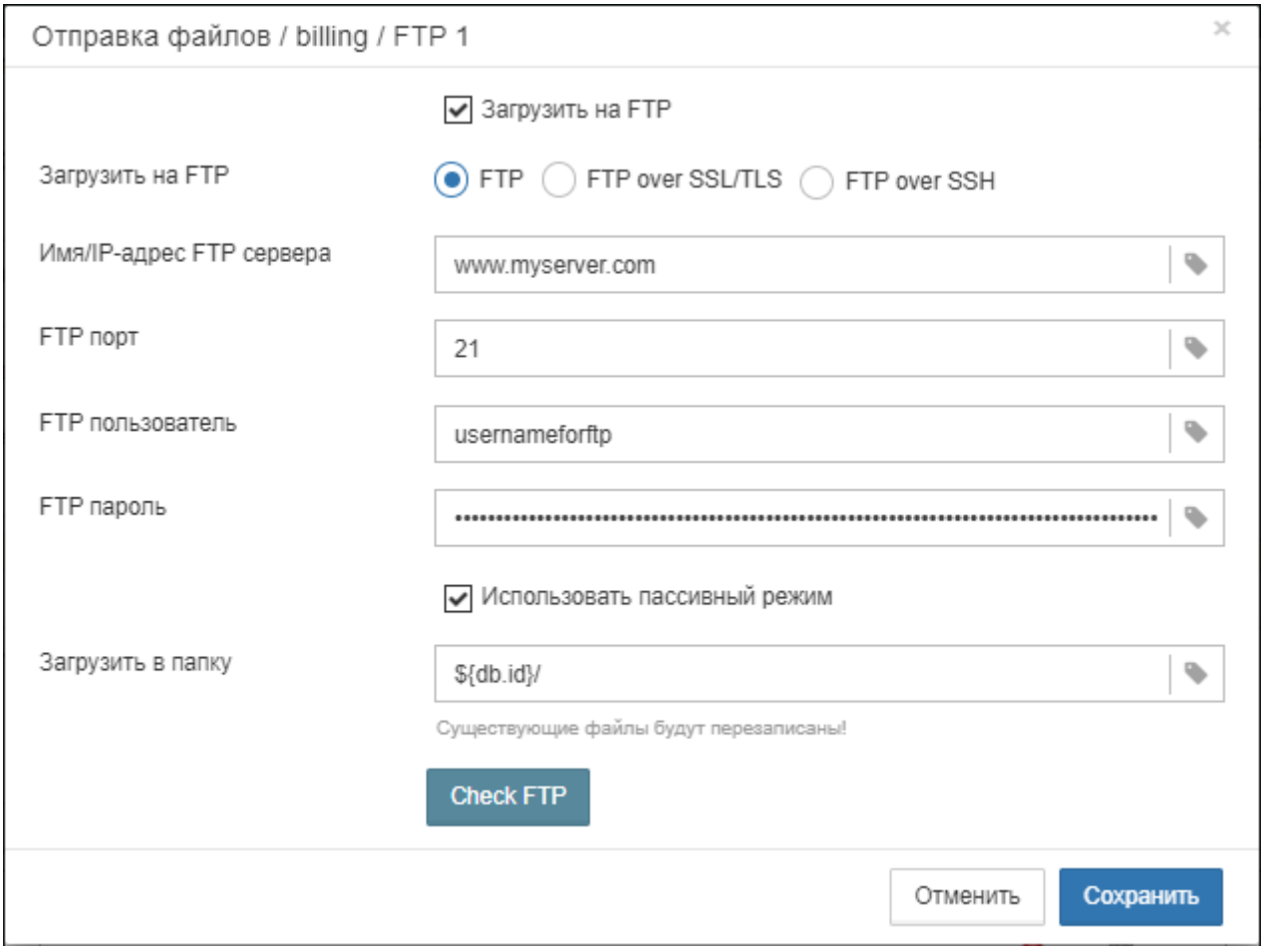
В результате FBDataGuard загрузит зашифрованные и сжатые файлы на удаленный сервер. Чтобы распаковать и расшифровать их в обычные файлы, на сервере реплики должен быть установлен еще один экземпляр HQbird FBDataGuard и настроено задание File Receiver — подробнее см. в разделе База данных: Приемник файлов.

Далее рассмотрим настройки каждого из протоколов передачи файлов.

### FTP/FTPS/FTPS через SSH

Существует несколько типов целевых серверов: FTP, FTP через SSL/TLS, FTP через SSH. При выборе необходимого типа в диалоговом окне отображаются обязательные для заполнения поля.

Вы можете выбрать до 5 одновременных удаленных серверов для загрузки резервных копий. Ниже вы можете увидеть диалог настройки FTP.

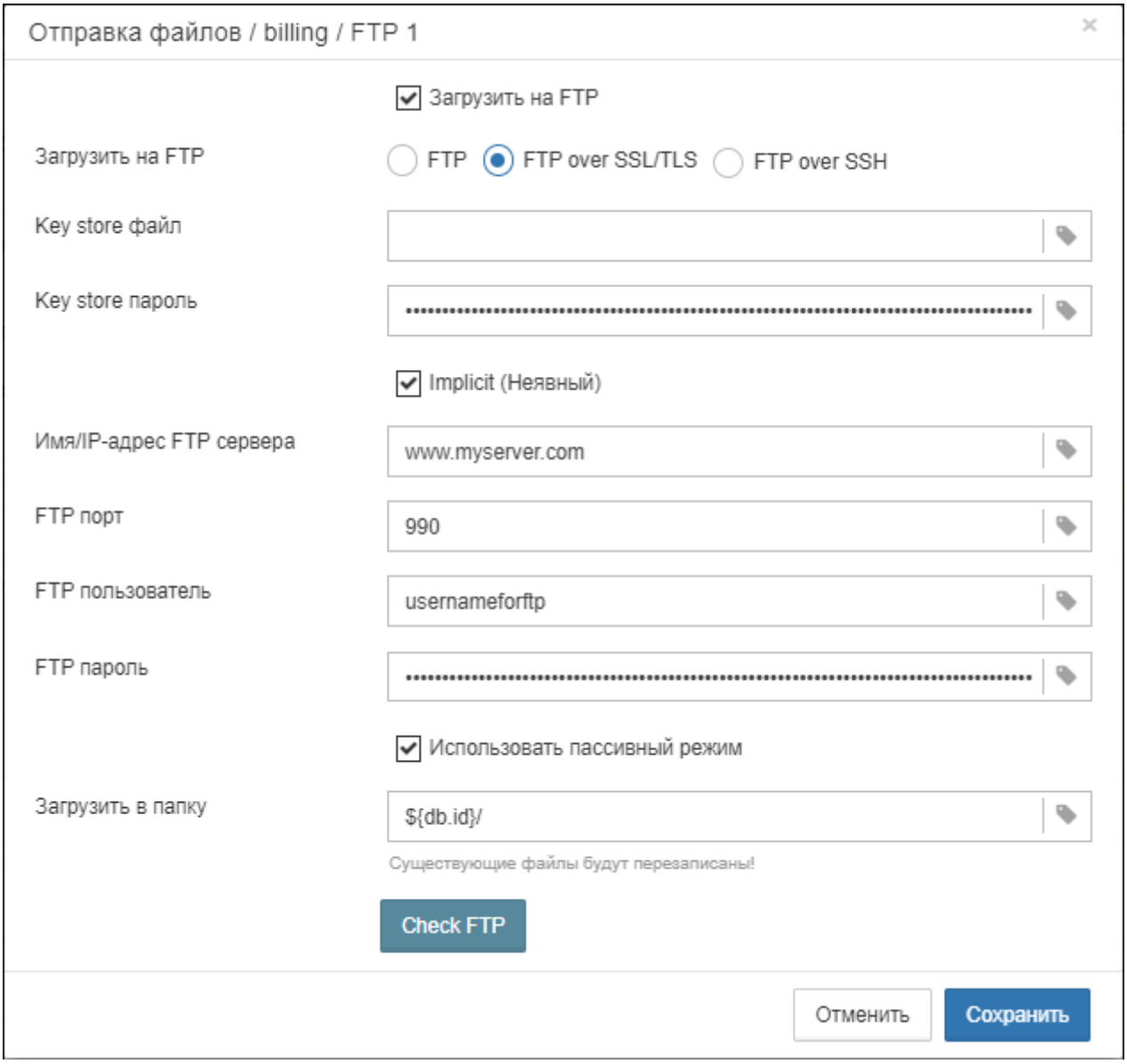



Если у вас не установлен FTP на целевом сервере с Windows, установите

Filezilla — это очень популярный быстрый и легкий FTP-сервер для Windows.

Сегменты репликации будут загружены в подкаталог, указанный в поле “Загрузить в папку”. По умолчанию это `${db.id}/`, где `db.id` — это идентификатор базы данных внутри DataGuard. Реплика об этом `db.id` ничего не знает, поэтому нужно прописать её вручную в “Распаковывать файлы в папку” (см. База данных: Приемник файлов).

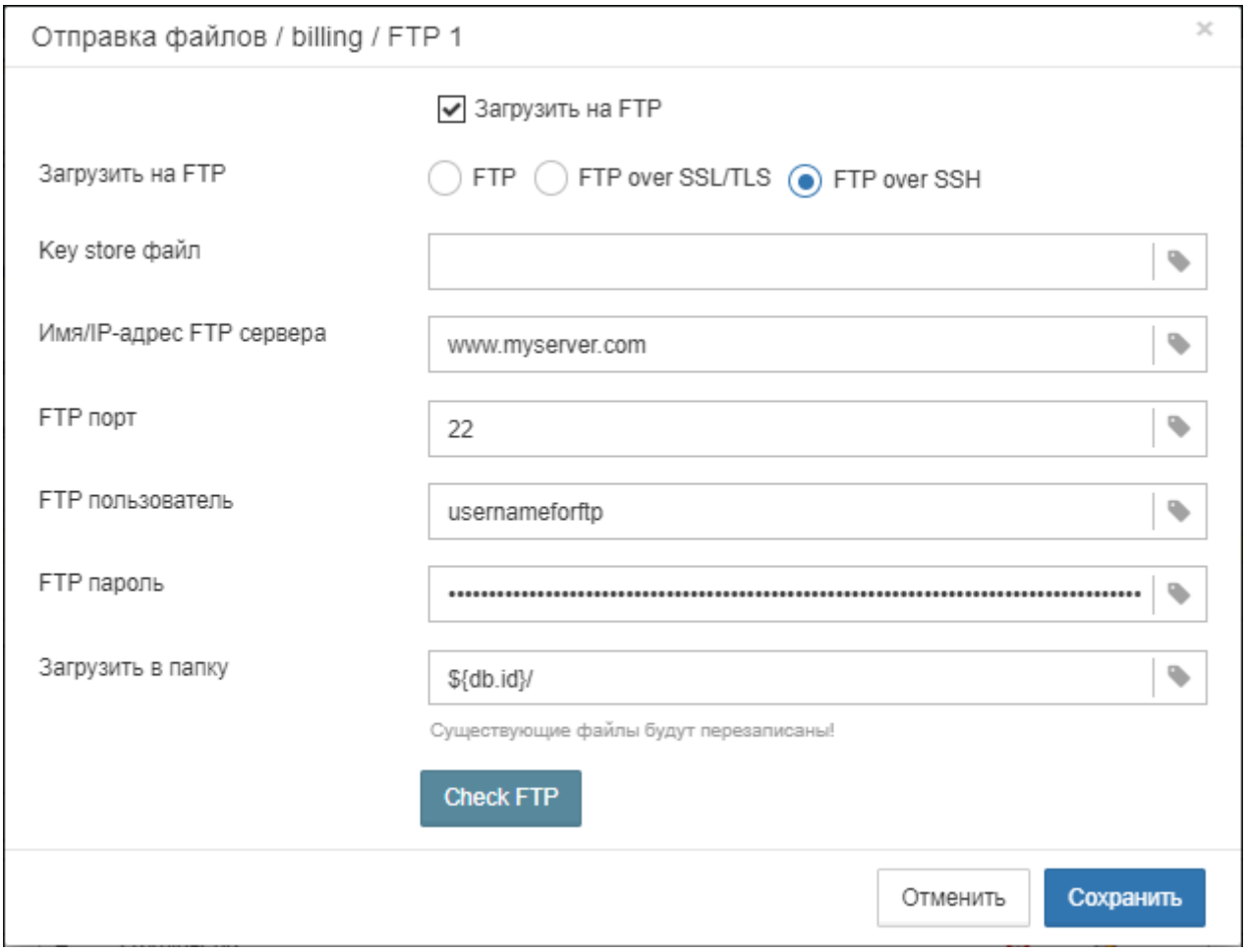
#### FTP через SSL/TLS




Для отправки файлов на FTPS необходимо создать jks-хранилище с файлом закрытого ключа, указать путь к нему в поле “Key store file” и пароль к нему в поле “Key store password”.

Подробности и пример создания файла jks и пароля см. здесь: http://xacmlinfo.org/2014/06/13/how-to-keystore-creating-jks-file-from-existing-private-key-and-certificate/

**FTP over SSH**




Чтобы использовать FTP через SSH с аутентификацией по закрытому ключу, укажите полный путь к нему в “Key store file”, остальные параметры аналогичны обычному FTP.

### Отправка проверенных и инкрементальных резервных копий через Cloud Backups

Cloud Backup также можно использовать для отправки любых файлов на FTP/FTPS и т. д. Например, вы можете настроить Cloud Backup для поиска файлов FBK, создаваемых проверенным заданием резервного копирования, и запланировать загрузку на удаленный FTP-сервер.

Необходимо помнить, что количество хранимых резервных копий должно быть меньше количества файлов, сохраняемых Cloud Backup (указанного в параметре “How many files to keep”). По умолчанию Cloud Backup сохраняет 10 последних отправленных файлов, и “Проверенная резервная копия” содержит 5 самых последних файлов резервных копий, поэтому все работает нормально, но если вы уменьшите количество сохраняемых файлов в Cloud Backup, дополнительные файлы будут удалены в соответствии с “Filename template”.

То же самое можно сделать и для инкрементных резервных копий.

## 3.4.12. База данных: Выгрузка Файлов

Целью задачи “Выгрузка Файлов” является перенос файлов из одной директории, доступной DataGuard, в какое-то другое место, обычно удаленное, с возможностью использования различных методов, которые можно подключить к DataGuard в виде плагинов и

выбирается в конфигурации задачи с возможностью установки уникальных для каждого плагина параметров. HQbird включает в себя два плагина для передачи файлов: fpt и sftp. Есть и другие плагины для передачи файлов. Плагины передачи представляют собой файлы `jar` и расположены в папке `Firebird DataGuard/plugins`.

Рассмотрим варианты и параметры данной задачи.

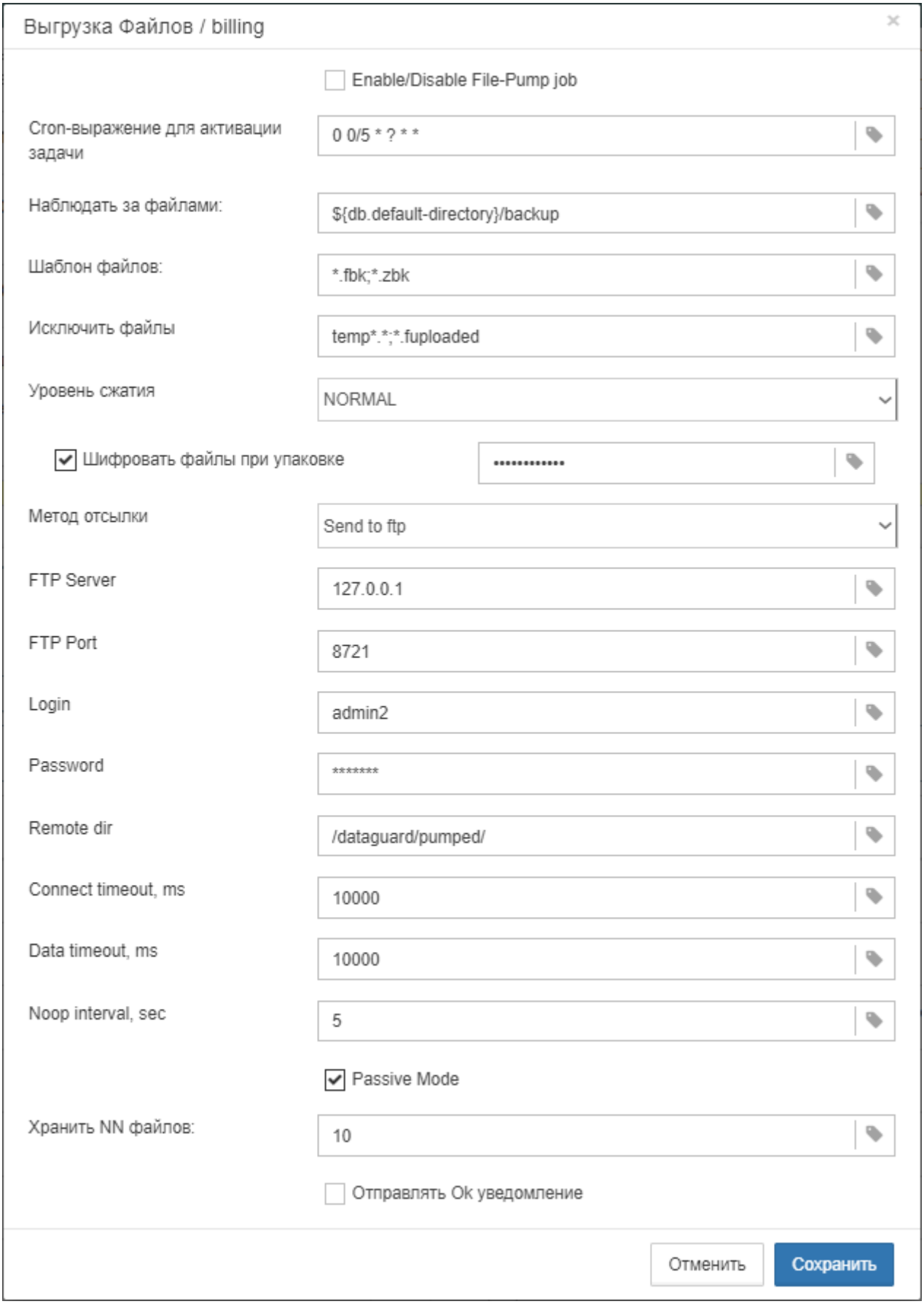



*Рисунок 36. Опции, доступные для плагина передачи файлов по FTP.*

- **Шаблон файлов** — белый список, по которому выбираются файлы для копирования. Представляют маски имен файлов. Должно быть разделено запятой.
- **Исключить файлы** — черный список представляет собой маску имен файлов, которые

следует исключить из передачи. Черный список имеет приоритет над белым списком.

- **Метод отсылки** — метод передачи файлов (плагин).

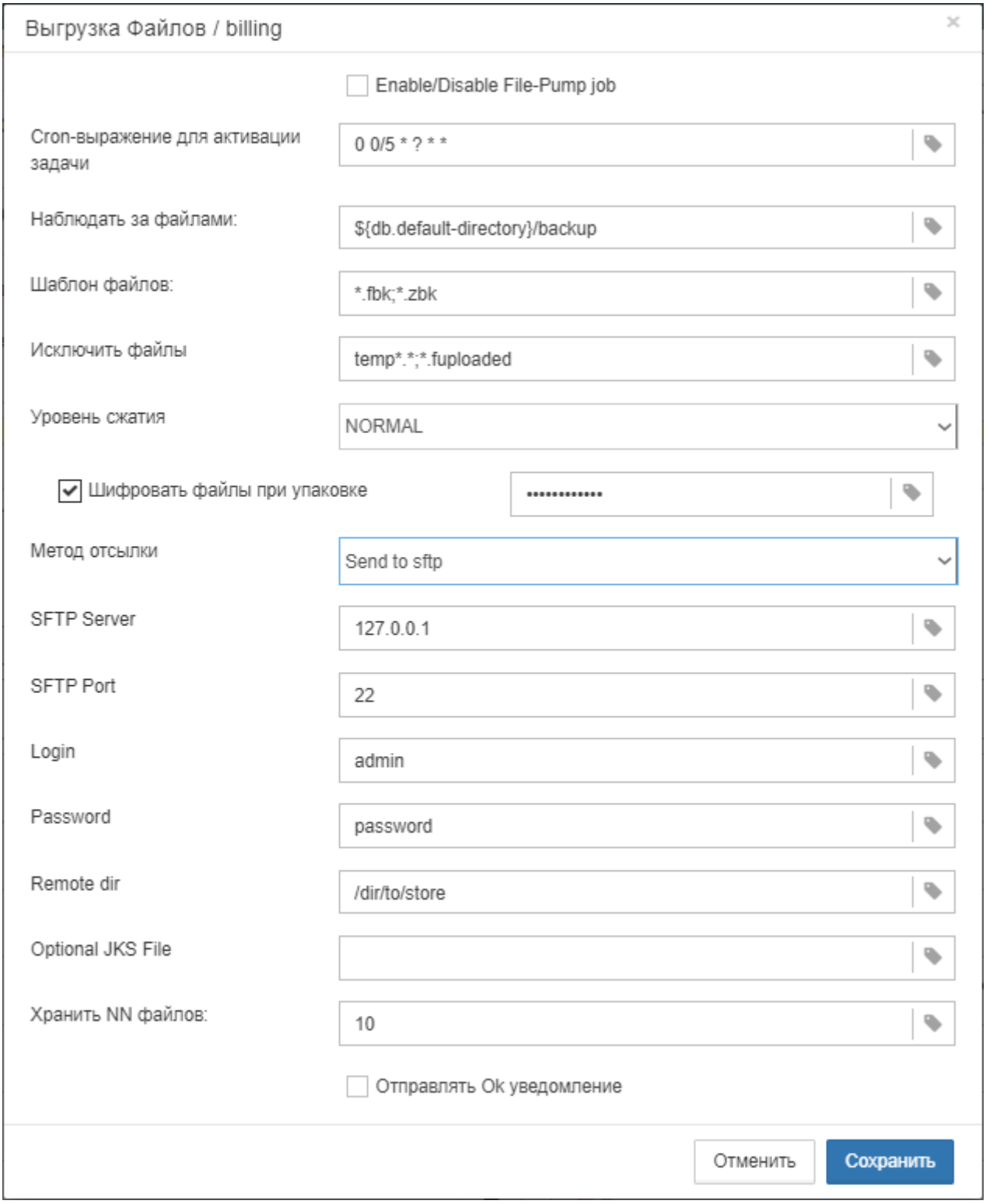



*Рисунок 37. Опции, доступные для плагина передачи файлов по SFTP.*

Алгоритм выполнения этой задачи следующий:

1. На каждой итерации задачи формируется список файлов для каталога для отправки файлов мониторинга. Для выбора списка файлов используются маски: “Шаблон файлов” и “Исключить файлы”.
2. Для каждого выбранного файла (из списка с шага 1, в порядке возрастания даты

последней модификации файла) выполняются следующие действия:

a. Если установлена опция сжатия, файл архивируется (если размер файла не нулевой). Имя сжатого файла формируется путем добавления жестко запрограммированного расширения: `.zipfilepump`. Файл будет архивирован в ту же директорию. Если файл окажется нулевого размера, алгоритм посчитает, что файл еще не завершен и прервет отправку остальных файлов с соответствующим сообщением.

b. Задача отправки файла настраивается на один из нескольких возможных вариантов отправки с помощью дополнительных плагинов (см. ниже). В зависимости от того, включена опция сжатия или нет, исходный или сжатый файл отправляется по заданному алгоритму (в текущей версии это ftp или sftp).

c. После отправки, если был выбран вариант сжатия, архивный файл удаляется.

d. ТИсходный файл переименовывается путем добавления расширения `.fuploaded`.

3. Алгоритм переходит к отправке следующего файла из списка. Общее количество отправленных за итерацию файлов и их исходный (распакованный) размер суммируются для отображения в виджете.

4. По завершении отправки всех файлов из сформированного списка происходит поочередная очистка каталога, из которой файлы удаляются по маске `*.fuploaded`. То есть создается список всех таких файлов, он сортируется по времени последней модификации, а все старые файлы удаляются, кроме последних “Хранить NN файлы”.

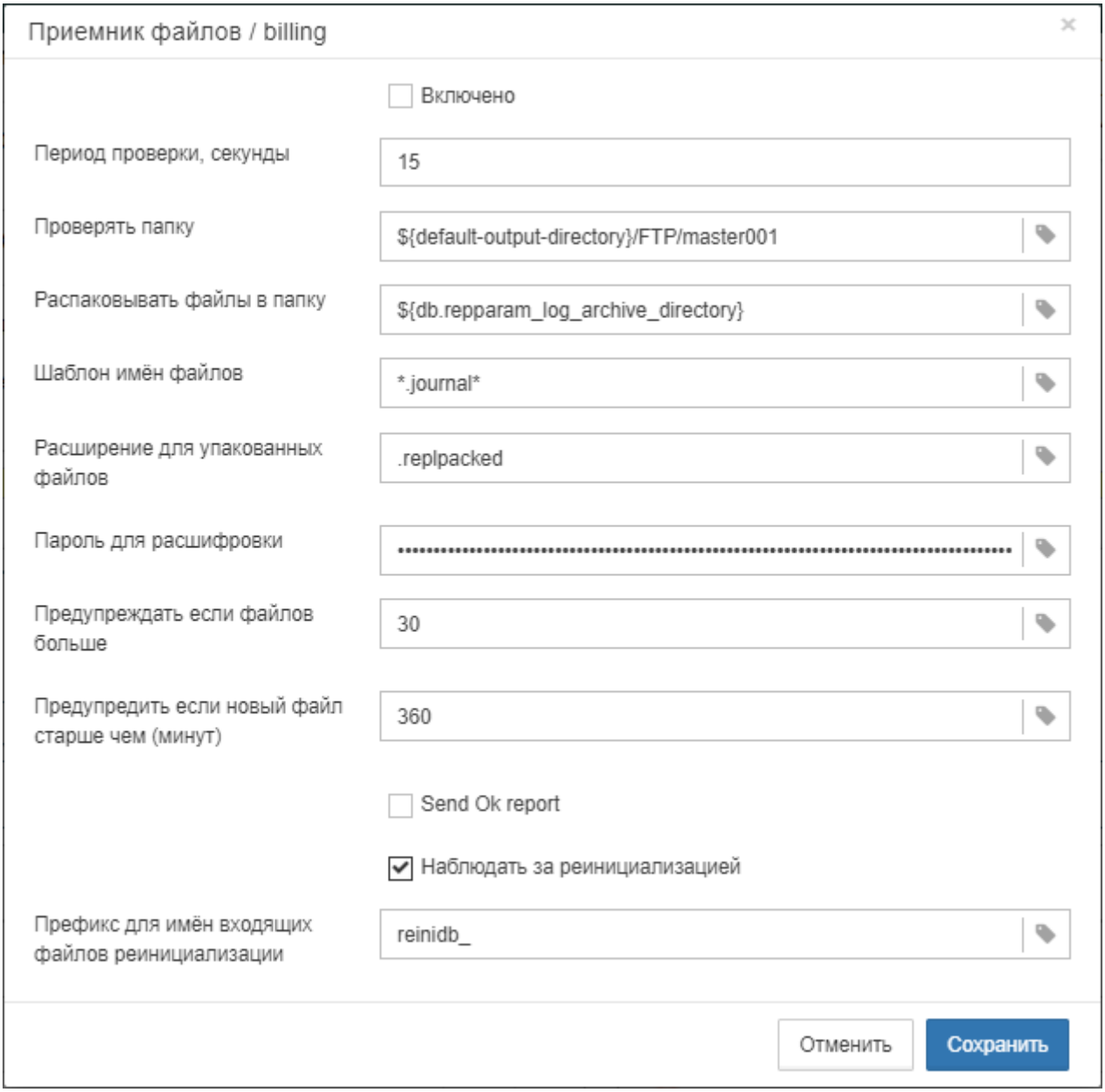
   По завершении отправки, если установлен флажок “Отправлять Ok уведомление”, то пользователю будет отправлен отчет о количестве и размере файлов, отправленных на текущей итерации.

### 3.4.13. База данных: Приемник файлов

В основном “Приемник файлов” предназначен для распаковки файлов из zip-архивов и чаще всего используется в паре с “Передача сегментов репликации” для передачи заархивированных сегментов репликации.

“Приемник файлов” проверяет файлы в папке, указанной в “**Проверять папку**”, с интервалом, равным “**Период проверки, секунды**”. Он проверяет только файлы с указанной маской в соответствии с “**Шаблон имён файлов**” (`.journal*` по умолчанию) и указанным расширением (`.replpacked` по умолчанию). Если он встречает такие файлы, то он распаковывает и расшифровывает их с помощью пароля, указанный в “**Пароль для расшифровки**”, и копирует в папку, указанную в “**Распаковывать файлы в папку**”.

Если параметр “**Наблюдать за реинициализацией**” включен, то “Приемник файлов” также будет отслеживать файлы предназначенные для реинициализации реплики, такие файлы имеют префикс указанный в “Префикс для имён входящих файлов реинициализации”.

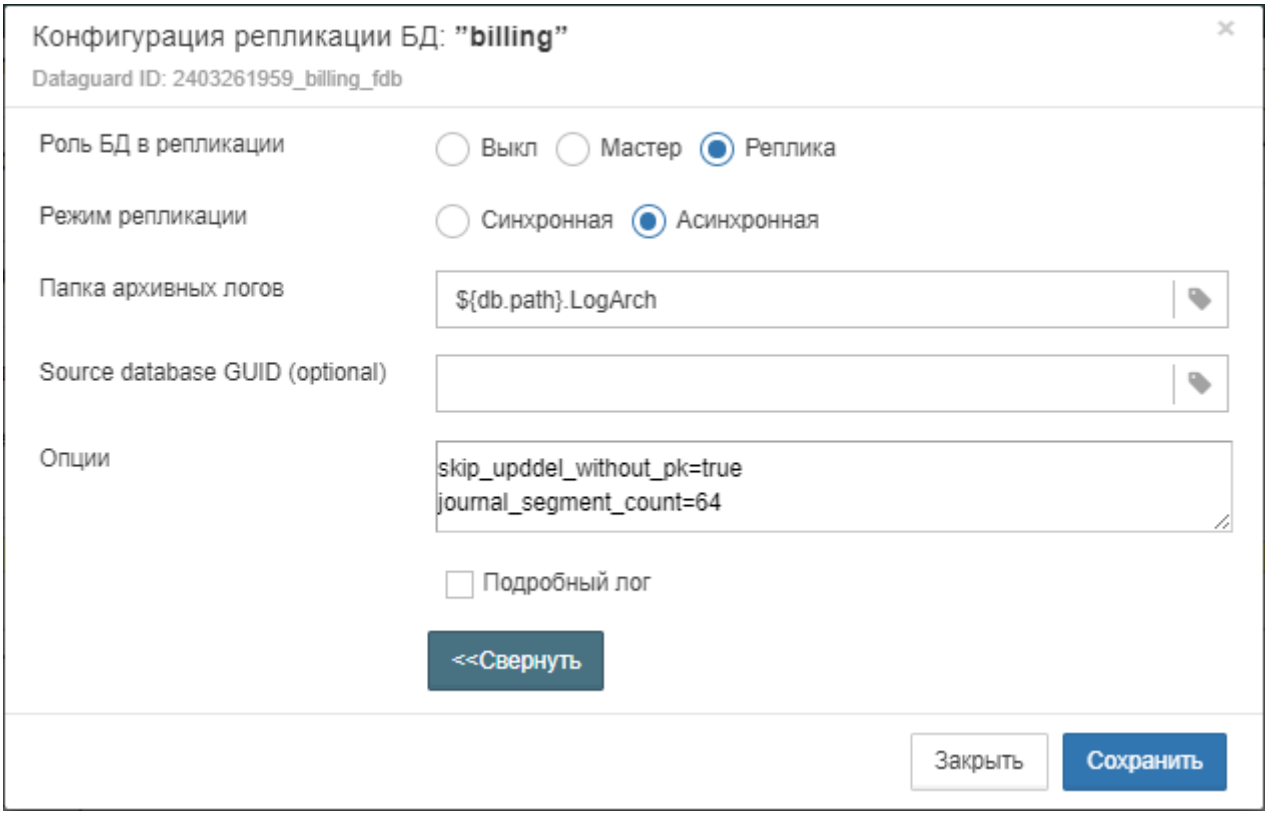



Имеются следующие дополнительные параметры:

- **Предупреждать если файлов больше** — по умолчанию 30. Если существует длинная очередь сегментов репликации, которые необходимо распаковать, то возможно это проблема с репликой базы данных, поэтому HQbird отправляет предупреждение для привлечения внимания администратора.
- **Предупредить если новый файл старше чем (минут)** — если самый последний файл (обычно сегмент репликации) слишком старый (более 360 минут), процесс репликации может быть прерван, и HQbird отправит соответствующее предупреждение.
- **Send Ok report** — о умолчанию отключено. Если он включен, HQbird отправляет электронное письмо о каждой успешной распаковке сегмента. Это может быть слишком часто для сегментов репликации, поскольку они поступают каждые 30–180 секунд, и это нормально для обычных файлов, таких как проверенные или инкрементные резервные копии.

После настройки “Приемник файлов” настройте реплику на поиск сегментов репликации: задайте в “Папка архивных логов” тот же путь, что и в “Приемник файлов” →

“Распаковывать файлы в папку”.




### Встроенный FTP-сервер

HQbird имеет встроенный FTP-сервер, который по умолчанию отключен. Для получения сегментов репликации удобно использовать встроенный FTP-сервер.

Чтобы включить встроенный FTP-сервер, необходимо отредактировать файл конфигурации `ftpsrv.properties`, который находится в папке `C:\HQbirdData\config` или `/opt/hqbird/ftpsrv.properties`

По умолчанию он содержит следующее:

```
#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=
```

Необходимо изменить `ftpsrv.enabled` на `true` и указать домашний каталог для FTP в параметре `ftpsrv.homedir`. Кроме того, рекомендуется использовать имя пользователя и

пароль, отличные от заданных по умолчанию.

После этого перезапустите службу FBDataGuard и проверьте доступность FTP.

**Внимание — пользователи Linux!**

В Linux служба FBDataGuard работает под пользователем **firebird**, поэтому домашний каталог FTP также должен иметь разрешения для пользователя **firebird**.

### 3.4.14. База данных: Сбор метаданных

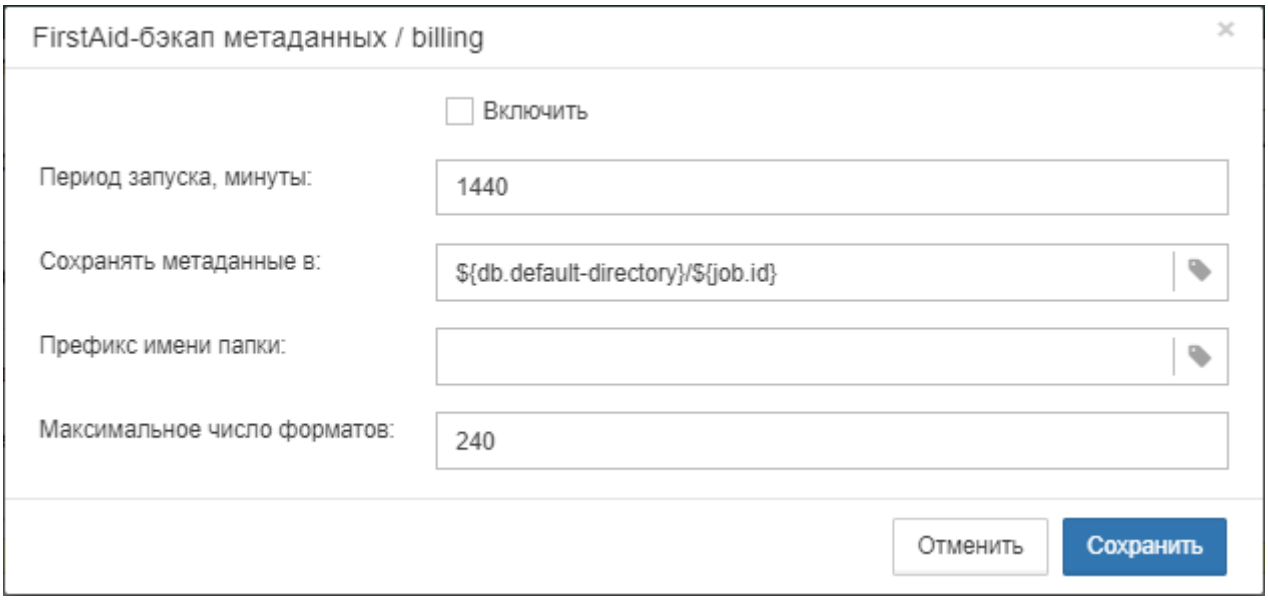
“База данных: Сбор метаданных” — одна из ключевых задач DataGuard, обеспечивающая защиту базы данных на низком уровне.

Прежде всего, это задание сохраняет “сырые” метаданные в специальном репозитории, поэтому в случае серьезного повреждения (например, из-за аппаратного сбоя) базы данных можно использовать этот репозиторий для восстановления базы данных.

Вторая цель этого задания — постоянная проверка всех важных системных таблиц на целостность. Каждые 20 минут оно просматривает все важные системные таблицы в базе данных и гарантирует отсутствие ошибок на уровне метаданных.

Третья цель — предупредить администратора о слишком большом количестве форматов для каждой таблицы.

В Firebird существует ограничение на количество форматов (256 на таблицу), однако даже несколько форматов могут значительно увеличить вероятность серьезного повреждения и замедлить производительность. Рекомендуется не изменять структуру таблиц в производственной базе данных и сохранять только один формат для каждой таблицы. Если это невозможно, администратору следует чаще выполнять резервное копирование/восстановление, чтобы преобразовать все форматы в один.



### 3.4.15. База данных: Валидация БД

Проверка базы данных Firebird требует монопольного доступа: т.е. во время проверки ни один пользователь не должен подключаться. Задание “База данных: Валидация БД” отключает (shutdown) базу данных и выполняет проверку базы данных, а затем включает (online) ее.

По умолчанию это задание ВЫКЛЮЧЕНО. Пожалуйста, внимательно подумайте, можно ли предоставить эксклюзивный доступ к базе данных. Проверка также может занять значительное время.

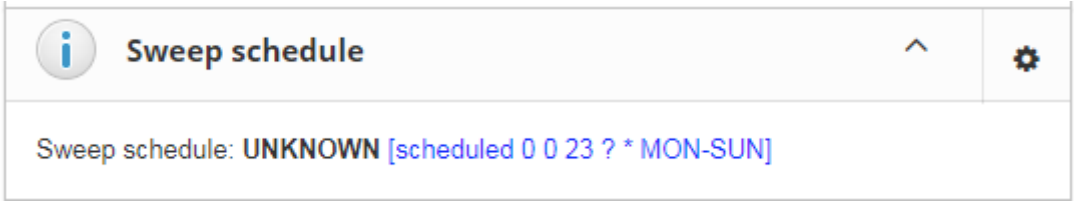

Используя диалоговое окно конфигурации, вы можете включить/отключить это задание, установить время выполнения, установить тайм-аут выключения (время ожидания перед запуском проверки), а также режим выключения (FORCE, ATTACH, TRANSNATIONAL). Если у вас нет глубоких знаний о том, что вы делаете, лучше оставить параметры по умолчанию.

Задание “База данных: Валидация БД” отправит предупреждение с критическим статусом, если возникнут какие-либо ошибки.

Кроме того, Firebird будет записывать ошибки в `firebird.log`, и они будут появляться в оповещениях, генерируемых заданием Сервер: Лог Firebird.

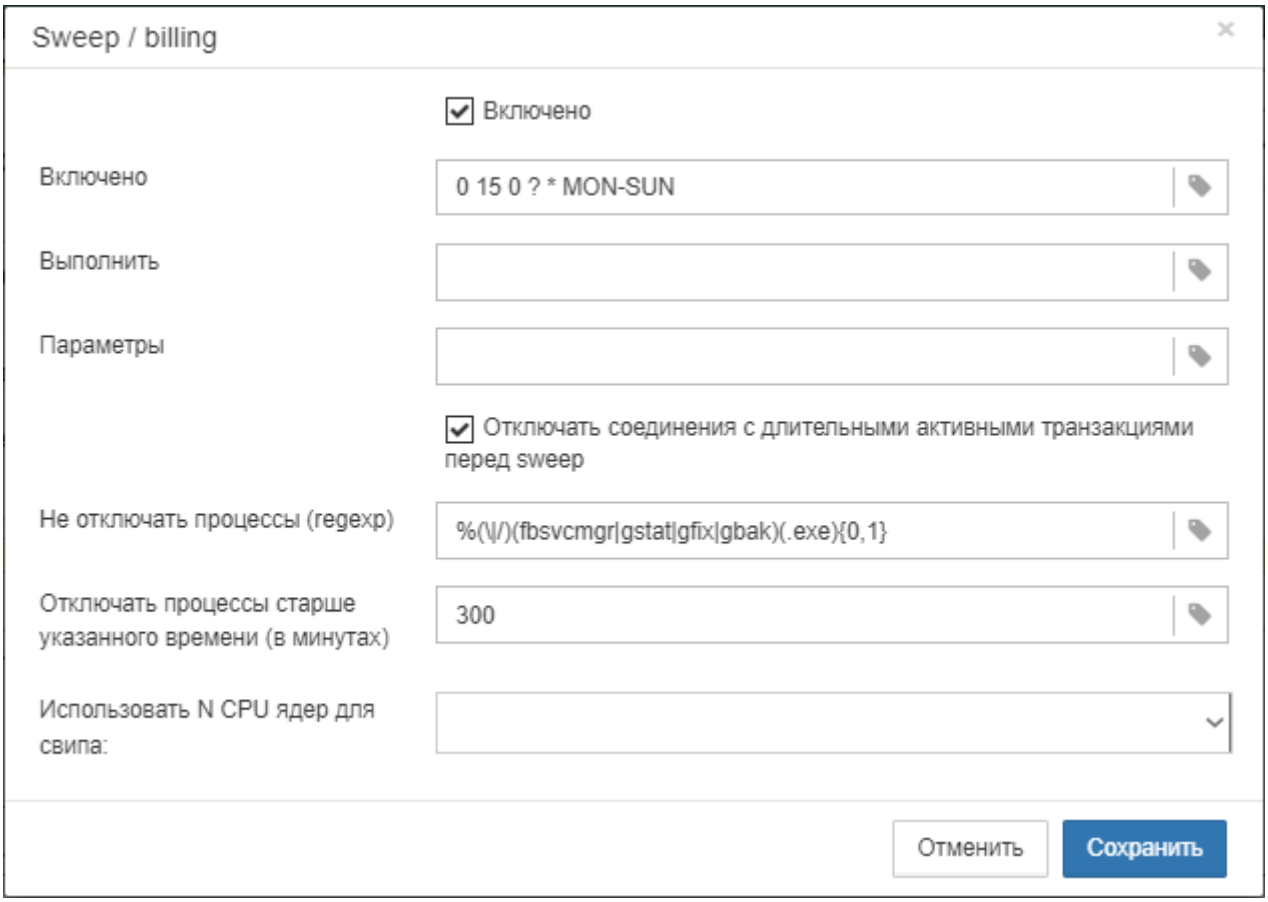
### 3.4.16. База данных: Sweep

FBDataGuard включает в себя специальное задание для явного запуска sweep в случае, если автоматический sweep отключен. По умолчанию задание отключено.




Рекомендуется запланировать явную запуск sweep с отключением длительных транзакций для всех баз данных, где такие транзакции обнаружены. Рекомендуемый период — один раз в день (обычно ночью, после завершения резервного копирования).

По умолчанию время запуска sweep установлено на 00-15, что может быть неподходящим временем, поскольку по умолчанию в это же время начинается резервное копирование, поэтому лучше измените его.




Обратите внимание: по умолчанию галочка “**Отключать соединения с длительными активными транзакциями перед sweep**” включена. Это означает, что HQbird обнаружит и отключит длительные транзакции (более 30 минут) перед запуском sweep, чтобы сделать sweep эффективным. Если долго выполняющиеся активные транзакции не будут отключены, sweep не сможет очистить старые версии записей.

“**Не отключать процессы (regexp)**” — в этом параметре укажите выражение `SIMILAR TO` для имен процессов, которые не будут отключаться. По умолчанию исключаются процессы `gfix`, `gbak`, `gstat` и `fbsvcmgr`.

“**Отключить процессы старше указанного времени (в минутах)**” — HQbird отключит процессы с длительно выполняемыми активными записываемыми транзакциями, пороговое значение по умолчанию составляет 300 минут. Практический верхний предел для этого параметра составляет 1440 минут (маловероятно, что транзакция будет делать что-то полезное более 1 дня).

“**Использовать N CPU ядер для sweep**” — HQbird Enterprise может использовать несколько ядер для выполнения sweep, чтобы ускорить sweep в 4-6 раз. Мы рекомендуем указывать не более половины ядер ЦП, если на сервере одна база данных, или 1/4 ядер ЦП, если баз данных несколько. Например, если у вас 16 ядер и 1 большая база данных, установите этот параметр равным 8, если несколько больших баз данных, то установите это значение равным 4.

### 3.4.17. База данных: Свободное место

Это задание отслеживает все объекты, связанные с базой данных: файлы базы данных (включая тома многотомной базы данных), дельта-файлы, файлы резервных копий и так далее.

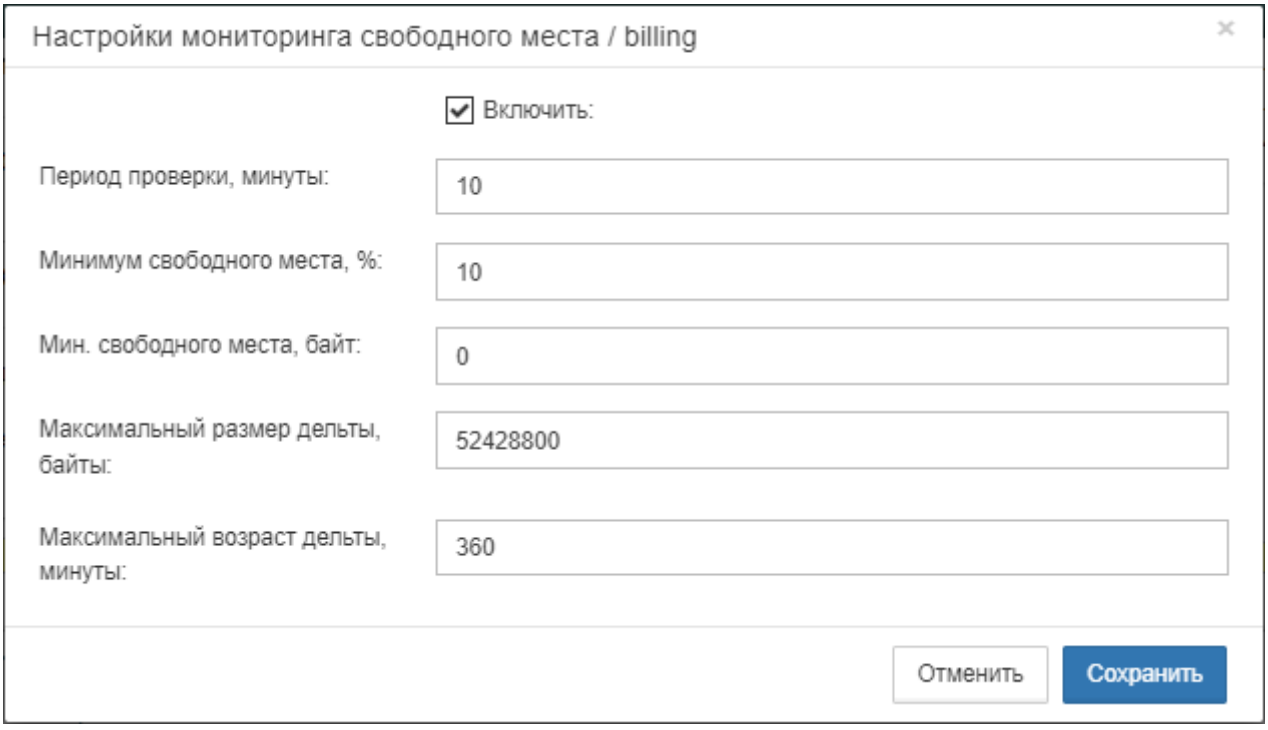
Задание “База данных: Свободное место” анализирует рост базы данных и оценивает, будет ли достаточно свободного места для следующей операции, такой как резервное копирование (включая тестовое восстановление), на конкретном жестком диске.

Оно генерирует несколько типов оповещений. Проблемы с дисковым пространством находятся в топе причин повреждения БД, поэтому обращайте внимание на оповещения по этому заданию.

Это задание также предоставляет данные для графика анализа свободного пространства сервера.

По умолчанию это задание включено.

Используя диалог конфигурации, вы можете указать период проверки и пороговые значения свободного места. Предупреждение будет выдано о первом достигнутом пороговом значении. Чтобы установить порог в % дискового пространства, вам необходимо установить явное пространство в байтах равным 0.




Если вы используете инкрементальное резервное копирование (или полное страничное резервное копирование), это задание критически важно. Он следит за временем жизни и размером дельта-файлов и предупреждает, если что-то пойдет не так. Забытые дельта-файлы часто являются причиной повреждений и значительных потерь данных.

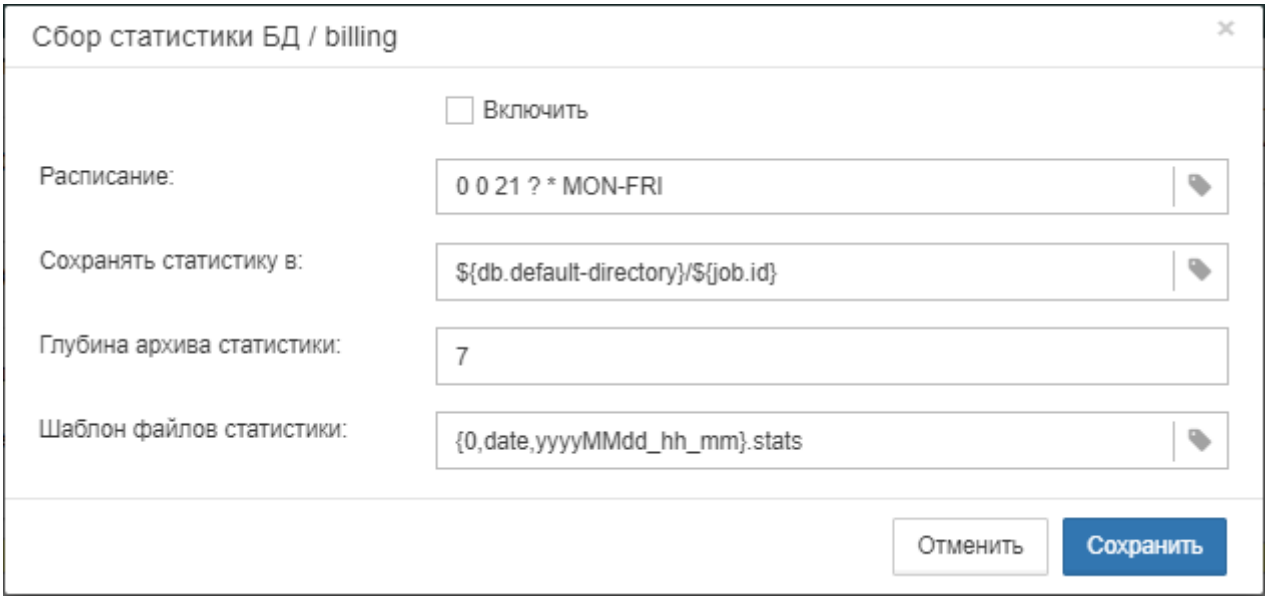
Это задание находит все дельта-файлы, связанные с базой данных, и проверяет их возраст и размер. Если один из этих параметров превышает пороговые значения “Максимальный размер дельты, байты” или “Максимальный возраст дельты, минуты”, администратор

получит предупреждение, и статус базы данных будет установлен на CRITICAL.

Если дельта-файл защищаемой базы данных был поврежден, то извлечь данные из него можно, используя метаданные из исходного файла базы данных или репозитория из задания База данных: Сбор метаданных.

### 3.4.18. База данных: Сбор статистики

Это задание очень полезно для выявления проблем с производительностью и выполнения общей проверки базы данных на низком уровне без резервного копирования.



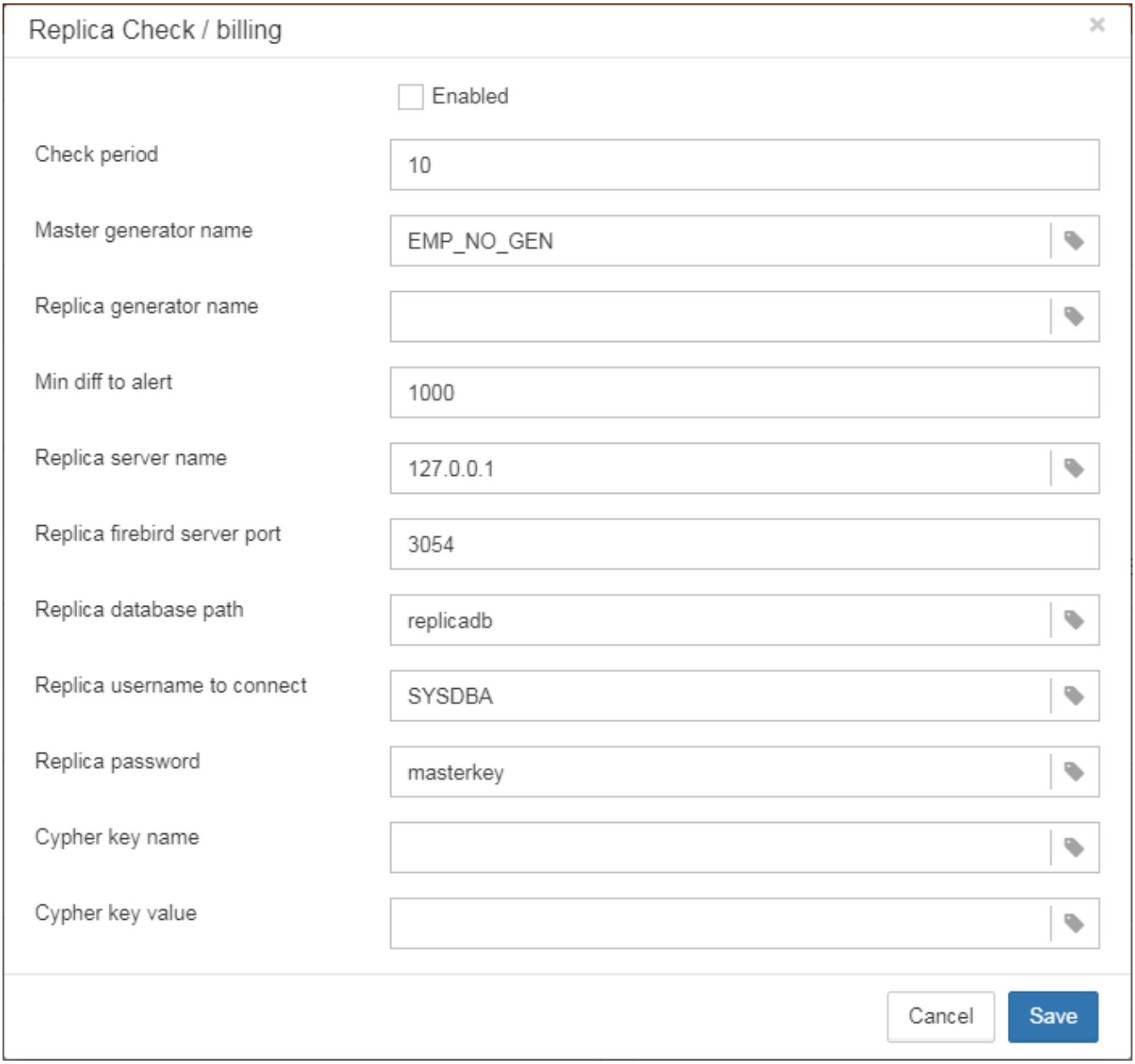

Мы рекомендуем запускать это задание каждый день и сохранять историю статистических отчетов.

Тогда с помощью HQbird Database IBAnalyst можно обнаружить проблемы с производительностью базы данных и получить полезные рекомендации по их устранению.

В качестве полезного побочного эффекта `gstat` посещает все страницы базы данных в поисках таблиц и индексов и проверяет их правильность.

### 3.4.19. База данных: Replica Check

Эта задача позволяет проверить доступность базы данных-реплики. По истечении заданного периода он меняет значение указанного генератора и сравнивает значение генератора на стороне реплики и главной базы данных.




**Min diff to alert** — разница между значениями генератора на стороне мастера и реплики, при превышении которого будут отсылаться оповещения.

## 3.4.20. База данных: Backup,Restore,Replace

Многие администраторы для решения проблем производительности используют следующий паттерн — бекап-рестор-замена. Обычно такая необходимость возникает, если приложение плохо управляет транзакциями, что приводит к накоплению мусора. При правильном проектировани приложения нет никакой необходимости постоянно выполнять резервное копирование и восстановление с заменой. Тем не менее HQBird предоставляет возможность автоматизировать этот процесс и запускать его как в ручном режиме (через Web-консоль), так и в автоматическом (по расписанию).

Паттерн бекап-рестор-замена состоит из следующих шагов:

1. выполнение команды копирования `gbak -b -g ⋯ database.fdb backup.fbk`
2. выполнение команды восстановления в новый файл БД `gbak.exe -c ⋯ backup.fbk`

`database_new.fdb`

3. переименование `database_new.fdb` в исходное имя базы данных

Однако в этом казалось бы простом процессе существует множество ловушек:

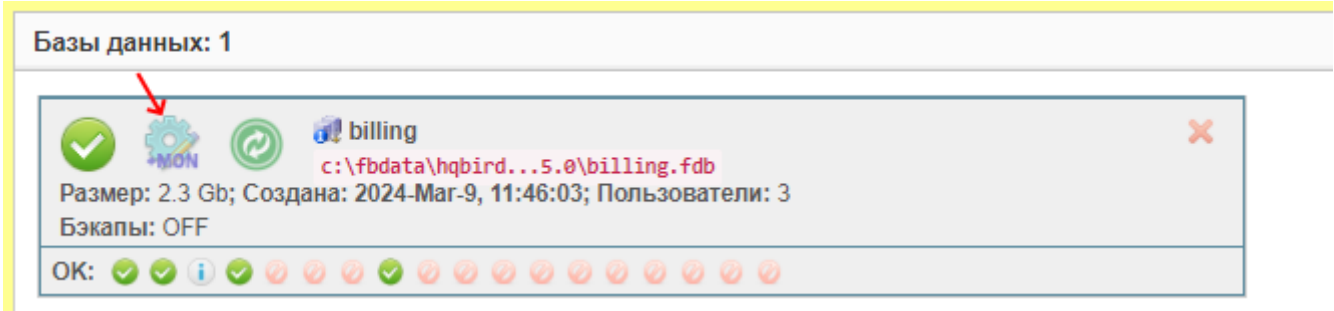
1. Недостаточно места на диске. Существует риск исчерпания дискового пространства во время операций резервного копирования или восстановления.
2. Неожиданные прерывания. Перезагрузка сервера, часто вызванная обновлениями Windows, может в любой момент нарушить выполнение задач резервного копирования или восстановления.
3. Повреждение исходной базы данных. Если исходная база данных повреждена, резервная копия может оказаться неполной.
4. Повреждение после восстановления. Различные проблемы, такие как нехватка места для сортировки больших индексов, могут привести к повреждению восстановленной базы данных.
5. Проблемы с производительностью. Процедуры резервного копирования и восстановления могут сильно замедляться из-за небрежного обслуживания, сбоев оборудования или параллельных задач, таких как полное резервное копирование виртуальных машин.
6. Контроль доступа. Крайне важно блокировать операции записи пользователя в исходную базу данных после начала процесса резервного копирования и восстановления, чтобы гарантировать, что изменения после резервного копирования не будут пропущены при восстановлении.

В случае возникновения любой из вышеперечисленных проблем необходимо отменить процесс резервного копирования-восстановления, то есть требуется откат к исходному состоянию базы данных.

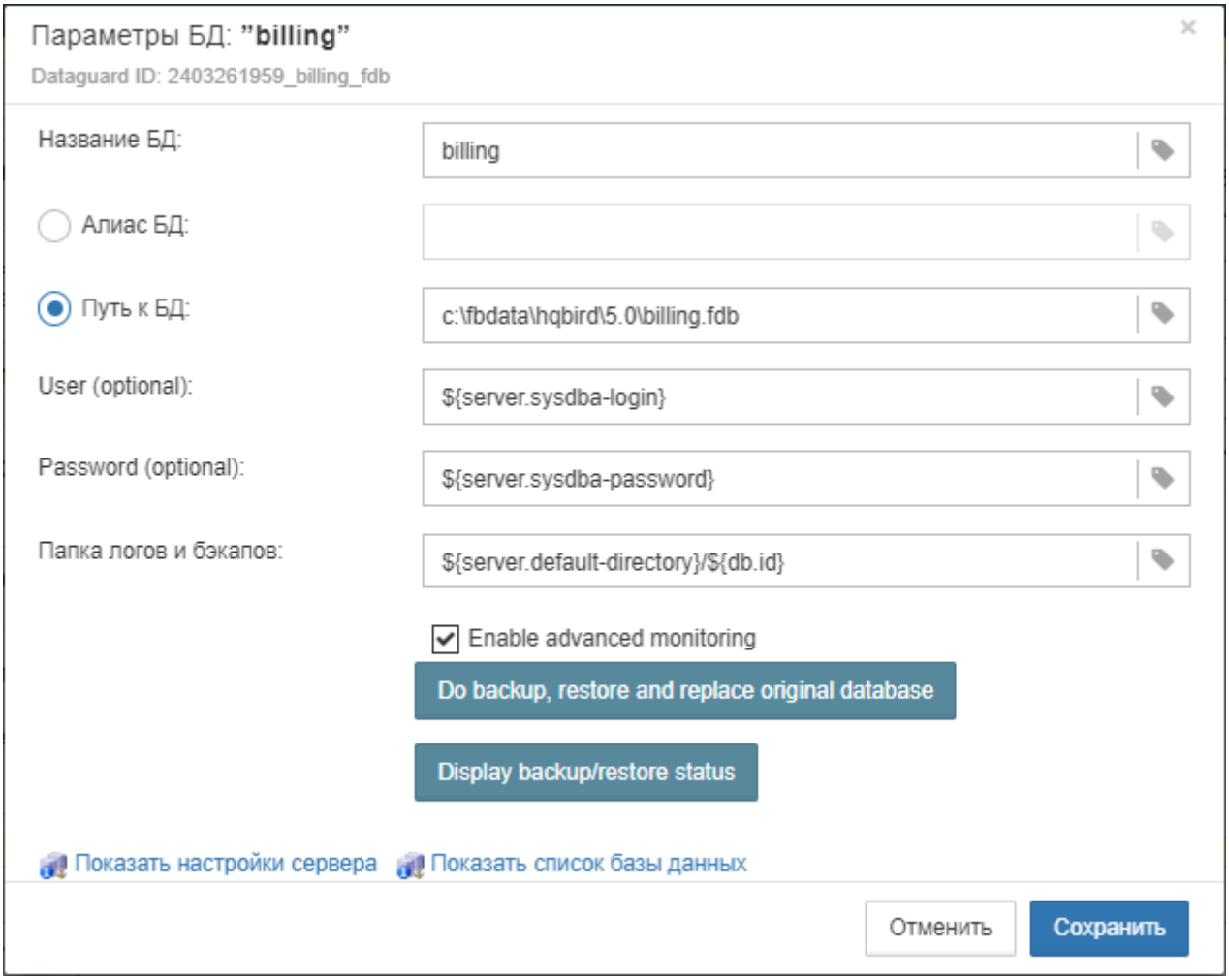
HQBird полностью автоматизирует процесс и позволяет избежать большинство проблем. Он контролирует дисковое пространство, отключат активных пользователей перед началом резервного копирования, следит за проиводительностью и целостностью резервной копии и восстановленной базы данных.

### Запуск Backup/Restore в один клик

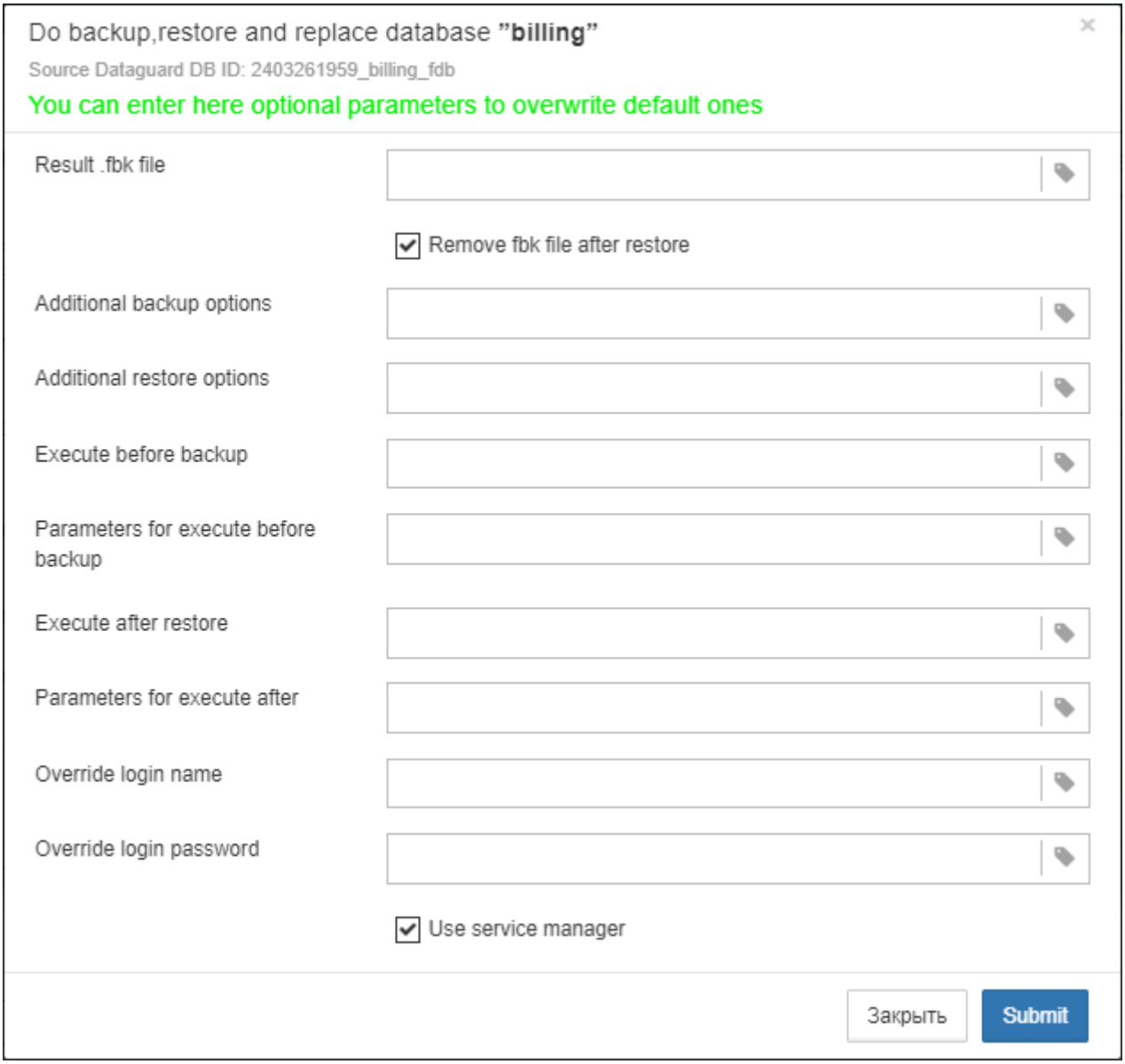
Чтобы сразу начать резервное копирование-восстановление, откройте диалог “Параметры БД” — нажмите на соответствующий значок:




После этого появится диалог “Параметры БД”:




Нажмите кнопку “Do backup, restore and replace original database”, откроется следующее диалоговое окно.




Все параметры являются необезательными:

- **Remove fbk file after restore** — по умолчанию эта опция включена, сохраните ее, если только вы не хотите сохранить промежуточный файл `fbk`.
- **Additional backup options** — дополнительные параметры резервного копирования, например, чтобы исключить некоторые таблицы из резервной копии.
- **Additional restore options** — дополнительные параметры восстановления, например, чтобы установить новый размер страницы.
- **Execute before backup** — позволяет указать исполняемый файл (файл `cmd`, sh-файл, `exe`), который будет выполняться непосредственно перед процессом резервного копирования.
- **Parameters for execute before backup** — позволяет указать параметры для исполняемого файла выше.
- **Execute after restore** — позволяет указать исполняемый файл (cmd-файл, sh-файл, exe), который будет выполняться сразу после процесса восстановления. Путь к восстановлению базы данных задан как первый параметр.
- **Parameters for execute after** — это аргументы или параметры, которые будут переданы

исполняемому файлу или сценарию, запуск которого запланирован после завершения процесса восстановления. Они помогают настроить поведение действий после восстановления в соответствии с конкретными потребностями среды или базы данных. Например, у вас может быть скрипт, которому необходимо знать путь к восстанавливаемой базе данных или установить определенные флаги для его работы.

- **Override login name** — позволяет использовать другого пользователя Firebird вместо имени пользователя, указанного в HQbird.
- **Override login password** – позволяет использовать другой пароль Firebird вместо пароля, указанного в HQbird.

Вы можете пропустить все эти параметры и просто нажать **[ Submit ]**, чтобы продолжить. Процесс начнется немедленно. HQbird выполнит все необходимые проверки и запустит процесс. В случае какой-либо ошибки процесс будет остановлен и исходная база данных будет возвращена обратно.

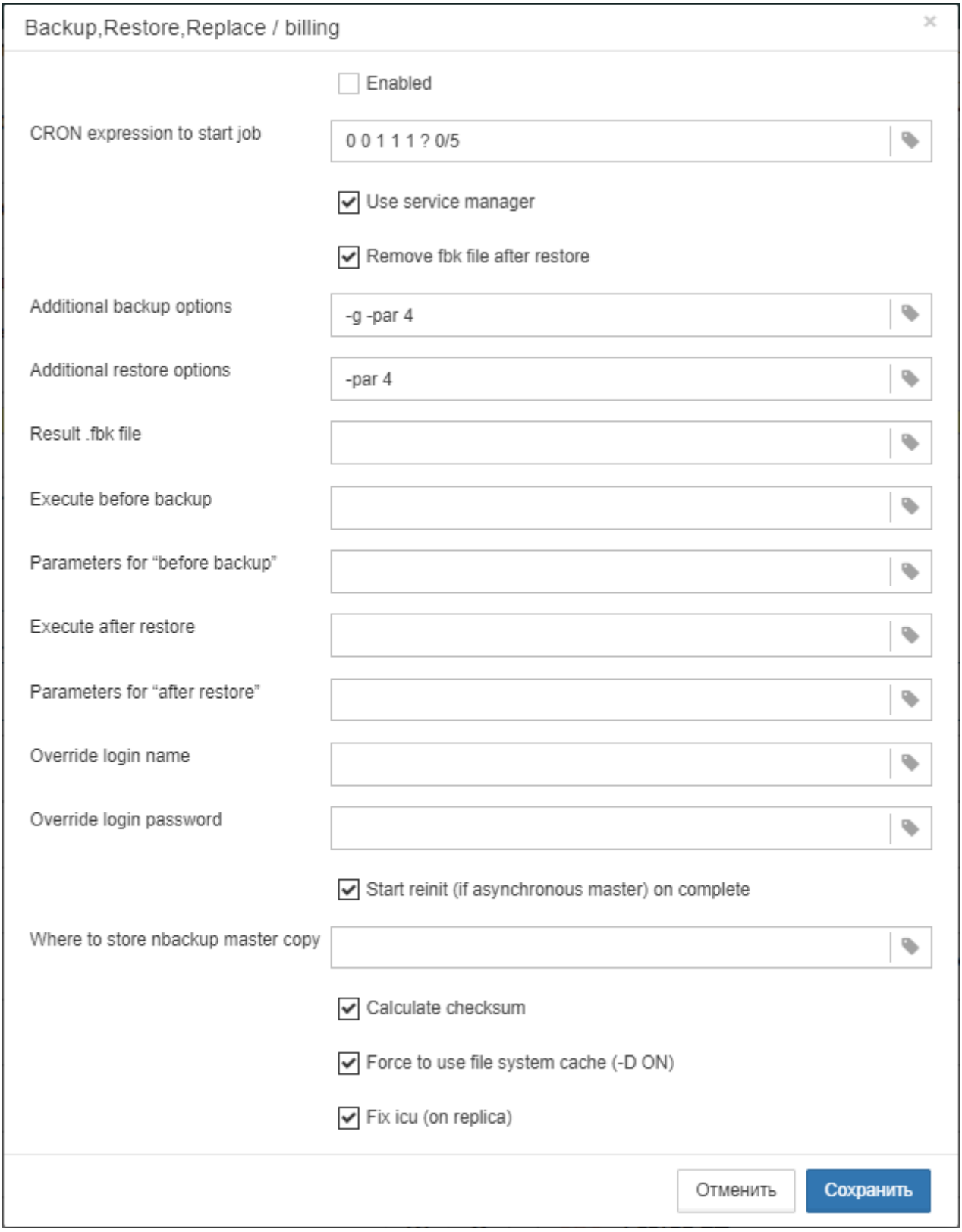
В результате HQbird сгенерирует следующий отчет:

```
gbakreplace: backup, restore and replace database is complete
task files are:
Original database file: "C:\HQbird\Firebird40\examples\empbuild\EMPLOYEE.FDB"
(2834432)
Replication operational log directory: ""
Replication archive log directory: ""
Temporary renamed original database file:
"C:\HQbird\Firebird40\examples\empbuild\EMPLOYEE.FDB.Apr-08--14-32-
06.ORIGINAL.FOR_MAINTAIN"
Resulted fbk file: "C:\HQbird\Firebird40\examples\empbuild\EMPLOYEE.FDB.Apr-08--14-32-
06.FBK" (80896)
Temporary restored database file:
"C:\HQbird\Firebird40\examples\empbuild\EMPLOYEE.FDB.Apr-08--14-32-06.restored"
(2834432)
backup log: "C:\HQbird\Firebird40\examples\empbuild\EMPLOYEE.FDB.Apr-08--14-32-
06.bkp.log"
restore log: "C:\HQbird\Firebird40\examples\empbuild\EMPLOYEE.FDB.Apr-08--14-32-
06.rst.log"
Stage report file: "C:\HQbird\Firebird40\examples\empbuild\EMPLOYEE.FDB.Apr-08--14-32-
06.hqbirdgbakreplace.report"
```

### Задача Backup,Restore,Replace

Кроме того, можно запланировать резервное копирование и восстановление в любое желаемое время и, если это необходимо выполнить инициализацию репликации сразу после этого. Чтобы запланировать его, нажмите кнопку “Настройка” в виджете **Backup, Restore, Replace**.


Появится следующий диалог:




Как видите, этот диалог пости такой же как диалог немедленного резевного копирования и

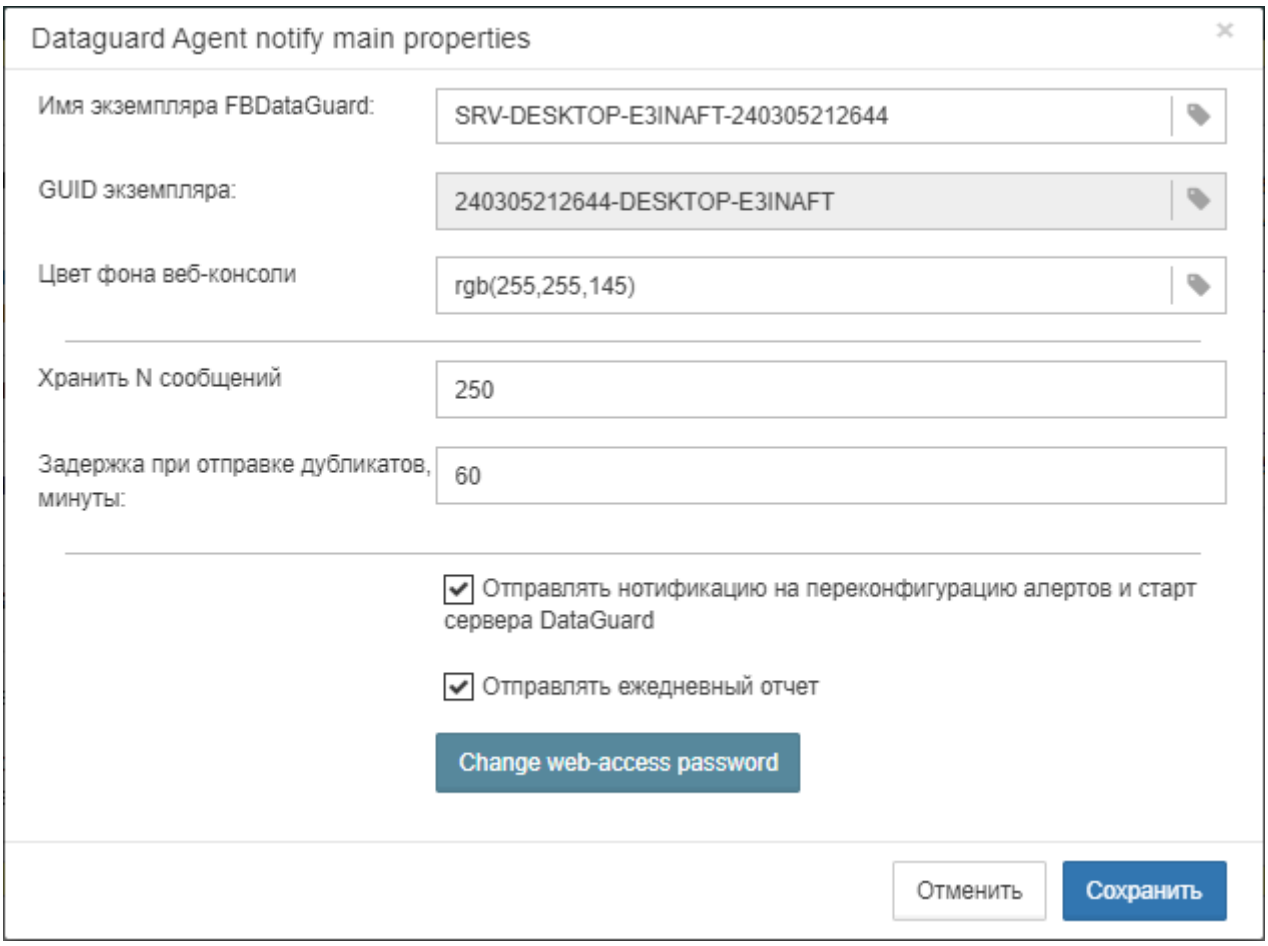
востановления. Тем не менее, он дополнительно включает в себя шаги для автоматической повторной реинициализации реплик после восстановления базы данных. Это важно из-за изменения GUID базе данных после восстановления. При активной репликации HQbird будет беспрепятственно повторно реинициализировать реплику после успешного восстановления.

# 3.5. Оповещения по электронной почте в HQbird FBDataGuard

FBDataGuard может отправлять оповещения по электронной почте администратору(ам): такие оповещения содержат информацию об успешном резервном копировании, а также потенциальных и реальных проблемах с базами данных.

Общие свойства уведомлений можно настроить, нажав на имя сервера (или имя компьютера) вверху веб-консоли:


После этого вы увидите диалог общих настроек оповещений:




Описания некоторых свойств, которые вы можете установить здесь:

- “**Имя экземпляра FBDataGuard**” — какое-нибудь читаемое имя для вашего удобства; оно будет упоминаться в электронных письмах и оповещениях.
- “**GUID экземпляра**” — служебное поле; нет необходимости менять его.
- “**Цвет фона веб-консоли**” — часто полезно настроить цвет веб-интерфейса HQbird.

Рекомендуется включить настройку оповещений по электронной почте. Для этого вам нужно нажать на кнопку конверта вверху веб-консоли:


После этого вы увидите диалог настройки оповещений:

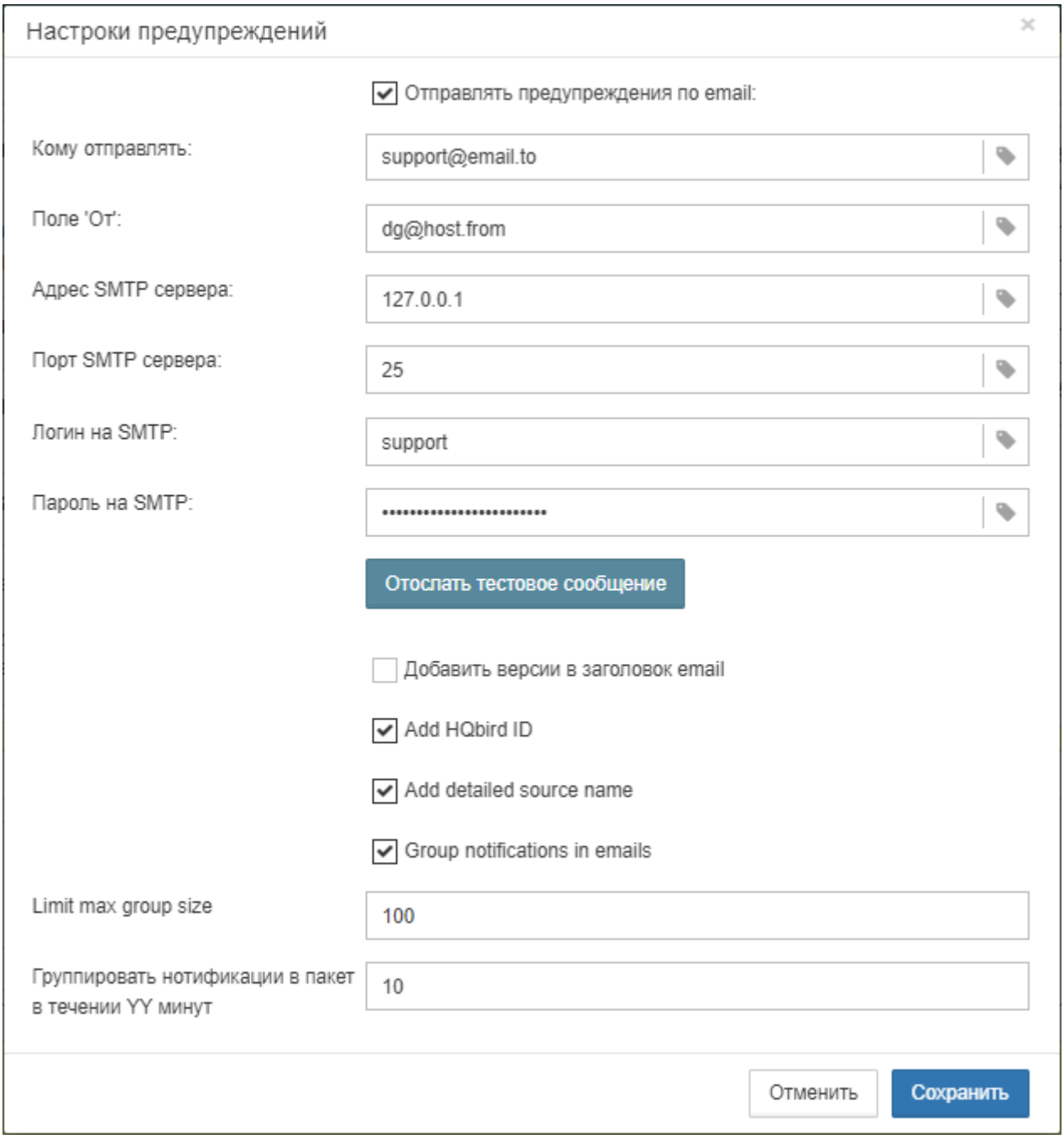



*Рисунок 38. Диалог настройки оповещений по электронной почте в FBDataGuard.*

Прежде всего, вам необходимо включить отправку оповещений, установив флажок “Отправлять предупреждения по email”.

- “**Отправлять предупреждения по email**” — включает оповещения по электронной почте и отображает настройки параметров электронной почты ниже.
- “**Кому отправлять**” — указывает куда отправлять электронные письма.
- “**Поле 'От'**” — это то, что будет установлено в качестве отправителя в электронном письме.

- “Адрес SMTP сервера”, “Порт SMTP сервера”, “Логин на SMTP” и “Пароль на SMTP” — это данные, которые будут использоваться для отправки электронных писем.

Перед сохранением настроек вы можете нажать кнопку “Отослать тестовое сообщение”, если настройки верны, вам должно прийти письмо на указанный адрес.

Чтобы ограничить количество писем, вы можете собирать сообщения в группы и отправлять их пакетами. Для этого установите флажок “Group notifications in emails”. Это также поможет обойти некоторые анти спам-системы, которые могут внести вас в черный список из-за слишком частой отправки электронных писем.

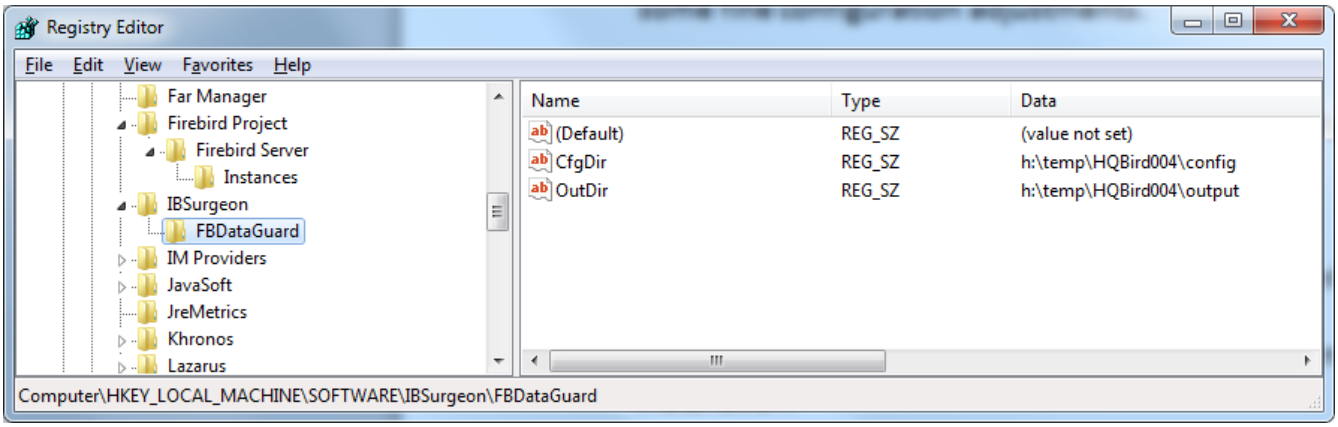
Нажмите “Сохранить”, чтобы сохранить настройки оповещений по электронной почте.

# 3.6. Советы и рекомендации FBDataGuard

FBDataGuard позволяет менять его настройки не только через веб-консоль, но и путем прямого изменения файлов конфигурации. Это может быть полезно, когда вам нужно установить FBDataGuard в автоматическом режиме (без взаимодействия с пользователем), связать его со сторонним программным обеспечением или выполнить некоторые тонкие настройки конфигурации.

## 3.6.1. Путь к конфигурации FBDataGuard

При запуске FBDataGuard ищет в реестре пути конфигурации и вывода:




Эти значения указывают пути к конфигурации FBDataGuard и выходной папке. Они выбираются во время установки.

## 3.6.2. Настройка порта веб-консоли

Один из наиболее часто задаваемых вопросов — как настроить порт для приложения веб-консоли (по умолчанию это 8082). Это можно сделать, изменив настройку порта в файле `%config%\agent\agent.properties` (`%config %` — это `C:\HQbirdData\config` или `/opt/hqbird/conf`).

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

`%config%` — папка для хранения информации о конфигурации, в которой она указана.

## 3.6.3. Как изменить пароль администратора

Вы можете указать этот пароль в файле `access.properties` (в `C:\HQbirdData\config` или `/opt/hqbird/conf`)

```
access.login=admin

access.password=youradminpasswordforhqbird
```

После установки пароля перезапустите FBDataGuard, и новый пароль будет зашифрован и применен.

### 3.6.4. Гостевой пользователь HQbird FBDataGuard

Для доступа к HQbird FBDataGuard существует пользователь только для чтения с именем `guest`.

```
access.guest-login=guest

access.guest-password=yournewpassword
```

# Глава 4. Настройка репликации HQBird

HQbird — это расширенный дистрибутив Firebird для больших баз данных с инструментами мониторинга, оптимизации и администрирования, он также включает в себя плагин для встроенной репликации master-slave и различные улучшения производительности.

В HQbird встроенная репликация доступна начиная с версии 2.5. В “ванильном” Firebird она появилась позже и доступна начиная с версии 4.0.

HQbird на 100% совместим с Firebird 2.5, 3.0, 4.0 и 5.0 — никаких изменений в ODS не требуется. Чтобы переключиться на HQbird и обратно, резервное копирование/восстановление не требуется, просто остановите/запустите Firebird и замените двоичные файлы. Репликация возможна между узлами с одинаковой версией, т.е. невозможна между версиями 3.0 и 2.5.

## 4.1. Как работает репликация

Репликация HQbird работает на логическом уровне: она реплицирует изменения сделанные DML (`INSERT`/`UPDATE`/`DELETE`, `EXECUTE PROCEDURE` и др.) и DDL (`CREATE`/`ALTER`/`DROP`) операторами; **никаких дополнительных триггеров не требуется**. Единственное требование к текущей версии репликации — наличие уникальных или первичных ключей для всех таблиц, изменения в которых необходимо реплицировать.

Чтобы использовать репликацию, вам необходимо установить HQbird, зарегистрировать его (с пробной версией или с полной лицензией) и настроить репликацию. HQbird должен быть запущен на главном сервере и на всех серверах-репликах.

Ниже мы рассмотрим, как настроить репликацию Firebird с помощью HQbird.

*Вы можете использовать HQbird в Windows и Linux с 32-разрядной и 64-разрядной версиями Firebird 2.5, 3.0, 4.0 и 5.0.*

## 4.2. Установка

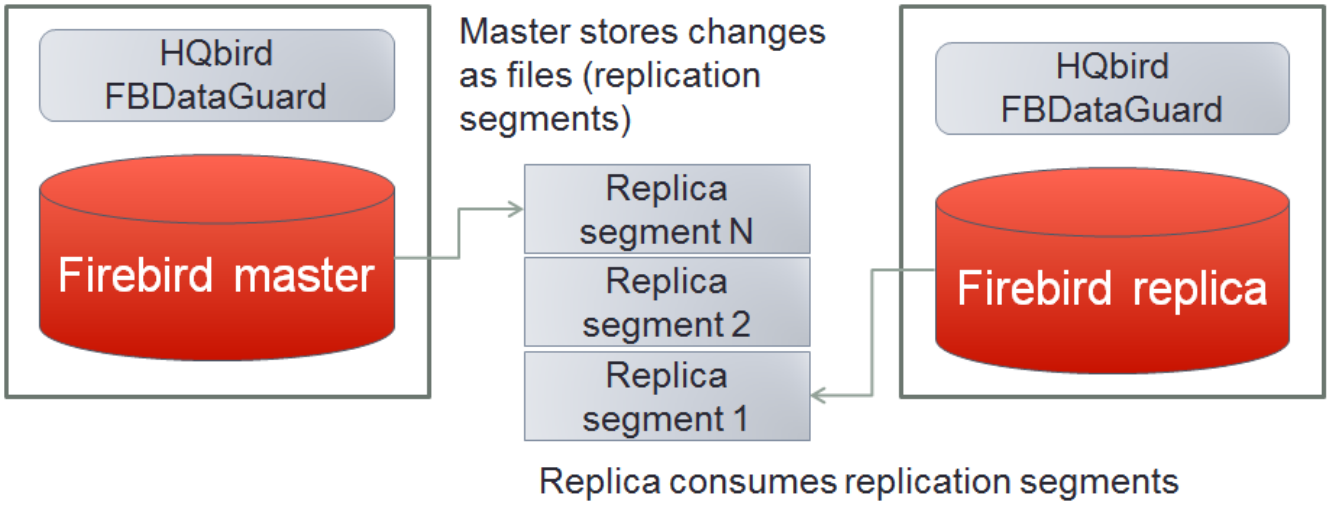
Пожалуйста, установите HQbird из прилагаемого дистрибутива. Если у вас установлена другая версия Firebird (2.5, 3.0), сначала удалите ее.

Чтобы включить репликацию, на вашем главном сервере и сервере-реплике должна быть рабочая и зарегистрированная (пробная или полная) копия HQbird.

Пожалуйста, обратитесь к разделу 2 этого руководства для получения подробной информации об установке сервера HQbird.

# 4.3. Асинхронная репликация в Firebird

HQbird поддерживает 2 типа репликации: асинхронную и синхронную. В случае асинхронной репликации главный сервер сохраняет подтверждённые изменения из главной базы данных в файлах (сегментах репликации), которые могут использоваться асинхронно одним или несколькими серверами-репликами.




Как работает асинхронная репликация:

- Изменения на стороне мастер-севера регистрируются в файлах журнала репликации
- Журнал состоит из нескольких сегментов (файлов)
- Сегменты репликации (заархивированные) передаются на сервера-реплики и применяются к репликам в фоновом режиме
- Реплику можно создавать и пересоздавать онлайн (без остановки мастера)

Важные моменты, которые следует учитывать:

- Практическая задержка между мастером и репликой настраивается и может составлять 15-30 секунд (по умолчанию — 90 секунд).
- Задержка между мастером и репликой может увеличиться в случае большой нагрузки (из-за задержки обработки сегментов репликации)
- Реплику можно переключить в основной (т.е. обычный) режим с помощью 1 команды

**Асинхронная репликация — рекомендуемый вариант для HQbird**:

- обеспечивает стабильность и защиту от повреждений базы данных Firebird;
- может быть быстро и легко сконфигурирована;
- не требует простоев для настройки;
- имеет возможность повторной инициализации в онлайн режиме;
- подходит для распределенных сред (когда реплика находится в облаке или на удаленном сервере).

Для настройки асинхронной репликации потребуются следующие шаги:

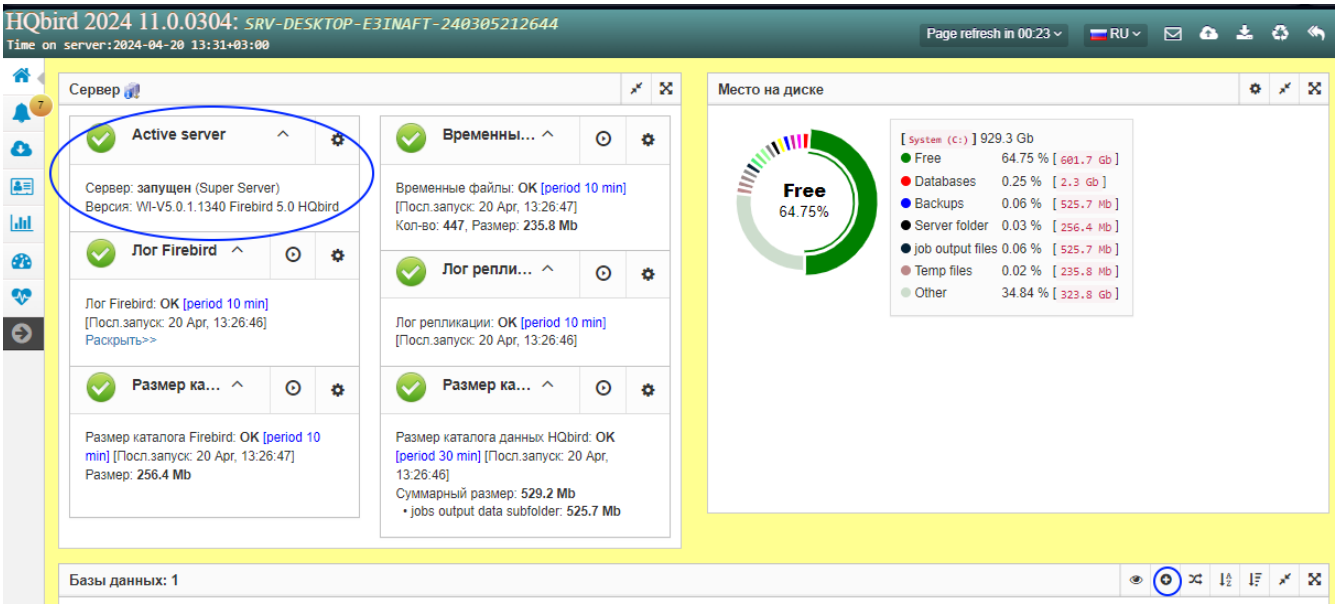
1. Настройка HQbird для репликации на главном сервере
2. Создание копии файла базы данных master
3. Настройка базы данных для репликации на сервере-реплике (подчиненном сервере)

### 4.3.1. Шаг 1: Настройка HQbird для репликации на главном сервере

Чтобы настроить репликацию, откройте HQbird FBDataGuard: запустите современный браузер (Chrome, Firefox и т.д.) и откройте этот локальный URL: http://127.0.0.1:8082 (порт, если необходимо настраивается в ini-файлах HQbird).

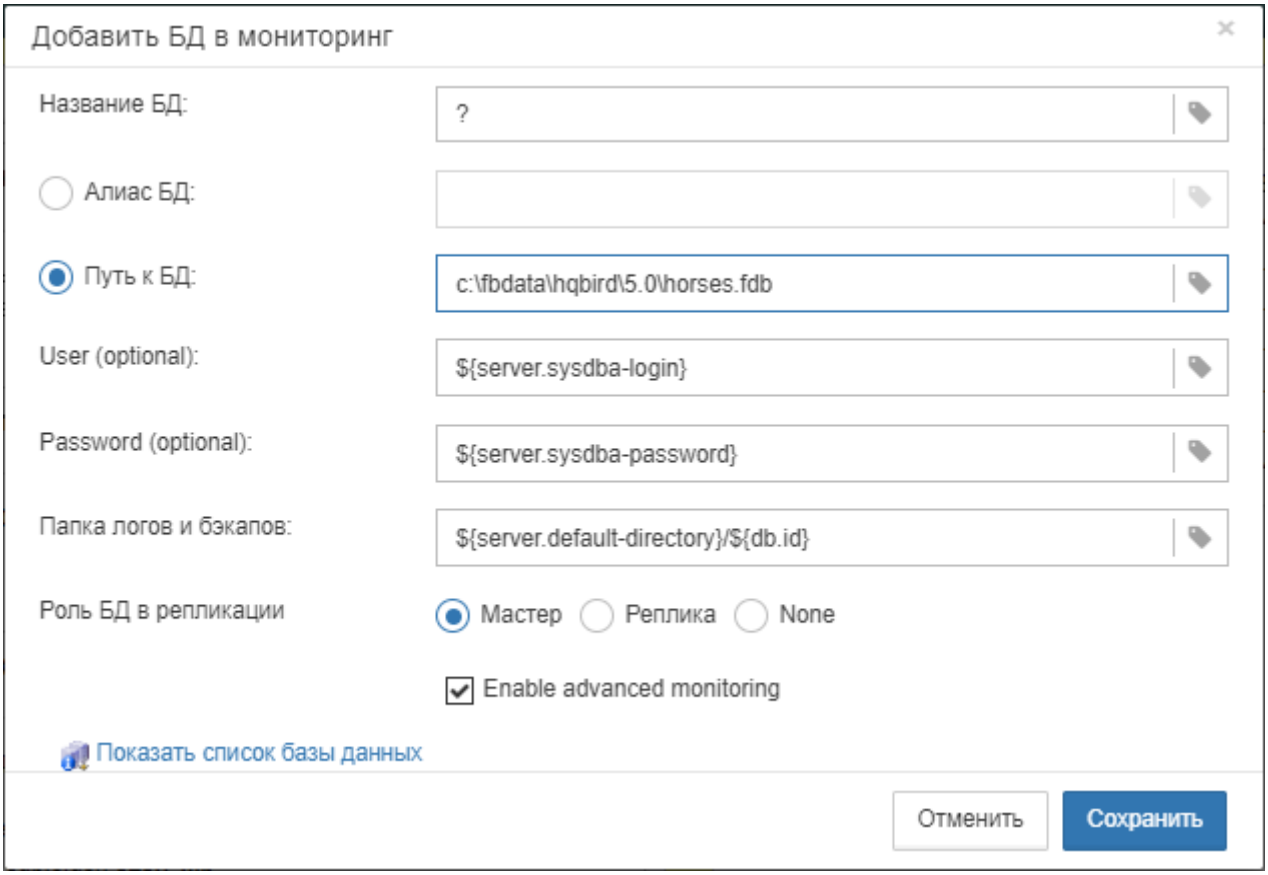
Введите им и пароль по умолчанию: **admin/strong password**.

Зарегистрируйте сервер Firebird, и появится следующая картинка:



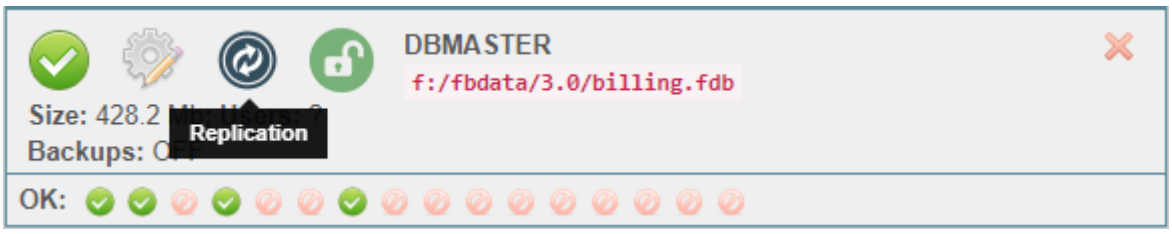

Убедитесь, что вы действительно подключены к правильной версии Firebird — в левом верхнем углу в виджете "Active server" должна быть указана версия “… Firebird 2.5 **HQbird**” или “… Firebird 3.0 **HQbird**” или “… Firebird 4.0 **HQbird**” или “… Firebird 5.0 **HQbird**”.

После этого нажмите “Add database” в правом нижнем углу и укажите ник и путь к базе данных, которая будет основной:




Пожалуйста, обратите внимание, что база данных должна быть зарегистрирована с указанным в ней явным путем, а не с псевдонимом — репликация не будет работать с псевдонимом.

После успешной регистрации базы данных нажмите на значок в заголовке базы данных, чтобы настроить репликацию:




После этого появится основное диалоговое окно настройки для баз данных master и replica.

Когда репликация не настроена, это диалоговое окно практически пустое:

### Асинхронная репликация на главном сервере

Асинхронная репликация записывает все изменения в главной базе данных в журнал репликации: набор файлов, называемых "сегментами репликации". Сервер реплики получает эти сегменты и применяет их к базе данных реплики.

Ранее мы зарегистрировали "H:\dbwmaster.fdb`, в данном примере это главная база данных. Чтобы настроить асинхронную репликацию на стороне мастера: выберите роль репликации: “Master”, затем “Asynchronous” и нажмите “Save”.

Начиная с Firebird 4.0, вам необходимо включить публикацию на уровне базы данных. Нажав на кнопку "Enable publications (and grant it for all tables)", вы активируете публикацию и добавляете все таблицы в список для публикации. Списком таблиц для публикации можно управлять в SQL, используя следующие инструкции:

```
ALTER DATABASE INCLUDE {TABLE <table_list> | ALL} TO PUBLICATION

ALTER DATABASE EXCLUDE {TABLE <table_list> | ALL} FROM PUBLICATION

<table_list> ::= tablename [, tablename ...]
```

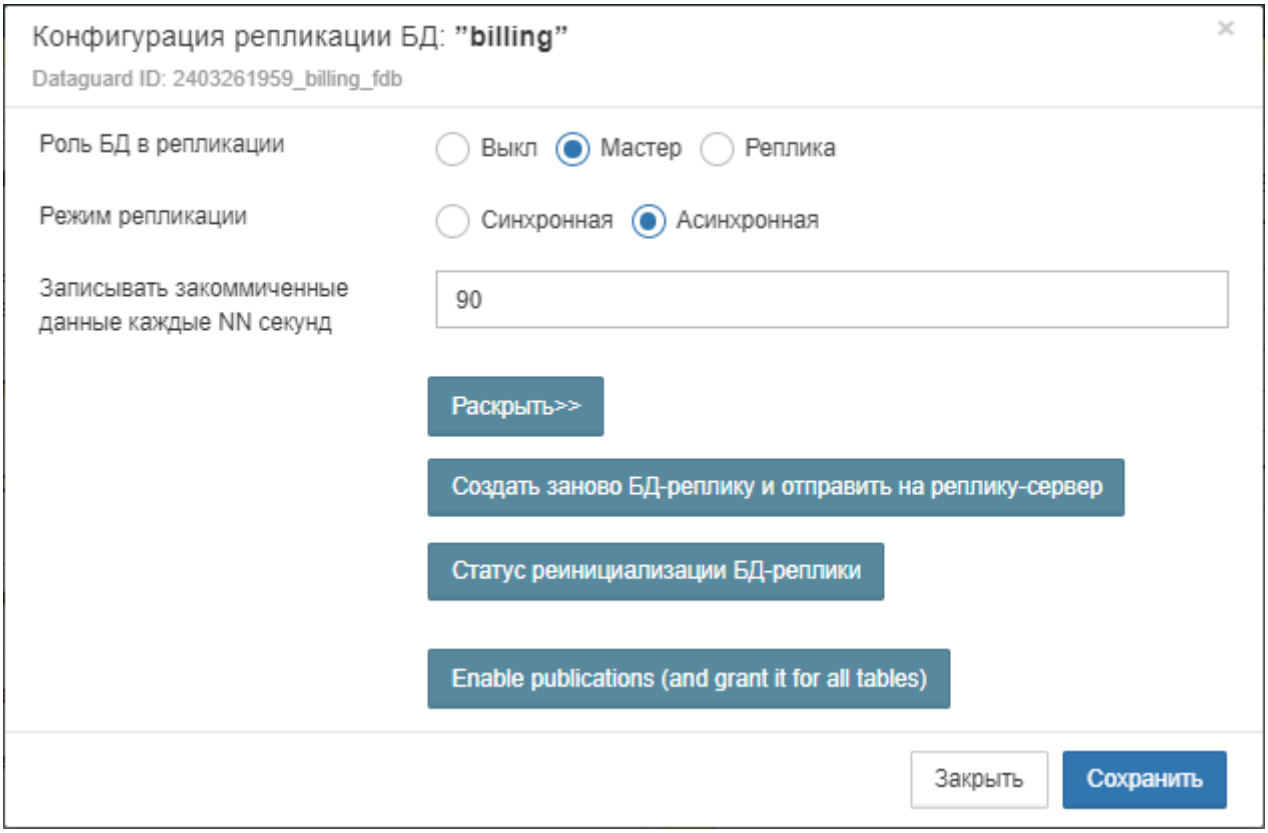
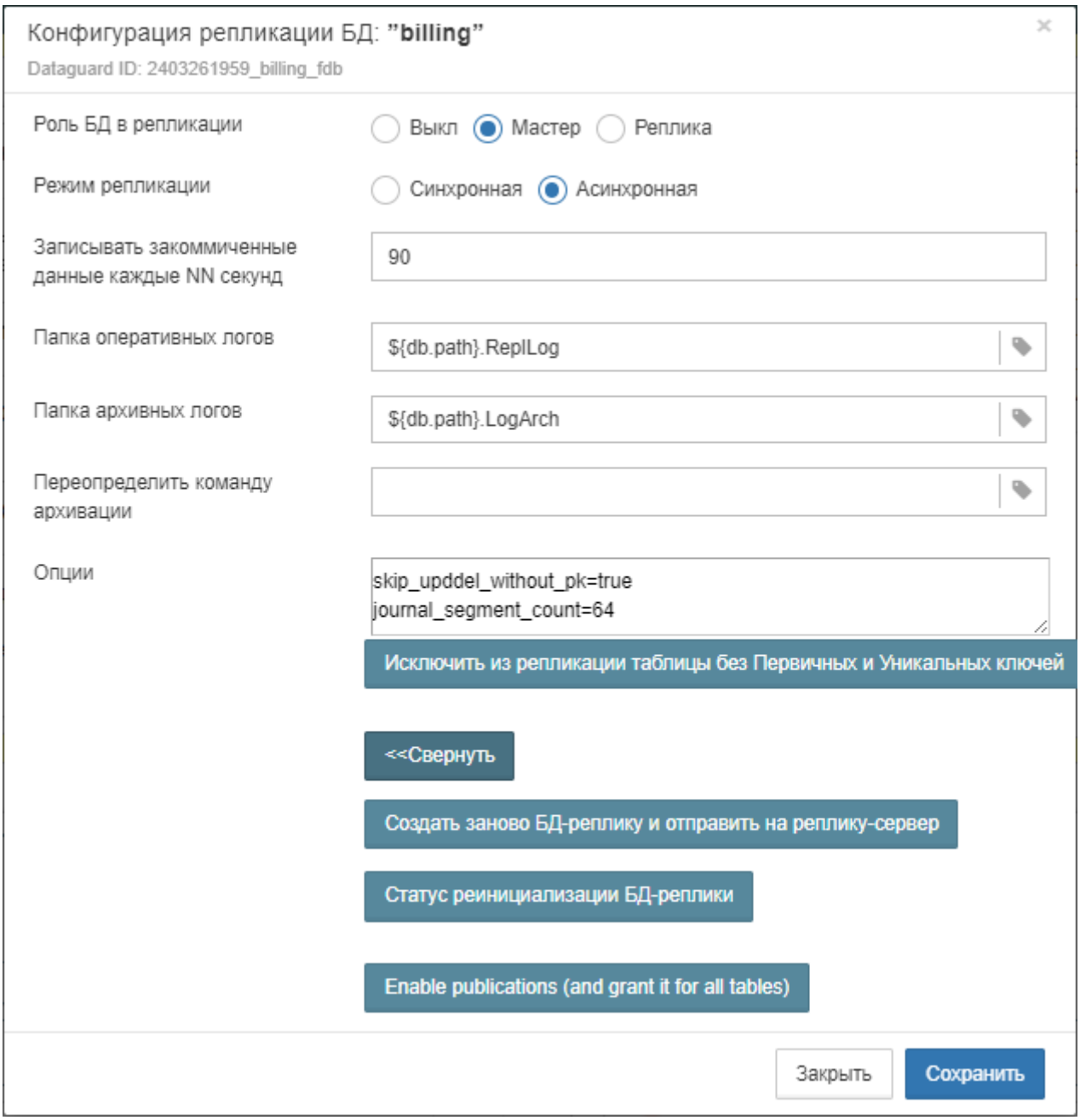



*Рисунок 39. Диалог настройки репликации для асинхронной репликации*

Единственный параметр, который вы можете изменить, — это “**Записывать закоммиченные данные каждые NN секунд**”, он определяет, как часто должны перемещаться зафиксированные данные в архивные сегменты репликации.

По умолчанию это значение равно 90 секундам.

Есть несколько необязательных параметров, которые вы можете изменить, если откроете подробное диалоговое окно с кнопкой **[ more>> ]**.:



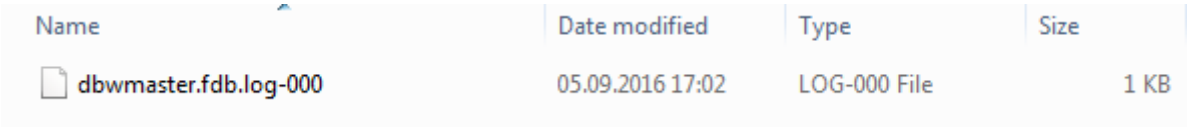

Давайте рассмотрим все параметры в этом диалоговом окне — просто чтобы дать вам представление о том, что они делают, **их не нужно изменять**:

- “Папка оперативных логов” — папка, в которой будут храниться операционные журналы. Это системная папка, полностью управляемая Firebird. По умолчанию `${db.path}.ReplLog` (`db.path` — это место, где находится база данных). **Нет необходимости изменять значение по умолчанию**.
- “Папка архивных логов” — папка, в которой будут храниться архивированные журналы. В соответствии со значением по умолчанию `${db.path}.LogArch`, HQbird создаст папку `DatabaseName.LogArch` в папке с базой данных, поэтому **нет необходимости изменять этот параметр**.
- Третий параметр (“Переопределить команду архивации”) необязательный, **оставьте его пустым**.

Пожалуйста, обратите внимание, что параметры репликации инициализируются при первом подключении к базе данных. Вот почему вам необходимо перезапустить службу Firebird (или все подключения в случае Classic) после настройки репликации — такой перезапуск гарантирует, что репликация запустится должным образом.

В этом случае сегменты журнала репликации сначала будут записаны в `${db.path}.ReplLog` (`db.path` — это место, где находится база данных — в нашем примере это будет `H:\DBWMaster.fdb.ReplLog`), а после достижения максимального размера сегмента, или количество коммитов, или после другого триггера, будет запущена команда архивирования по умолчанию — она скопирует заархивированные сегменты репликации в `${db.path}.LogArch` (в нашем примере это будет `H:\DBWMaster.fdb.LogArch`).

После запуска репликации вы сможете увидеть файлы сегментов репликации в папке, указанной в “Папка оперативных логов”, сразу после выполнения любой операции с главной базой данных:




Сегменты репликации ротируются ядром Firebird, и когда отдельный сегмент заполнен, он копируется в папку архива логов. Размер сегмента по умолчанию 16 мегабайт.

Пожалуйста, обратите внимание - вам не нужно ничего делать с операционными сегментами!

После подтверждения транзакции и/или указанного времени ожидания сохраненных данных вы увидите заархивированные сегменты в папке, указанной в “Папка архивных логов”.

Архив логов репликации это набор файлов в хронологическом порядке. Эти файлы должны быть импортированы сервером реплики в базу данных реплики.

**Important!**

Для пользователей Linux — убедитесь, что владельцем папки с базой данных является пользователь `firebird`. HQbird работает под управлением пользователя “firebird” в Linux, и папка с базой данных должна иметь разрешения для “firebird” для создания папки журналов (`chown firebird -R /your/database/folder`).

### Как скопировать сегменты репликации с мастера на реплику?

Существует 2 популярных способа скопировать архивные сегменты с главного сервера на серверы-реплики: через общий сетевой ресурс и с помощью задания База данных: Передача сегментов репликации на главном сервере и База данных: Приемник файлов на реплике.

#### Общий сетевой ресурс

Вы можете предоставить общий доступ к папке с архивированными сегментами в качестве общего сетевого ресурса. В этом случае у службы Firebird должно быть достаточно прав для чтения, записи и удаления файлов на этом общем сетевом ресурсе. Обычно службы Firebird и HQbird запускаются под учетной записью LocalSystem, у которой нет доступа к общим сетевым ресурсам. Замените ее на какую-нибудь более мощную учетную запись (например, Администратора домена).

#### Отправка/приём сегментов репликации

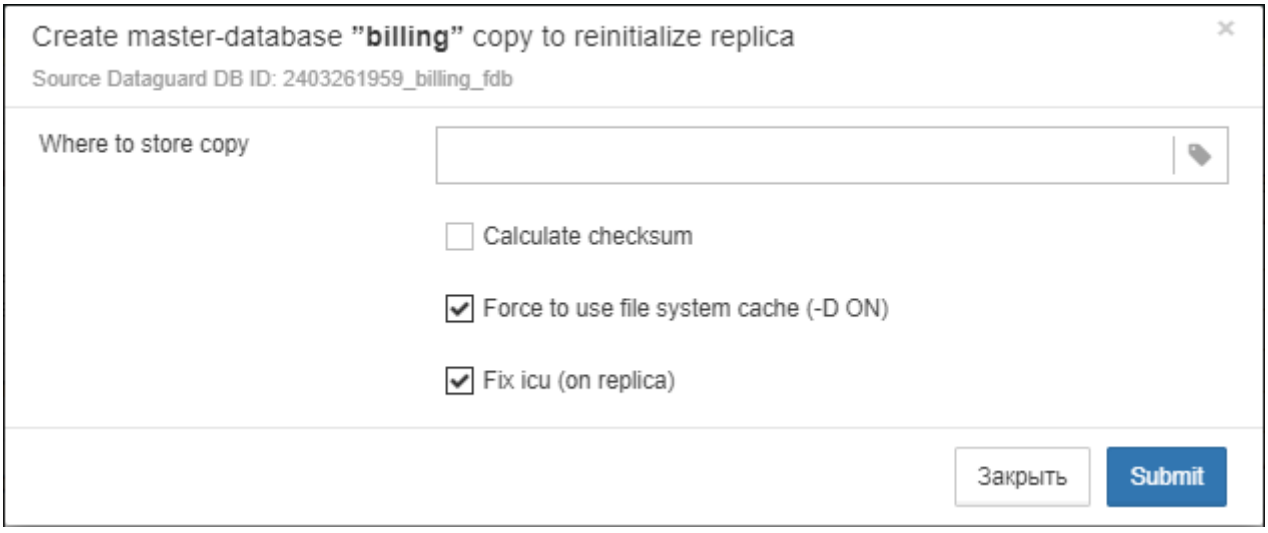
Мы рекомендуем использовать HQbird FBDataGuard для отправки сегментов репликации с главного сервера на реплику по FTP: он сжимает, шифрует и загружает сегменты на указанный FTP-сервер. На этом сервере другой HQbird FBDataGuard распаковывает сегменты и копирует их в необходимую папку для дальнейшего использования репликой.

Пожалуйста, ознакомьтесь с заданием База данных: Передача сегментов репликации для получения более подробной информации о том, как настроить передачу архивированных сегментов между мастером и репликами.

### 4.3.2. Шаг 2: Создание копии файла базы данных master

Чтобы начать репликацию, нам необходимо создать копию исходного файла базы данных, которая будет использоваться в качестве целевой для процесса репликации. Давайте будем называть такой файл базы данных "`реплика`".

В HQbird реплика будет автоматически создаваться в папке, которую вы укажете в диалоговом окне после нажатия на “Создать заново БД реплику и оправить на реплику-сервер”.




Если у вас достаточно места в папке с базой данных, **просто оставьте путь пустым** и нажмите **[ Submit ]**, и рядом с базой данных будет создана реплика. Или вы можете указать другое место назначения на локальных дисках с достаточным количеством свободного места.

**Важно!**

Если свободного места будет недостаточно (менее 105% от размера базы данных), HQbird не будет создавать копию — появится соответствующее сообщение об ошибке.

Если вы нажмете кнопку **[ Submit ]**, HQbird запустит процесс создания реплики. Об этом появится соответствующее сообщение:


В случае действия по умолчанию результирующая база данных будет находиться в той же папке, что и база данных. Имя реплики будет `DATABASE_NAME.EXT.DD-MMM-YYYY_NNNN.4replica` — например, `employee30.fdb.17-Apr-2018_142507.4replica`

Обратите внимание! Создание реплики может занять значительное время в случае большой базы данных!

Все этапы создания реплики отображаются в виде оповещений в HQbird (также отправляются по электронной почте):

| 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 proccess | 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 proccess | Start gfix -replica {1B90FF37-9776-498D-58B0-5B3C597B1571} "f:\fbdata\3.0\billing.fdb.04-Nov-2019_183000.4replica.temp" |

Убедитесь, что процесс создания реплики был успешно завершен — проверьте вкладку “Alerts”!

### 4.3.3. Шаг 3: Настройка базы данных для асинхронной репликации на сервере-реплике (подчиненном сервере)

После завершения настройки асинхронной репликации на главном сервере нам необходимо настроить ее для базы данных-реплики на экземпляре сервера-реплики.

Прежде всего, мы предполагаем, что вы успешно установили HQbird на сервер-реплику. Мы

рекомендуем использовать на сервере-реплике SuperClassic для Firebird 2.5 и SuperServer для Firebird 3.0 и выше (это конфигурации HQbird по умолчанию).

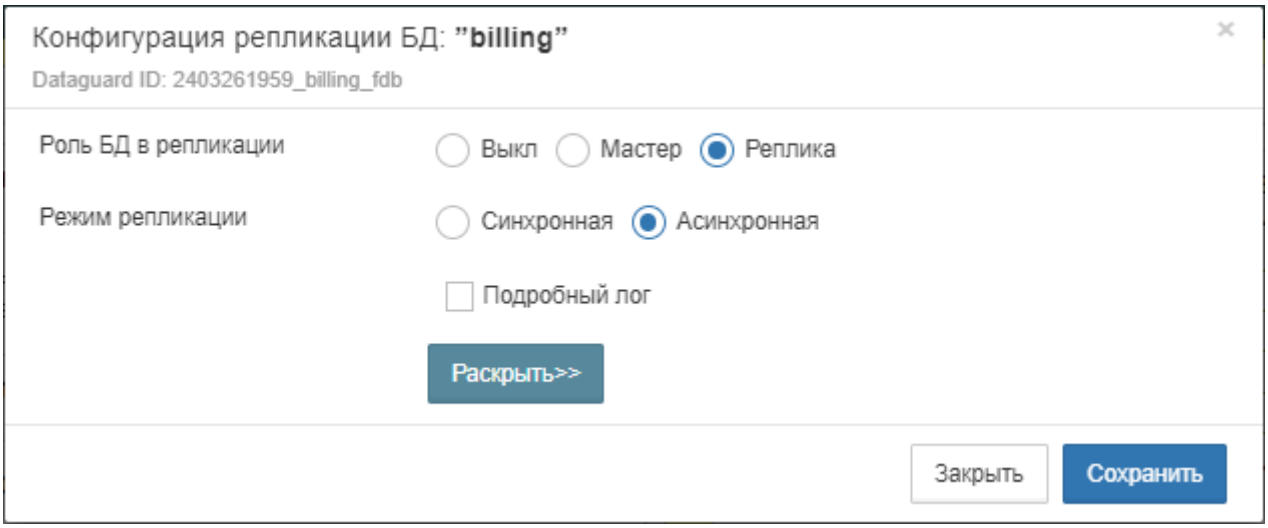
**Пользователи Firebird Classic Linux**: Если вы запускаете Firebird на сервере-реплике в классическом режиме в Linux, вам необходимо запустить дополнительный процесс Firebird replicator с помощью команды "fb_smp_server -r`.

Во-вторых, база данных реплик должна быть зарегистрирована в HQbird FBDataGuard. Если вы намерены использовать автоматическую повторную инициализацию, вы можете зарегистрировать какую-нибудь небольшую базу данных (`employee.fdb`) с требуемым именем и выполнить повторную инициализацию: в результате реплика базы данных будет автоматически перенесена с главного сервера.

В-третьих, мы предполагаем, что вам удалось настроить передачу журналов с помощью заданий База данных: Передача сегментов репликации/База данных: Приемник файлов или с помощью сетевого общего доступа.

Обратите внимание: перед регистрацией база данных должна иметь GUID реплики базы данных! Этот GUID создается автоматически, если вы воспользовались ссылкой “Reinitialize replica database”, но если вы выполняете повторную инициализацию вручную, не забудьте установить его, иначе будет выдана ошибка об отсутствии GUID базы данных.

Затем завершите настройку репликации — единственным обязательным параметром является путь к папке с архивированными сегментами репликации, и по умолчанию он уже установлен — HQbird создаст папку с журналами рядом с базой данных:




Итак, здесь не нужно ничего менять, просто нажмите **[ Сохранить ]**.

Предполагая, что база данных реплики настроена как `D:\DATABASE\DBWREPLICA.FDB`, HQbird создаст папку `D:\DATABASE\DBWREPLICA.FDB.LogArch`, и replica импортирует из нее файлы сегментов репликации.

Нажмите **[ Сохранить ]** и перезапустите службу Firebird (чтобы убедиться, что параметры репликации были применены).

После перезапуска сервер-реплика начнет использовать сегменты репликации из папки — обратите внимание, что после импорта все обработанные сегменты будут удалены. Также будет создан файл с именем `{DATABASE-GUIDE}` — это так называемый контрольный файл. Firebird хранит в нем некоторую внутреннюю информацию о ходе репликации.

Не рекомендуется хранить архивные сегменты репликации из разных баз данных в одной папке! Всегда выделяйте отдельную папку для каждой пары баз данных мастер-реплика!

# 4.4. Автоматическая инициализация и переинициализация реплики

Мы рекомендуем использовать База данных: Передача сегментов репликации на мастере и База данных: Приемник файлов на реплике, чтобы реализовать передачу и проверку целостности сегментов репликации через FTP. В этом случае также можно выполнить повторную инициализацию базы данных-реплики в один клик.

Если в База данных: Передача сегментов репликации и <hqbird-config-cloud-backup-receiver>> включены следующие параметры (по умолчанию), HQbird автоматически выполняет повторную инициализацию, включая перезапуск базы данных реплики:


Параметр “Префикс для переименования файлов репликации” следует изменить, если вы собираетесь инициализировать несколько копий главной базы данных через одну папку — в этом случае он должен быть уникальным для каждой базы данных.

В случае единой базы данных никаких изменений не требуется.

## 4.4.1. Как работает повторная инициализация

Если настроены База данных: Передача сегментов репликации/База данных: Приемник файлов, то можно выполнить полную повторную инициализацию одним щелчком мыши по “Создать заново БД реплику и отправить на реплику сервер”.

После кнопки на главной БД HQbird выполнит следующее:

1. Спросит у вас, где хранить копию базы данных (по умолчанию она находится рядом с основной базой данных, нажмите **[ Submit ]**, чтобы сохранить базу данных там).
2. Основная база данных будет скопирована (с помощью `nbackup`)
3. Созданная копия базы данных будет переведена в режим реплики.
4. для копии будет рассчитана хеш-сумма md5
5. В соответствии с настройками База данных: Передача сегментов репликации главный HQbird загрузит базу данных на указанный FTP.

Следующие шаги будут выполняться экземпляром HQbird репликой:

1. Как только реплика HQbird заметит файлы `reini*` во входящей папке FTP, База данных: Приемник файлов начнет процедуру повторной инициализации.
2. Обработка архивных сегментов будет приостановлена
3. Полученная база данных будет проверена — будет рассчитана хеш-сумма md5 и

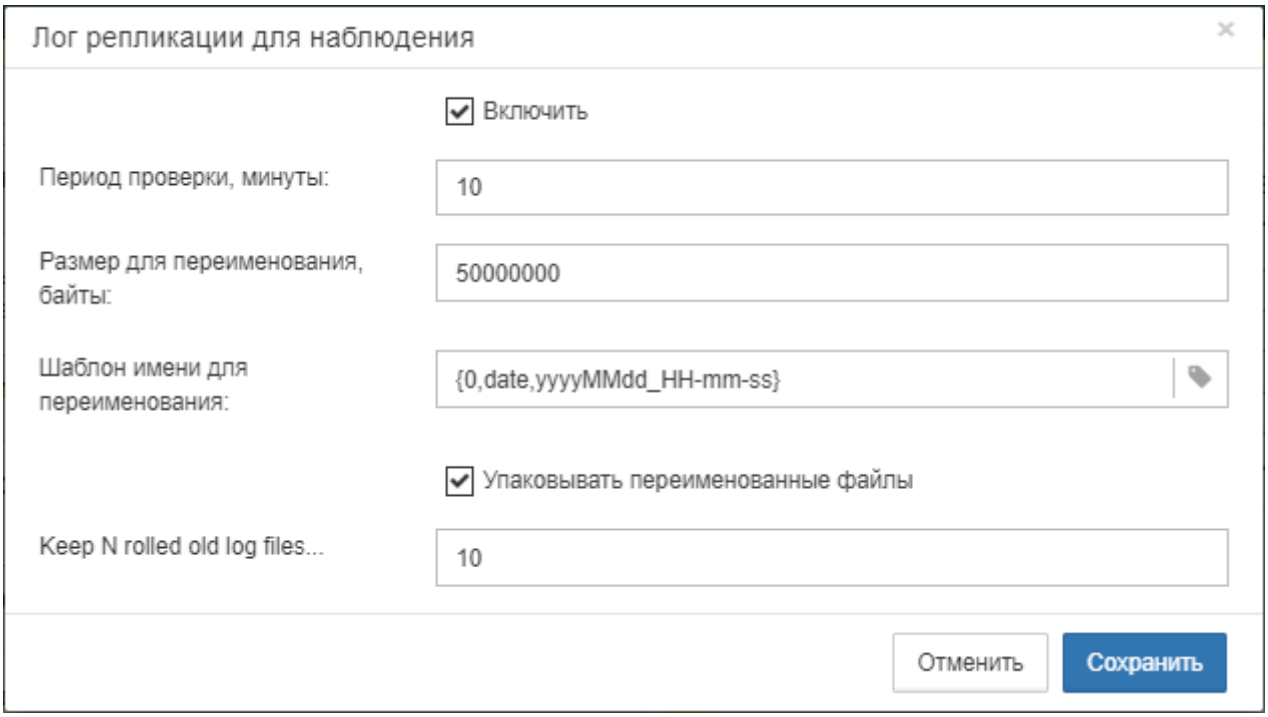
сравнена со значением в прилагаемом файле отчета.

4. Существующая реплика базы данных будет отключена, чтобы отключить всех пользователей.
5. Новая база данных-реплика будет скопирована поверх существующей базы данных.
6. Серверу реплики может потребоваться перезагрузка, чтобы увидеть новую реплику.

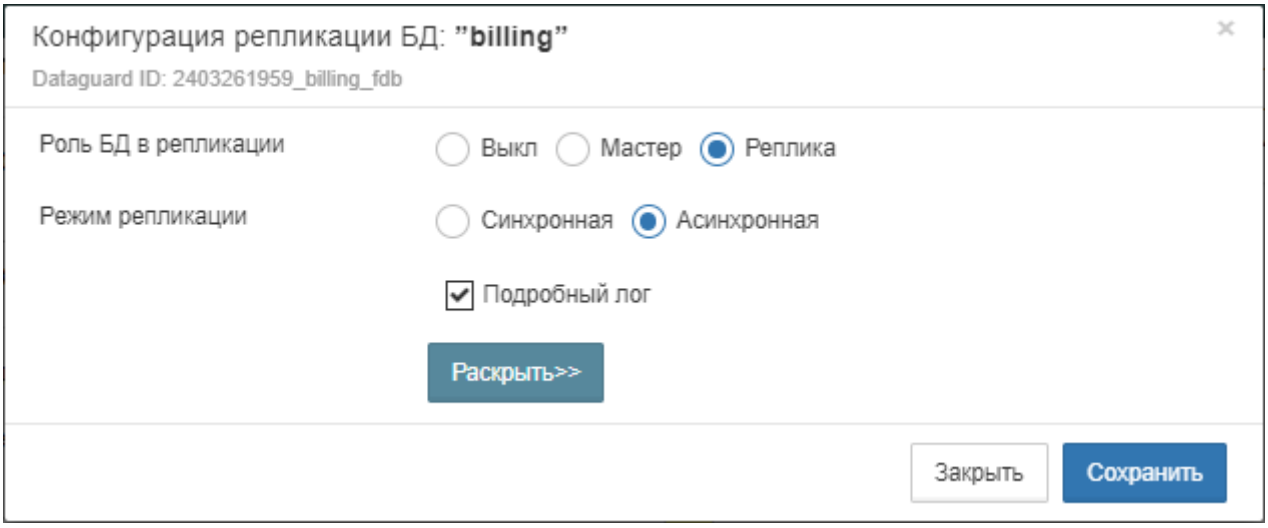
Реплика вернулась в обычный режим.

### 4.4.2. Устранение неполадок асинхронной репликации

Если вы настроили асинхронную репликацию, но она не работает, первым делом включите задание Сервер: Лог репликации на мастере и на реплике. Это задание анализирует файлы `replication.log` и в случае наличия ошибок создает соответствующее предупреждение.




Кроме того, полезно включить опцию “Подробный лог” на реплике и перезапустить Firebird. “Подробный лог” заставит Firebird записывать много подробностей о репликации в файл `replication.log` (рядом с `firebird.log`).



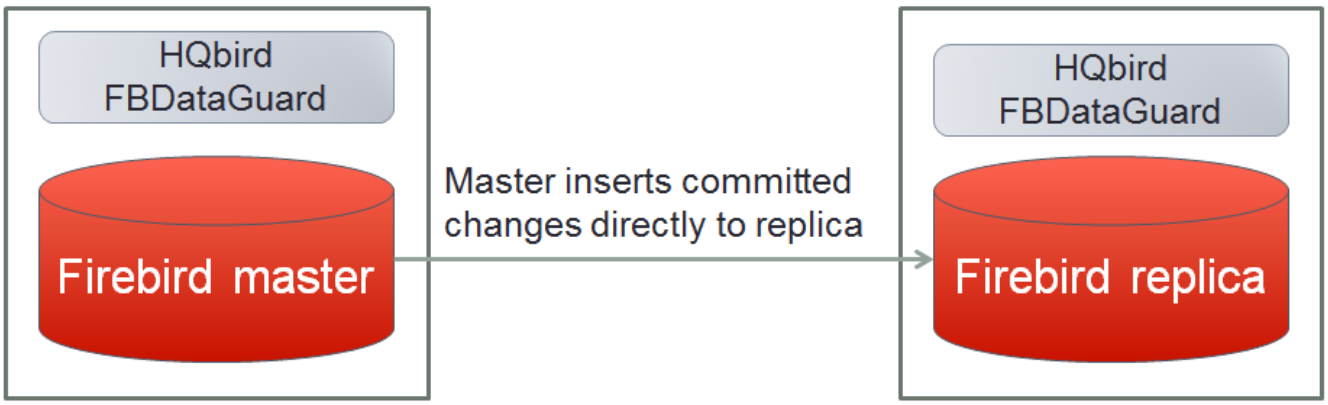

Обычно текст ошибки не требует пояснений, но поскольку некоторые популярные вопросы возникают регулярно, мы решили создать таблицу со списком основных проблем с асинхронной репликацией и способами их решения.

| **Проблема** | **Возможные причины и способы решения** |
|---|---|
| Репликация на стороне мастера настроена, но папки для оперативных или архивных сегментов (`${dbpath}.LogRepl` или `${dbpath}.LogArch`) не создаются. | HQbird создает эти папки автоматически, но для этого требуются разрешения.<br><br>В Windows: эти папки должны находиться на локальных дисках, или службы HQbird и Firebird должны запускаться с помощью “Запуск от имени” с учетной записью с повышенными привилегиями (Администратор домена?).<br><br>В Linux: папки должны иметь разрешения для пользователя “firebird”. |
| Репликация на стороне мастера настроена; папки `ReplLog` и `LogArch` созданы, но в них ничего нет. `Replication.log` пуст. | Firebird не видит конфигурацию репликации. Перезапустите службу Firebird (все соединения в случае Classic), чтобы прочитать новую конфигурацию. |
| Репликация на стороне мастера настроена; в папке `ReplLog` присутствую файл вида `databasename.log-000`, но нет файлов в `LogArch`. Кроме того, могут быть ошибки о нехватке места в `replication.log`. | Это означает, что у Firebird нет разрешения на доступ к папке `LogArch` и создание там файлов сегментов репликации (`databasename-logarch.000XXX`).<br><br>Если папка `LogArch` находится на сетевом ресурсе или подключенном диске, убедитесь, что у Firebird есть права (полный доступ) на доступ к ней. |

| Проблема | Возможные причины и способы решения |
| --- | --- |
| Опция “Подробный лог” для реплики включена, но `replication.log` пуст или не создаётся. | Иногда Firebird не может создать `replication.log` ли даже записать в уже созданный файл. Попробуйте создать его вручную и применить к нему необходимые разрешения (особенно в Linux). Подробный вывод должен записываться в `replication.log` каждые 60 секунд, даже если нет сегментов для импорта. |
| Репликация на стороне мастера в порядке, но реплика не использует сегменты репликации. Файл `replication.log` пуст. | Реплика не прочитала новую конфигурацию репликации. Перезапустите Firebird. |
| Репликация на стороне мастера в порядке, но реплика не использует сегменты репликации. Файл `replication.log` содержит ошибки прав доступа. | У реплики недостаточно прав для чтения из папки `LogArch`. Установите необходимые разрешения или запустите реплику под привилегированной учетной записью. |
| Реплика имеет ошибки в `replication.log` “Segment NNN is missing” | Проверьте, есть ли такой сегмент на стороне реплики и соответствует ли он размеру мастера. Если размер сегмента в реплике равен 0, скопируйте его вручную или инициализируйте реплику заново. |
| Реплика имеет ошибки в `replication.log` о неправильных внешних ключах и перестала использовать сегменты. | Это означает, что копия реплики десинхронизирована, поэтому некоторые записи не имеют соответствующих значений в ссылочных таблицах для указанного внешнего ключа. Реплику следует повторно инициализировать. Если вы часто видите эти ошибки, обратитесь в службу поддержки IBSurgeon. |

## 4.5. Синхронная репликация в Firebird

В случае синхронной репликации главный сервер напрямую вставляет зафиксированные изменения главной базы данных в одну или несколько баз данных-реплик:




Основными особенностями синхронной репликации являются следующие:

- Изменения буферизуются для каждой транзакции, передаются пакетами и синхронизируются при фиксации.
- Практическая задержка менее 1 секунды.
- Мастер база блокируется до применения изменений репликой
- Ошибки репликации могут либо прерывать операции, либо просто отсоединять реплику.
- Реплика доступна для запросов только для чтения (с оговорками).
- Может быть реализован автоматическое переключение между сервера (с помощью HQbird Cluster Manager).

На что обратит внимание:

- Дополнительная нагрузка на ЦП и ввод-вывод на стороне реплики.
- Требуется прямое и постоянное сетевое соединение от мастера к репликам, рекомендуется 1+Гбит/с.
- Реплику можно воссоздать онлайн, повторная инициализация синхронной репликации требует остановки мастера

Когда использовать синхронную репликацию:

- Пользовательские отказоустойчивые кластерные решения с более чем 3 узлами (особенно для веб-приложений).
- Для масштабирования производительности, перемещая операции чтения на отдельный сервер реплик (серверы отчетов, витрины данных или веб-представления, доступные только для чтения).
- В сочетании с асинхронной репликацией для масштабирования производительности.

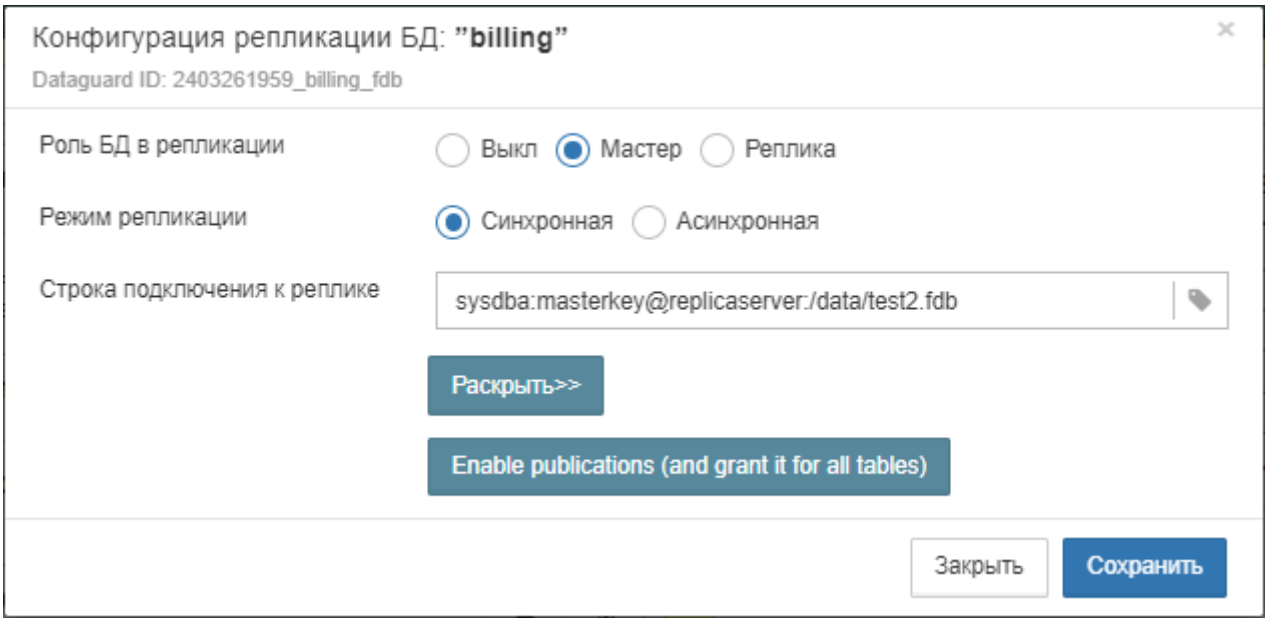
### 4.5.1. Шаги по настройке синхронной репликации

1. Остановите Firebird
2. Создайте копию главного файла базы данных, переключите ее в режим реплики и скопируйте на сервер(ы) реплики.
3. Настройте серверы реплик и базы данных для репликации с помощью HQbird FBDataGuard.
4. Запускайте серверы-реплики — перед главным сервером!
5. Настройте главный сервер и главную базу данных для репликации с помощью HQBird FBDataGuard.
6. Запустить главный сервер

Как видите, время простоя, необходимое для инициализации синхронной репликации, больше, чем время простоя для настройки асинхронной репликации, поскольку база данных реплики должна быть онлайн до запуска мастера.

### 4.5.2. Синхронная репликация на мастере и реплике

Синхронная репликация предназначена для записи изменений из главной базы данных непосредственно в базу данных-реплику. Большим преимуществом синхронной репликации является то, что задержка репликации может быть очень небольшой, но недостатком является то, что в случае потери соединения между мастер-сервером и сервером-репликой часть передаваемых данных будет потеряно.

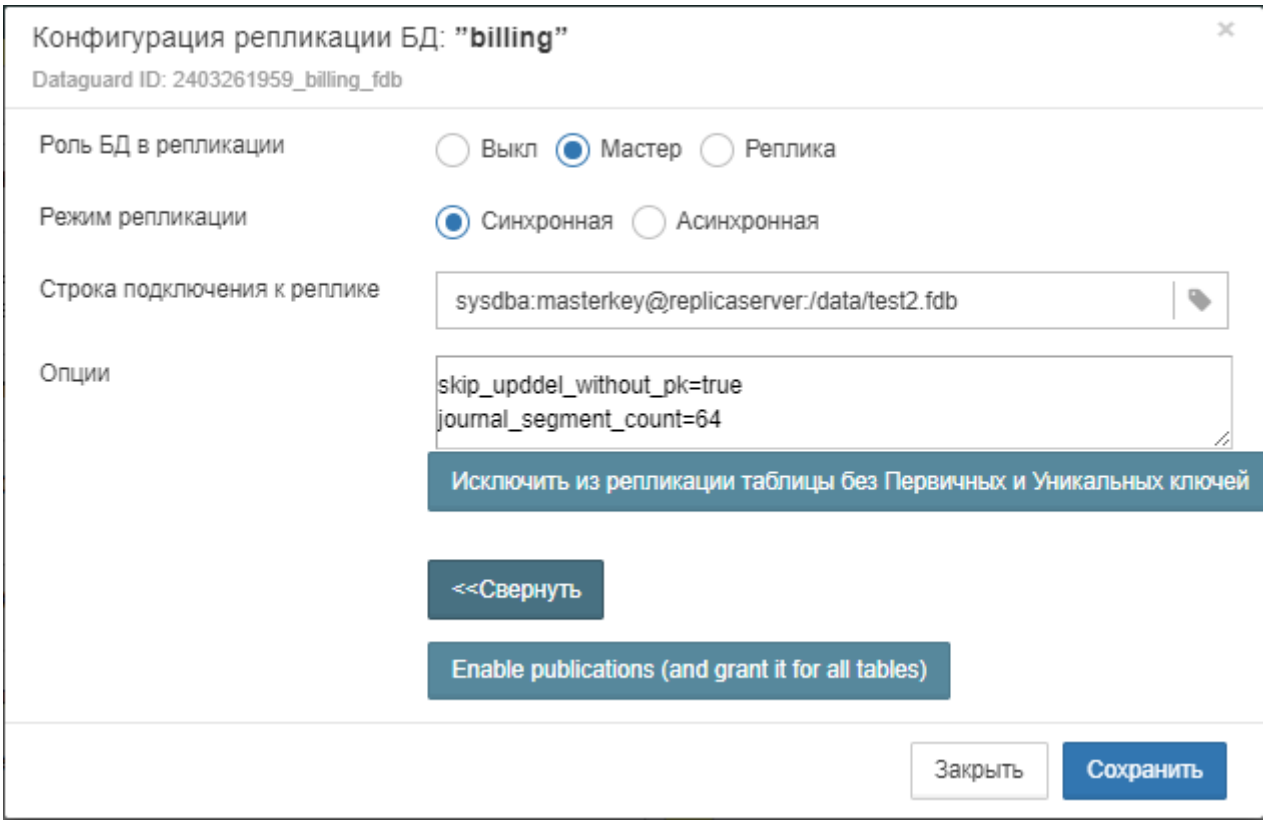



В этом примере база данных синхронной реплики находится на удаленном сервере с IP-адресом **сервер реплики** и путем `/data/test2.fdb`.

Для синхронной репликации на сервере реплики не требуется никакой настройки, за исключением `gfix –replica <master-guid>` для переключения базы данных реплики в режим реплики.

### 4.5.3. Параметры репликации для тестирования синхронной репликации

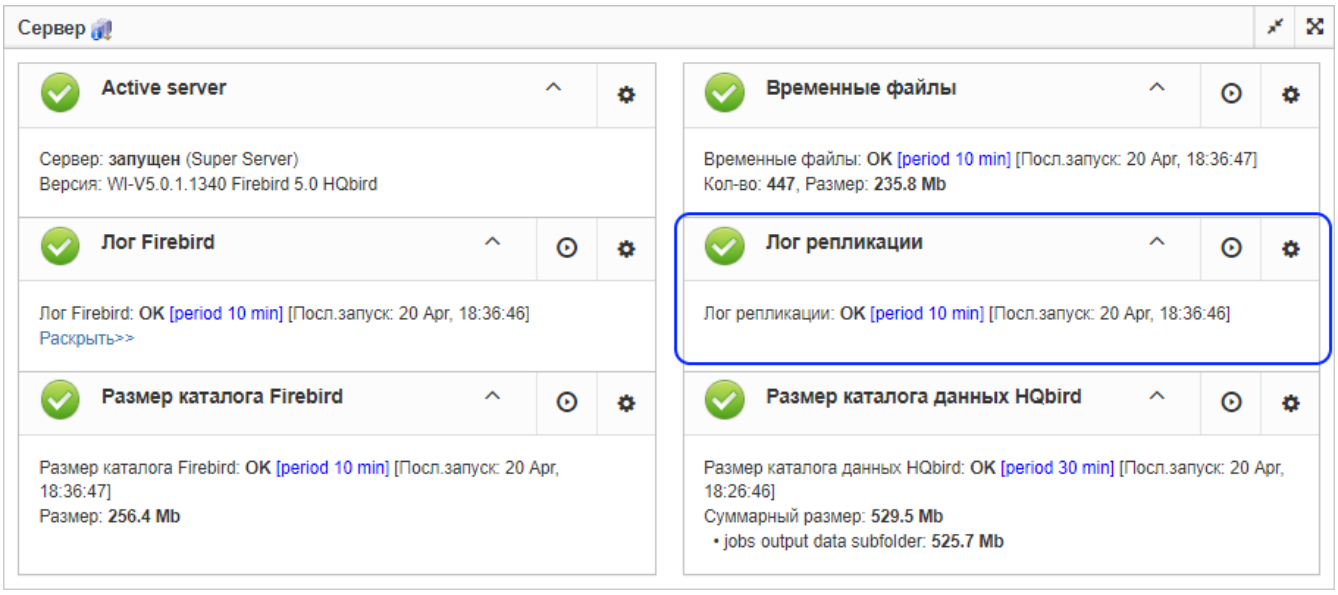
В случае тестирования синхронной репликации HQbird в производственной системе мы рекомендуем установить для параметра `disable_on_error` значение `true`.




В случае ошибки репликации он отключит репликацию, и главный сервер продолжит работать без репликации.

Для повторной инициализации репликации необходимо проанализировать журнал репликации и повторить все шаги инициализации.

Кроме того, включите задание “Лог репликации” в HQbird FBDataGuard, чтобы отслеживать журнал репликации на наличие ошибок и предупреждений:



# 4.6. Как вручную создать реплику базы данных?

*Конечно, всегда можно создать реплику с помощью простого процесса копирования: остановите Firebird на мастере, скопируйте файл базы данных, завершите настройку репликации на реплике, затем запустите Firebird. Однако HQbird поддерживает создание онлайн-реплик — подробности см. ниже.*

Если по каким-то причинам вы не можете использовать автоматическое создание реплики, вы можете создать реплику-копию главной базы данных вручную.

Начиная с HQbird 2018, можно создать файл реплики без остановки главного сервера с помощью `nbackup`. Это просто для асинхронной репликации, кроме того возможно создавать дополнительные реплики онлайн, т. е. без остановки мастера.

## 4.6.1. Создание копии онлайн (с помощью nbackup)

Рассмотрим, как создать реплику для асинхронной репликации с помощью `nbackup`:

1. заблокировать файл базы данных с помощью `nbackup`

```
nbackup –l database_path_name -user SYSDBA –pass masterkey
```

2. скопировать заблокированный файл базы данных, чтобы создать реплику

```
copy database_path_name replica_path_name
```

3. разблокировать основную базу данных

```
nbackup –n database_path_name -user SYSDBA –pass masterkey
```

4. Исправление реплики базы данных

```
nbackup –f replica_path_name_name
```

5. Переключить базу данных в режим реплики

   для Firebird 2.5 и 3.0

```
gfix replica_path_name –replica {DATABASEGUID} –user SYSDBA –pass masterkey
```

   для Firebird 4.0 и выше

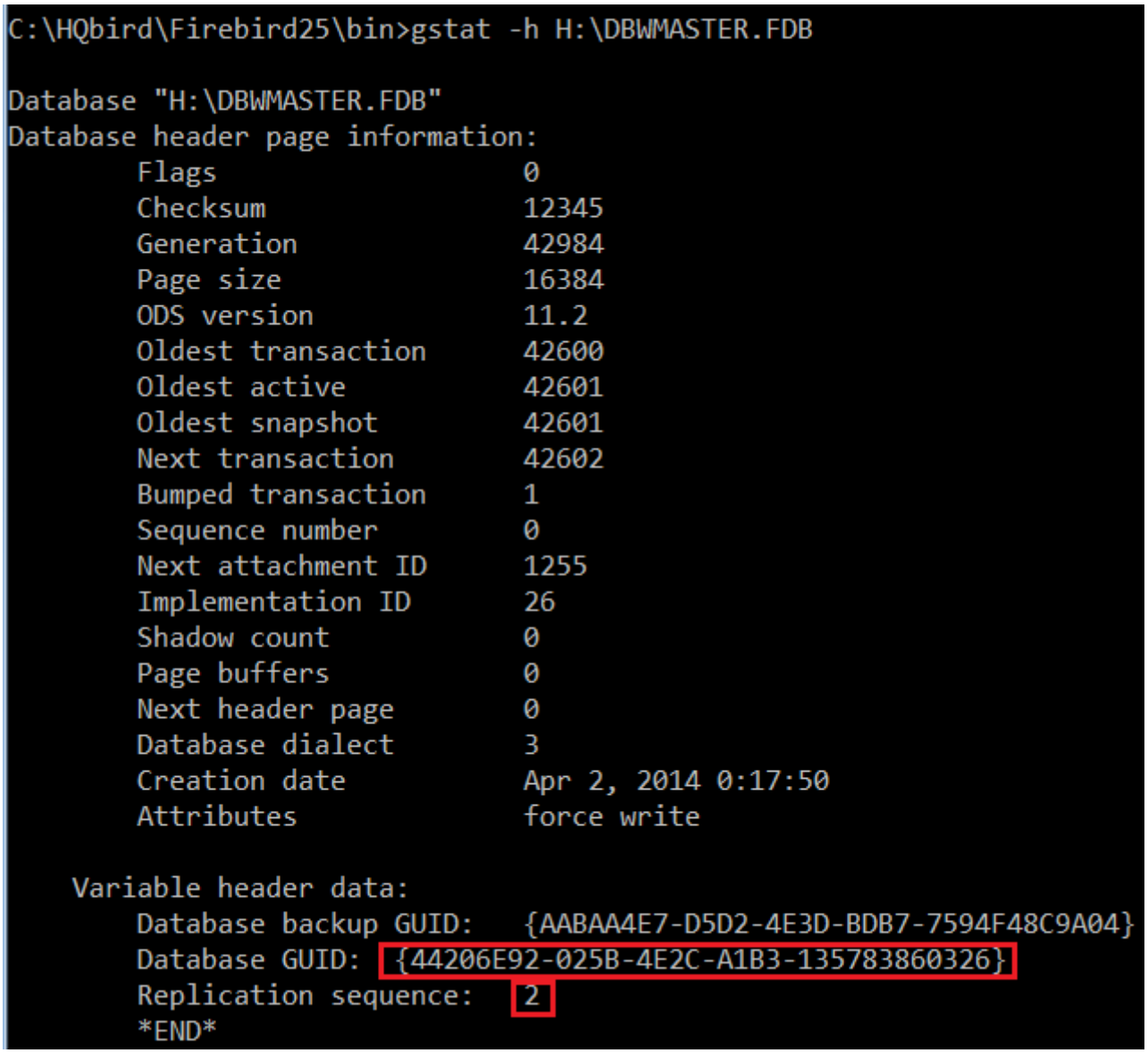
```
gfix replica_path_name –replica <replica_mode> –user SYSDBA –pass masterkey
```

```
<replica_mode> ::= read_only | read_write
```

### 4.6.2. Что такое {DATABASEGUID}?

GUID базы данных — это уникальный идентификатор главной базы данных.

Его можно получить {DATABASEGUIDE}, выполнив команду `gstat –h`:




Чтобы переключить базу данных в режим реплики, выполните следующую команду:

```
gfix disk:\path\mydatabase.fdb -replica {guid} -user SYSDBA -pass masterkey
```

Если вы не видите GUID базы данных в выводе `gstat –h`, подключитесь к главной базе данных, используя двоичные файлы Firebird из дистрибутива HQbird (через `isql` или любым другим приложением), и снова запустите `gstat –h`.

### 4.6.3. Как перевести базу данных реплики в мастер режим

Чтобы переключить базу данных в обычный (главный) режим, выполните ту же команду с пустым {} вместо GUID базы данных:

для Firebird 2.5 и 3.0

```
gfix disk:\path\mydatabase.fdb -replica {} -user SYSDBA -pass masterkey
```

для Firebird 4.0 и выше

```
gfix replica_path_name –replica none –user SYSDBA –pass masterkey
```

# 4.7. Как отличить главную базу данных от реплики

## 4.7.1. Используйте gstat -h

Если вы запустите `gstat –h database_name`, выходные данные будут содержать ключевое слово “replica” в разделе Attributes для базы данных, настроенной как реплика:

```
Database "D:\O30.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
```

Для основной базы данных в атрибутах нет специальных отметок.

## 4.7.2. Через SQL-запрос к контекстной переменной

В Firebird 2.5 и 3.0 в пространстве `SYSTEM` есть контекстная переменная `` `REPLICA ``, содержащая информацию о состоянии базы данных:

```
SQL> select RDB$GET_CONTEXT('SYSTEM', 'REPLICA') from rdb$database;

RDB$GET_CONTEXT
```

```
=============================================================
FALSE
```

В Firebird 4.0 используйте другую контекстную переменную REPLICA_MODE:

```
SQL> select RDB$GET_CONTEXT('SYSTEM', 'REPLICA_MODE') from rdb$database;

RDB$GET_CONTEXT
===============================================================
READ-ONLY
```

Также в Firebird 4.0 и выше вы можете использовать таблицу мониторинга MON$DATABASE:

```
SQL> SELECT MON$REPLICA_MODE FROM MON$DATABASE;

MON$REPLICA_MODE
================
               1
```

Режимы реплики базы данных:

- 0 — не реплика
- 1 — read-only реплика
- 2 — read-write реплика

# 4.8. Дополнительные параметры репликации

Есть возможность указать несколько дополнительных параметров для тонкой настройки процесса репликации. Эти параметры можно указать в “Опции” диалога настройки репликации.

1. Размер локального буфера, используемого для накопления событий репликации, которые можно отложить до фиксации/отката транзакции. Чем больше это значение, тем меньше сетевых обменов между главным и подчиненным хостами. Однако это требует больше времени на “контрольные точки” репликации (время на синхронизацию исходной базы данных с ее репликой).

```
buffer_size = 1048576
```

2. Если этот параметр включен, любая ошибка во время репликации приводит к тому, что мастер прекращает репликацию изменений и продолжает работать в обычном режиме. В противном случае (поведение по умолчанию) мастер сообщает об ошибке.

```
disable_on_error = false
```

3. Если этот параметр включен, реплицированные записи сжимаются с помощью RLE перед передачей и распаковываются на подчиненной стороне. Это уменьшает трафик и (косвенно) количество round-trips за счет дополнительных циклов ЦП с обеих сторон.

```
compress_records = false
```

4. Если этот параметр включен, конфликтующие записи в целевой базе данных изменяются в соответствии с записями в основной базе данных. В частности:
   - если происходит вставка и существует целевая запись, то она обновляется;
   - если происходит обновление и целевая запись не существует, она вставляется;
   - если произошло удаление, а целевая запись не существует, она игнорируется.

```
master_priority = false
```

1. Шаблон (регулярное выражение), определяющий, какие таблицы необходимо включать в репликацию. По умолчанию все таблицы реплицируются.

```
include_filter
```

2. Шаблон (регулярное выражение), определяющий, какие таблицы необходимо исключить из репликации. По умолчанию все таблицы реплицируются.

```
exclude_filter
```

3. Если этот параметр включен, таблицы без первичного ключа (или уникального индекса) исключаются из репликации. По умолчанию все таблицы реплицируются.

```
exclude_without_pk = false
```

4. Программа (команда оболочки с аргументами), которая выполняется, когда текущий сеанс репликации обнаруживает критическую ошибку. Эта команда выполняется один раз для каждого неудачного сеанса репликации. Обратите внимание, что программа выполняется синхронно, и сервер ожидает ее завершения, прежде чем продолжить свою работу.

```
alert_command
```

5. Префикс для имен файлов журнала репликации. К нему автоматически будет добавлен порядковый последовательный номер. Если не указано, в качестве префикса используется имя файла базы данных (без пути).

```
log_file_prefix
```

6. Максимально допустимый размер для одного сегмента репликации. Он должен как минимум вдвое превышать указанный *buffer_size*.

```
log_segment_size = 16777216
```

7. Максимально допустимое количество полных сегментов репликации. По достижении этого предела процесс репликации задерживается на *log_archive_timeout* секунд (см. ниже), чтобы архивирование могло наверстать упущенное. Если какой-либо из полных сегментов не заархивирован и не помечен для повторного использования в течение таймаута, репликация завершится ошибкой.

   Ноль означает неограниченное количество сегментов, ожидающих архивирования.

```
log_segment_count = 8
```

# Глава 5. Улучшения производительности

## 5.1. Пул внешних подключений

HQbird поддерживает пул внешних подключений для Firebird 2.5 и Firebird 3. Стандартная сборка Firebird 4.0 и выше поддерживает создание внешних пулов соединений “из коробки”.

Пул внешних подключений позволяет выполнять инструкции `EXECUTE STATEMENT ON EXTERNAL` с меньшими затратами на повторное соединение с внешней базой данных.

Обратите внимание: этот пул выделяется для каждого экземпляра Firebird.

Эта функция настраивается в файле `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
```

С точки зрения приложения не требуется никаких дополнительных действий для использования или неиспользования — пул включается или отключается в конфигурации сервера и абсолютно незаметно для приложений.

Для управления пулом существуют следующие команды:

- изменение размера пула

  ```
  ALTER EXTERNAL CONNECTIONS POOL SET SIZE N
  ```

  Пример: эта команда устанавливает размер пула на 190 подключений.

```
ALTER EXTERNAL CONNECTIONS POOL SET SIZE 190
```

- изменение времени жизни неактивного соединение в пуле

```
ALTER EXTERNAL CONNECTIONS POOL
SET LIFETIME N {SECOND | MINUTE | HOUR}
```

Пример: эта команда ограничивает время жизни соединения в пуле до 1 часа.

```
ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME 1 HOUR
```

- очистить все соединения в пуле

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL
```

- очистить самое старое соединение в пуле

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST
```

Для получения информации о состоянии пула были введены новые контекстные переменные. Следующий пример демонстрирует их использование

```
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;
```

# 5.2. Кеш подготовленных запросов

В HQbird есть возможность улучшить производительность движка Firebird (версии 4.0 и 5.0) в случае большого количества частых и быстрых SQL-запросов: кэширование подготовленных операторов SQL на стороне сервера. Начиная с Firebird 5.0 эта функция доступна в стандартной сборке “из коробки”.

## 5.2.1. Кеш подготовленных запросов в Firebird 3.0 и 4.0

Эту функцию можно включить в `firebird.conf` с помощью параметра `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
```

Это число указывает, сколько последних запросов для каждого подключения к базе данных необходимо кэшировать.

Чтобы применить новое значение, потребуется перезапуск Firebird.

По умолчанию `DSQLCacheSize` равен 0, что означает выключен. Мы рекомендуем осторожно включать его: начните со значений, таких как 4, 8, 16, чтобы найти лучший эффект производительности.

Обратите внимание: включение кэширования подготовленных операторов увеличивает использование памяти.

## 5.2.2. Кеш подготовленных запросов в Firebird 5.0

В Firebird 5.0 кеш подготовленных запросов был переработан и включён в “ванильную” сборку.

Кеш подготовленных запросов управляется параметром `MaxStatementCacheSize` в `firebird.conf`.

```
# ----------------------------
# Maximum statement cache size
#
# The maximum amount of RAM used to cache unused DSQL compiled statements.
# If set to 0 (zero), statement cache is disabled.
#
# Per-database configurable.
#
# Type: integer
#
```

```
#MaxStatementCacheSize = 2M
```

По умолчанию кеш подготовленных запросов включен. Для его отключения необходимо установить параметр `MaxStatementCacheSize` в 0.

# 5.3. TempSpaceLogThreshold: мониторинг запросов с большими сортировками и BLOB

HQbird имеет новый параметр в `firebird.conf` in Firebird 2.5, Firebird 3.0, Firebird 4.0 и Firebird 5.0:

```
TempSpaceLogThreshold=1000000000 #bytes
```

Когда Firebird видит этот параметр, он записывает тексты запросов в `firebird.log` для запросов, которые производят большие сортировки (запросы с `GROUP BY`, `ORDER BY` и т. д.).

Когда такой запрос создаст файл сортировки, размер которого превышает указанный порог, в `firebird.log` появится следующее сообщение:

```
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**

общий размер файла сортировки

**Cached**

часть сортировки, уместившаяся во временное пространство (задается параметром `TempCacheLimit`)

**On disk**

часть сортировки, которая была сохранена во временный файл, который может быть кэширован в памяти ОС или сохранен на диске (в папке, указанной параметром `TempDirectories`, или в временной папке по умолчанию)

Для очень больших BLOB в файле `firebird.log` появится следующее сообщение

```
SRV-DB1 Tue Nov 27 17:35:39 2018
    Temporary space of type "blob" has exceeded threshold of 500000000 bytes.
    Total size: 500377437, cached: 0 bytes, on disk: 501219328 bytes.
```

Используйте `TempSpaceLogThreshold` для поиска неоптимизированных запросов с большой сортировкой и большими BLOB-объектами. Начиная с Firebird 3.0 в нём также будет сообщаться о больших хеш-таблицах (возникают при HASH JOIN).

Если вы столкнулись с такими запросами, оптимизируйте их либо путем изменения самого SQL-запроса, либо попробуйте включить параметр `SortDataStorageThreshold`.

# 5.4. SortDataStorageThreshold: REFETCH вместо SORT для широких наборов данных

HQbird поддерживает новый метод оптимизации REFETCH. Стандартная сборка Firebird версии 4.0 и выше поддерживает этот алгоритм оптимизации “из коробки”.

В HQbird появился новый параметр `SortDataStorageThreshold` в `firebird.conf` (Firebird 3.0+):

```
SortDataStorageThreshold=16384 # bytes
```

Начиная с версии 4.0 этот параметр был переименован в `InlineSortThreshold`.

```
InlineSortThreshold=16384 # bytes
```

Если размер записи, возвращаемой SQL-запросом, будет больше указанного порога, Firebird будет использовать другой подход для сортировки наборов записей: REFETCH вместо SORT.

Например, у нас есть следующий запрос

```
select tdetl.name_detl
    ,tmain.name_main
    ,tdetl.long_description
from tdetl
join tmain on tdetl.pid=tmain.id
order by tdetl.name_detl
```

со следующим планом выполнения:

```
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)
```

В этом случае размер каждой сортируемой записи составляет 32860+36 байт. Это может привести к созданию очень больших файлов сортировки, которые будут записаны на диск, и выполнение запроса может замедлиться.

С параметром `SortDataStorageThreshold=16384` или `InlineSortThreshold=16384` Firebird будет использовать метод доступа REFETCH, гв котором сортируется только ключ, а данные перечитываются из базы данных:

```
Select Expression
    -> Refetch
        -> Sort (record length: 76, key length: 36)
            -> Nested Loop Join (inner)
```

Такой подход позволяет существенно (в 2-5 раз) ускорить запросы с сортировкой очень широких наборов записей (обычно это тяжелые отчеты).

**Обратите внимание!**

Не рекомендуется устанавливать `SortDataStorageThreshold` (`InlineSortThreshold`) меньше чем 2048 байт.

## 5.5. Многопоточные sweep, backup, restore

В HQbird появилась возможность многопоточного выполнения очистки, резервного копирования и восстановления, что ускоряет их работу от 2x до 6 раз (в зависимости от конкретной базы). Многопоточные операции работают в HQbird для Firebird 2.5, 3.0 и 4.0 на любых архитектурах — Classic, SuperClassic, SuperServer. Для Firebird 5.0 многопоточные операции доступны в “ванильной” версии из коробки.

Чтобы включить многопоточное выполнение, утилиты командной строки `gfix` и `gbak` имеют параметр `-par n`, где `n` — количество потоков, которые будут задействованы в конкретной операции. На практике выбор числа `n` следует соотносить с количеством доступных ядер процессора.

Примеры:

- `gfix -sweep database -par 8 …`
- `gbak -b database backup -par 8 …`
- `gbak -c backup database -par 8 …`

Кроме того, чтобы ограничивать количество рабочих потоков и устанавливать их число по умолчанию, в `firebird.conf` введены два новых параметра, которые влияют только на sweep и restore, но не на резервное копирование:

```
# ============================
# 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
```

Пример: если вы установите `MaxParallelWorkers = 10`, вы сможете выполнять

- run `gfix -sweep database -par 10`
- run `gfix -sweep database -par 5` and `gbak -c -par 5 …`

То есть всего будет использовано не более 10 потоков. В случае превышения (например, если вы установили 6 потоков на sweep и 6 потоков на restore), для процесса, превышающего лимит, будет выведено сообщение “No enough free worker attachments”).

Таким образом, чтобы включить многопоточные возможности`sweep и restore, вы должны установить параметр `MaxParallelWorkers` в `firebird.conf`.

```
MaxParallelWorkers = 64
```

и перезапустить Firebird.

ParallelWorkers по умолчанию устанавливает количество потоков, используемых для sweep и restore, если не указана опция –par n.

```
#  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
```

Например, если ParallelWorkers = 8, то запуск

```
gfix –sweep
```

без опции -par n будет использоваться 8 потоков для параллельного выполнения sweep.

До Firebird 5.0 в HQbird при восстановлении заполнение таблиц из резервной копии всегда выполняется в одном потоке, а распараллеливается только создание индексов. Таким образом, ускорение восстановления зависит от количества индексов в базе данных и их размера. Также параметр ParallelWorkers автоматически влияет на создание индексов, выполняемых операциями CREATE INDEX и ALTER INDEX… ACTIVE.

Начиная с Firebird 5.0 заполнение таблиц при восстановлении тоже используется параллелизм.

Как упоминалось выше, эти параметры не влияют на резервное копирование. Многопоточность резервного копирования регулируется только параметром –par n в командной строке:

- gbak –b –par 6 …
- gbak –b –par 8 –se …

Если база данных находится в состоянии single shutdown, когда к базе данных разрешено только 1 соединение, то в версии 2.5 sweep, и резервное копирование с -par 2 или более будут вызывать ошибку через несколько секунд после запуска:

- sweep — connection lost to database
- backup — ERROR: database … shutdown (по протоколу xnet строка с этим сообщением не будет отображаться в журнале резервного копирования)

Это связано с тем, что для этих операций требуется соответствующее количество подключений к базе данных, более 1.

В 3.0 только резервное копирование выдает ошибку “ERROR: database … shutdown”, sweep будет работать

Многопоточное восстановление Firebird 2.5, 3.0, 4.0 и 5.0 создает базу данных в режиме shutdown multi, поэтому такие ошибки не возникают. Однако существует риск подключения к базе данных в процессе восстановления других приложений от SYSDBA или владельца.

**Notes**

- Новые параметры в `firebird.conf` влияют только на очистку и восстановление. Чтобы упростить администрирование и устранить двусмысленность, рекомендуется всегда явно указывать параметр `-par n` для `gfix` и `gbak`, если вам нужно выполнять многопоточные операции sweep, восстановления и резервного копирования. Например, если вы установите `ParallelWorkers = 4` и не укажете `-par n`, то sweep и восстановление будут использовать 4 потока по умолчанию, а резервное копирование будет использовать 1 поток, поскольку не использует значения из `firebird.conf` ни локально, ни с помощью `-se`.
- Прирост производительности не обязательно зависит от количества ядер процессора и их соответствия заданному значению `–par n`. Это зависит от количества ядер, архитектуры Firebird и производительности дисковой подсистемы (IOPS). Поэтому оптимальное значение `–par n` для вашей системы необходимо подобрать экспериментально.

# 5.6. BLOB_APPEND function

Обычный оператор || (конкатенация) с аргументами BLOB создает временный BLOB для каждой пары аргументов с BLOB. Это может привести к чрезмерному потреблению памяти и увеличению размера файла базы данных. Функция BLOB_APPEND предназначена для объединения BLOB-объектов без создания промежуточных BLOB-объектов.

Результирующий BLOB-объект функции BLOB_APPEND остается открытым для записи, а не закрывается сразу после заполнения данными. Т.е. в такой BLOB можно добавлять столько раз, сколько необходимо. Движок помечает такой объект новым внутренним флагом BLB_close_on_read и автоматически закрывает его при необходимости.

**Available in**: DSQL, PSQL.

*Syntax:*

```
BLOB_APPEND(<blob> [, <value1>, ... <valueN]>
```

*Таблица 1. Parameters of BLOB_APPEND function*

| Параметр | Описание |
|---|---|
| blob | BLOB или NULL. |
| value | Значение любого типа. |

**Return type**: временный не закрытый BLOB (т.е. открытый для записи), помеченный флагом BLB_close_on_read.

Входные аргументы:

- Первый аргумент BLOB или NULL. Возможны следующие варианты:
  - NULL: создаётся новый временный не закрытый BLOB с флагом BLB_close_on_read
  - постоянный BLOB (из таблицы) или временный уже закрытый BLOB: создаст новый пустой незакрытый BLOB с флагом BLB_close_on_read и к нему будет добавлено содержимое первого BLOB
  - временный незакрытый BLOB с флагом BLB_close_on_read: будет использоваться далее
- остальные аргументы могут быть любого типа. Для них определено следующее поведение:
  - NULL игнорируется
  - не-BLOB будут преобразованы в строку и добавлены к содержимому результата
  - BLOB, при необходимости транслитерируются в набор символов первого аргумента, а их содержимое добавляется к результату.

Функция BLOB_APPEND возвращает временный незакрытый BLOB с флагом BLB_close_on_read. Это либо новый BLOB, либо тот же самый, что и в первом аргументе. Таким образом, серия операций типа blob = BLOB_APPEND (blob, ⋯) приведет к созданию не более одного BLOB (если вы не попытаетесь добавить BLOB к самому себе). Этот BLOB-

объект будет автоматически закрыт механизмом, когда клиент попытается прочитать его, назначить таблице или использовать в других выражениях, требующих чтения содержимого.

Проверка BLOB-объекта на наличие значения NULL с использованием оператора IS [NOT] NULL не приводит к его чтению, поэтому временный BLOB-объект с флагом BLB_close_on_read не будет закрыт во время такой проверки.

```
execute block
returns (b blob sub_type text)
as
begin
  -- создаёт временный незакрытый BLOB
  -- и запишет в него строку из второго аргумента
  b = blob_append(null, 'Hello ');
  -- добавляет две строки во временный BLOB без его закрытия
  b = blob_append(b, 'World', '!');
  -- сравнение BLOB со строкой закроет его, потому что для этого вам нужно прочитать
BLOB
  if (b = 'Hello World!') then
  begin
  -- ...
  end
  -- создаст временный закрытый BLOB, добавив к нему строку
  b = b || 'Close';
  suspend;
end
```

Используйте функции LIST и BLOB_APPEND для объединения BLOB. Это позволит сэкономить потребление памяти, дисковый ввод-вывод и предотвратить рост базы данных из-за создания множества временных BLOB-объектов при использовании операторов конкатенации.

Допустим, вам нужно собрать JSON на стороне сервера. У нас есть PSQL пакет JSON_UTILS с набором функций для преобразования примитивных типов данных в нотацию JSON. Тогда построение JSON с использованием функции BLOB_APPEND будет выглядеть так:

```
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
```

Аналогичный пример с использованием обычного оператора конкатенации || на порядок медленнее и выполняет в 1000 раз больше операций записи на диск.

# 5.7. Преобразование LEFT joins в INNER

HQbird позволяет преобразовывать `LEFT JOIN` в `INNER JOIN`, если условие `WHERE` нарушает правила внешнего соединения.

Начиная с Firebird 5.0 это функциональность доступна в “ванильной” версии Firebird.

Пример:

```
SELECT *
FROM T1 LEFT JOIN T2 ON T1.ID = T2.ID
WHERE T2.FIELD1 = 0
```

В этом случае условие `T2.FIELD1 = 0` эффективно удаляет все “поддельные NULL” строки `T2`, поэтому результат тот же, что и для `INNER JOIN`. Однако оптимизатор вынужден использовать порядок соединения T1→T2, хотя мог бы также рассмотреть T2→T1. Имеет смысл обнаружить этот случай во время обработки соединения и заменить `LEFT` на `INNER` перед началом оптимизации.

В первую очередь это предназначено для улучшения автоматически генерируемых (например, ORM) запросов.

Эта оптимизация не будет включена, если фильтруется значение `NULL`, например

```
SELECT *
FROM T1 LEFT JOIN T2 ON T1.ID = T2.ID
WHERE T2.ID IS NULL
```

или

```
SELECT *
FROM T1 LEFT JOIN T2 ON T1.ID = T2.ID
WHERE T2.ID IS NOT NULL
```

# Глава 6. Мониторинг

## 6.1. Мониторинг с помощью HQbird FBDataGuard

### 6.1.1. Обзор

HQbird контролирует все аспекты функционирования сервера и базы данных Firebird, включая непрерывный и детальный мониторинг.

Непрерывный мониторинг осуществляет HQbird FBDataGuard. Это малоинвазивный, но очень эффективный мониторинг, который может помочь найти и устранить большинство проблем с производительностью и стабильностью баз данных.

FBDataGuard выполняет следующие действия по мониторингу:

- Дополнительный мониторинг производительности с помощью TraceAPI и MON$
- Мониторинг динамики маркеров транзакций (Next, OAT, OIT, OST, активные транзакции)
- Мониторинг активности таблицы блокировок (queues, deadlocks, mutexes)
- Мониторинг журнала Firebird (errors, warnings, messages)
- Количество подключенных пользователей
- Мониторинг свободного места для сервера, баз данных и их резервных копий
- Мониторинг работоспособности посредством анализа метаданных базы данных и общей доступности сервера и баз данных.
- Количество и размер временных файлов Firebird (сортировка и т. д.)
- Мониторинг индексов: целостность и активность
- Общая проверка базы данных Firebird
- Мониторинг статусов резервных копий
- Мониторинг доступности реплик

FBDataGuard может графически представлять информацию, собранную в ходе мониторинга, например о транзакциях и количестве пользователей:

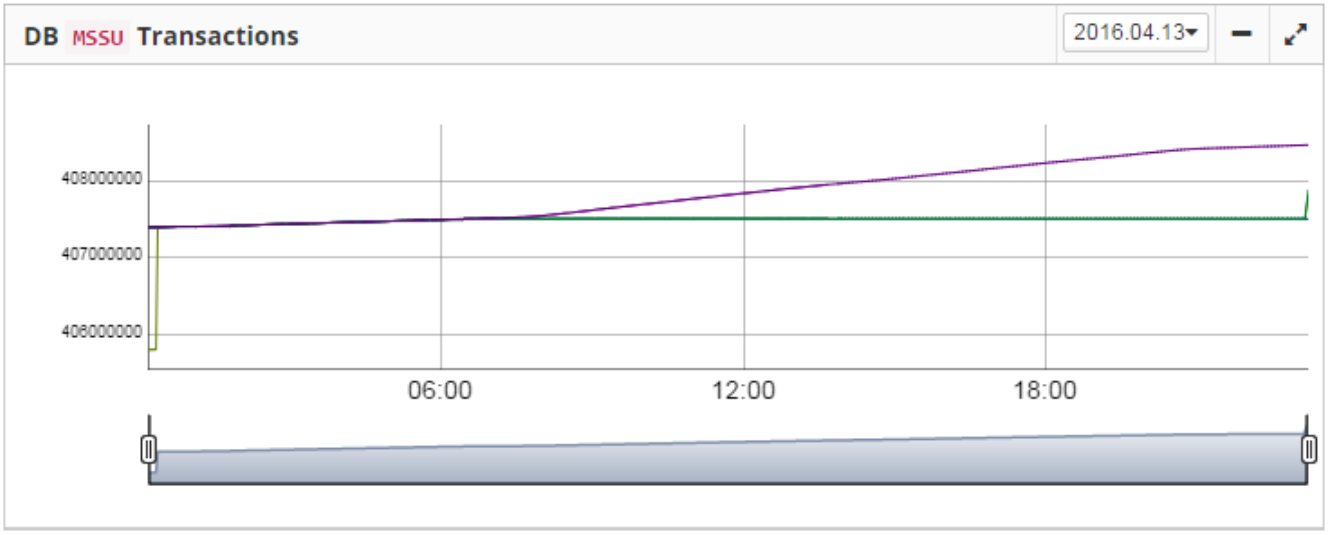



*Рисунок 40. Динамика транзакций в FBDataGuard.*

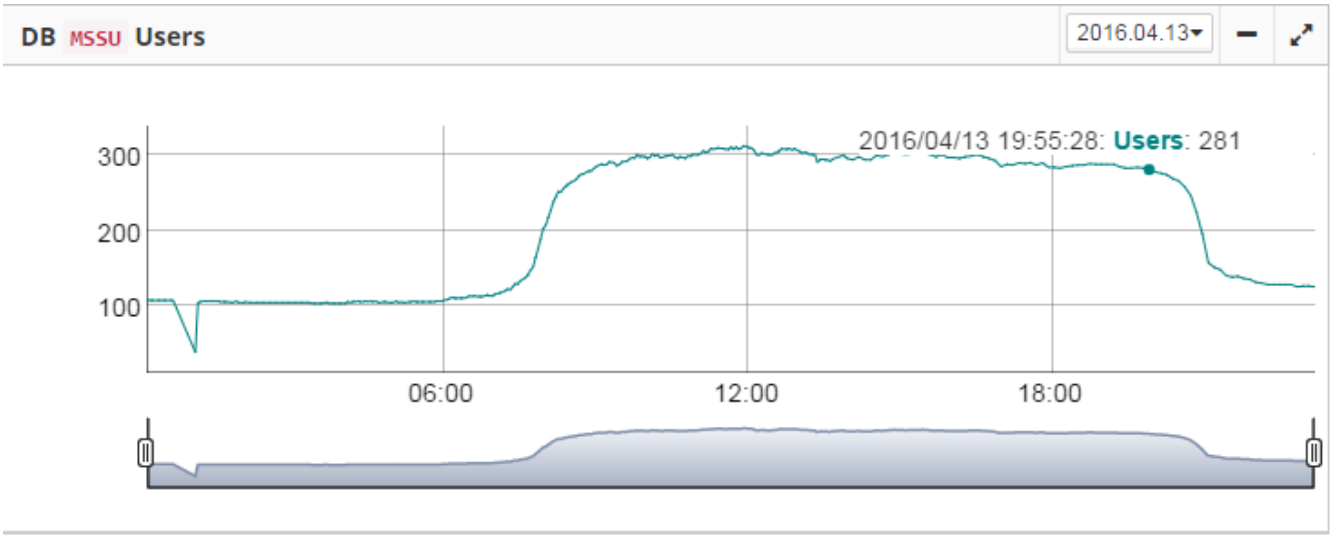


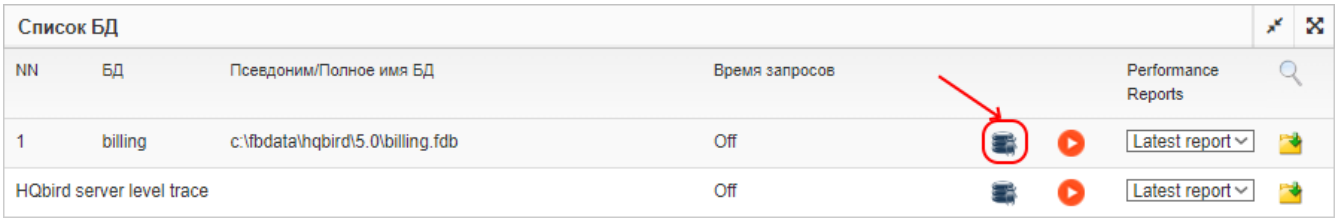

*Рисунок 41. Количество пользователей.*

## 6.1.2. Автоматический мониторинг с помощью FBDataGuard (Trace API)

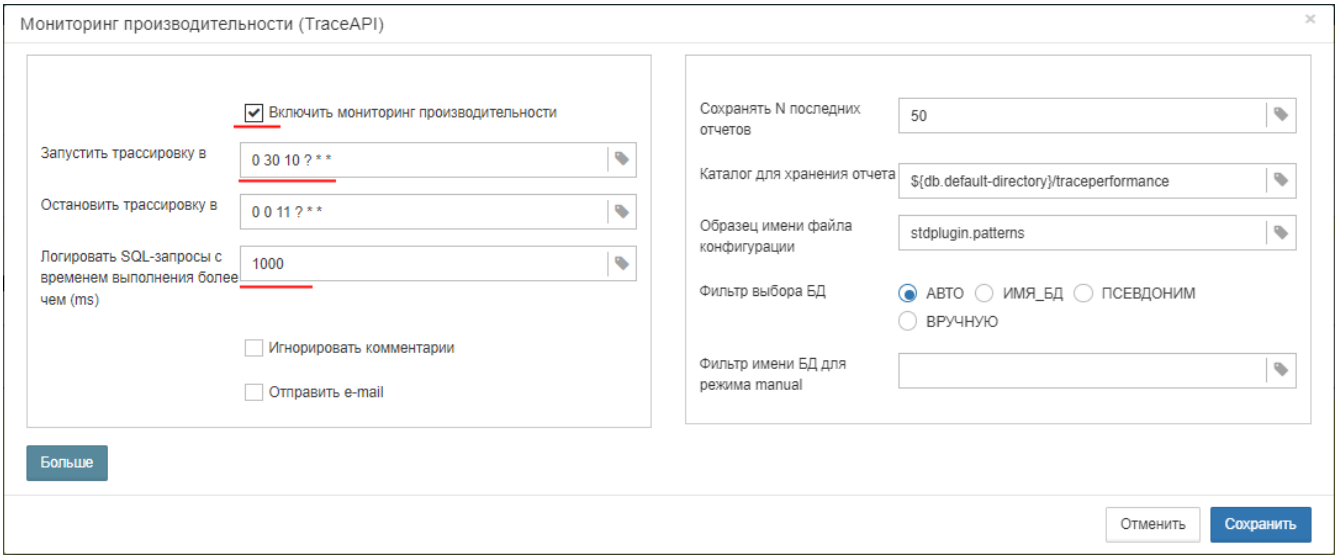
В HQBird 2020 представляет новый подход к автоматическому мониторингу производительности с помощью таблиц Firebird TraceAPI.

Теперь регулярную проверку работоспособности базы данных можно запланировать менее чем за 1 минуту.

Для этого просто откройте вкладку “Производительность” и настройте мониторинг транзакций и запросов:




Чтобы настроить Мониторинг производительности, укажите в диалоге его обязательные параметры:




Первый обязательный параметр — “**Включить мониторинг производительности**” — его необходимо включить для запуска трассировок по расписанию.

Следующие важные параметры — “Запустить трассировку в” и “Остановить трассировку в” Они содержат выражения CRON, которые определяют, когда начинается и заканчивается трассировка.

**По умолчанию трассировка начинается в 10:30 и заканчивается в 11:00**. Рекомендуется принять график отслеживания в соответствии с вашими потребностями. Ниже вы можете увидеть таблицу с некоторыми популярными вариантами.

| **Выражения CRON** | | **Описание** |
|---|---|---|
| **Старт** | **Завершение** | |
| 0 0 * ? * * | 0 10 * ?* * | Запускать трассировку каждый час с 0 до 10 минут |
| 0 0 8 ? * * | 0 0 17 ?* * | Запускать трассировку каждый день с 8:00 до 17:00 |
| 0 30 10,13,15 ? * * | 0 0 11,14,16 ? * * | Запускать трассировку каждый день с 10:30 до 11:00, с 13:30 до 14:00 и с 15:30 до 16:00 |

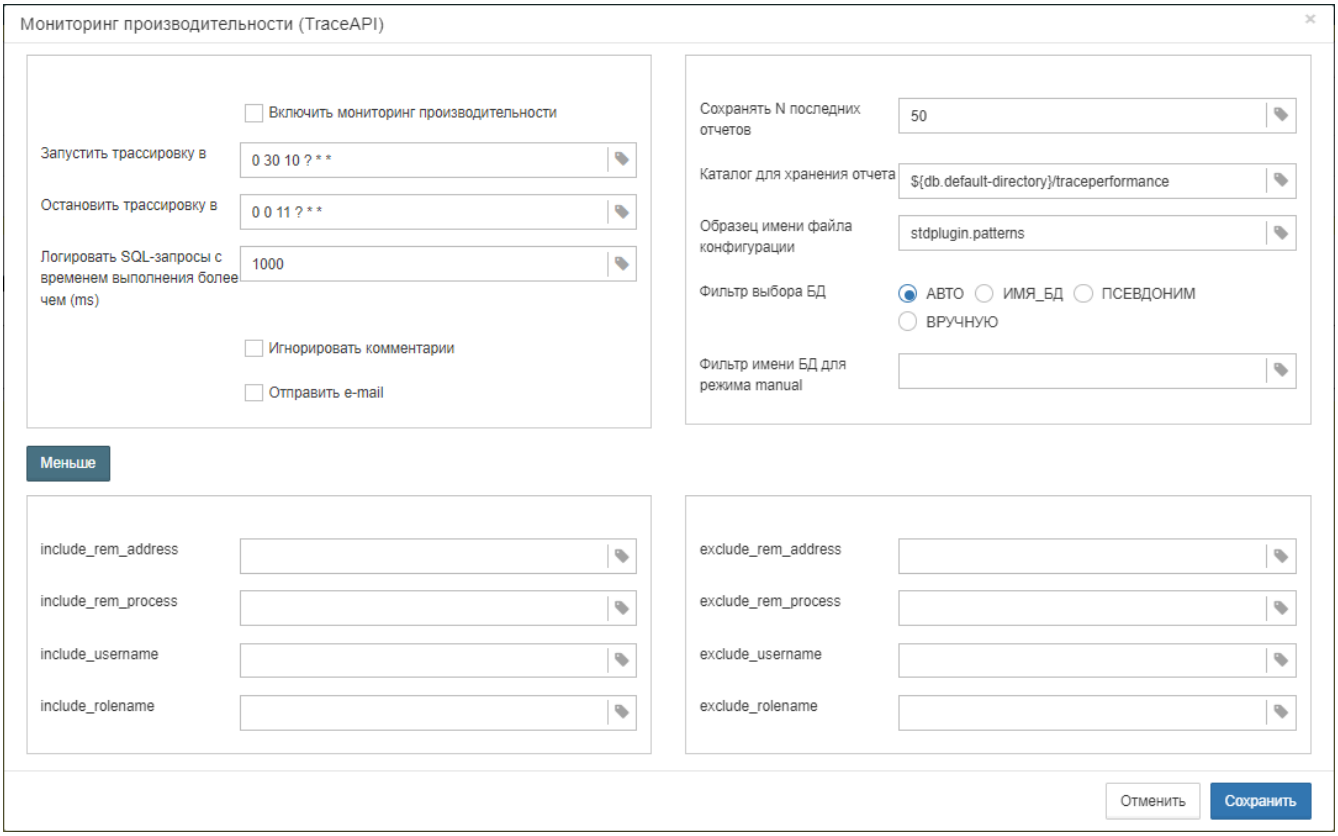
Следующий важный параметр — порог времени для медленных запросов, он задается в поле “Логировать SQL-запросы с временем выполнения более чем (ms)”. В этом поле необходимо установить порог времени выполнения запросов (в миллисекундах), после его превышения запросы будут сохраняться и анализироваться.

По умолчанию время установлено на 1000 миллисекунд или 1 секунду. Это означает, что будут регистрироваться и анализироваться только запросы, которые выполняются более 1 секунды.

Мы рекомендуем использовать базовое значение 1000 мс, пока ваша база данных не станет очень медленной: в этом случае 3000-5000 мс могут быть хорошим началом.

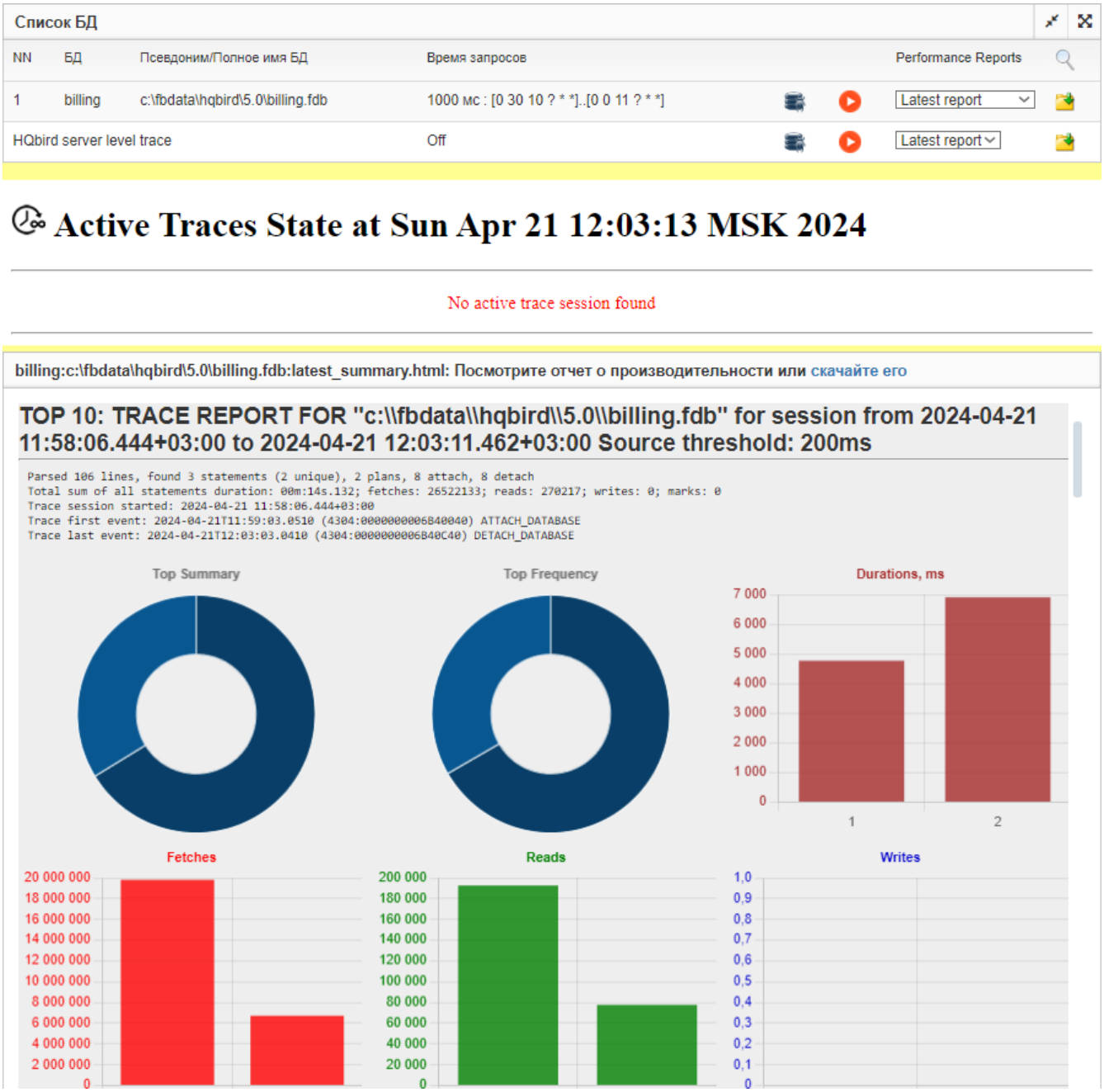
Флажок “Отправить e-mail” указывает на необходимость отправки отчета о производительности. Настройки электронной почты из конфигурации оповещений будут использоваться для отправки отчета о производительности.

Для более сложных настроек в диалоговом окне “Мониторинг производительности” есть дополнительные параметры (обычно их настраивать не требуется).




- “**Образец имени файла конфигурации**” — имя файла шаблона конфигурации, который следует использовать для настроек трассировки.
- “**Фильтр выбора БД**” — как должна идентифицироваться база данных. Обычно “АВТО” достаточно, он отслеживает указанную базу данных. В случае “ИМЯ_БД” или “ПСЕВДОНИМ” для фильтрации событий базы данных будет использоваться имя файла или псевдоним. “ВРУЧНУЮ” предоставляет возможность задать любое регулярное выражение, например, для трассировки нескольких баз данных или более одного псевдонима для одной базы данных.
- “**Database name filter**” — используется если выбран фильтр “MANUAL”.
- “**Сохранять N последних отчетов**” — указывает, сколько отчетов должно храниться в “Каталог для хранения отчета” для возможного ретроспективного использования.

В результате этого задания HQbird сформирует отчет о производительности, который будет сохранен в папке “Каталог для хранения отчета” в виде файла с расширением **html** и отправлен по электронной почте (в случае, если “Отправить e-mail” включен). Также самые последние отчёты доступны для просмотра и скачивания в интерфейсе HQBird.

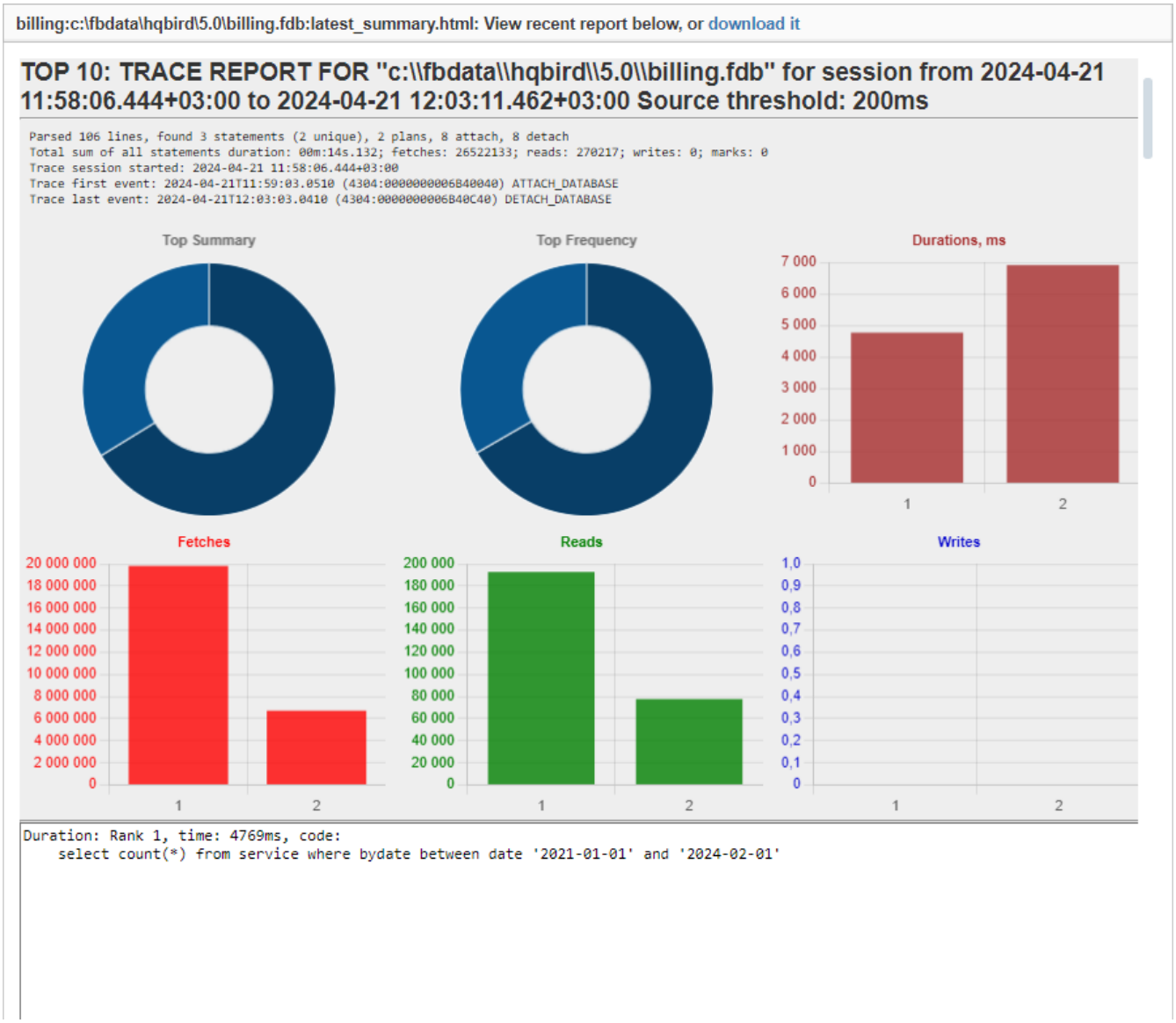


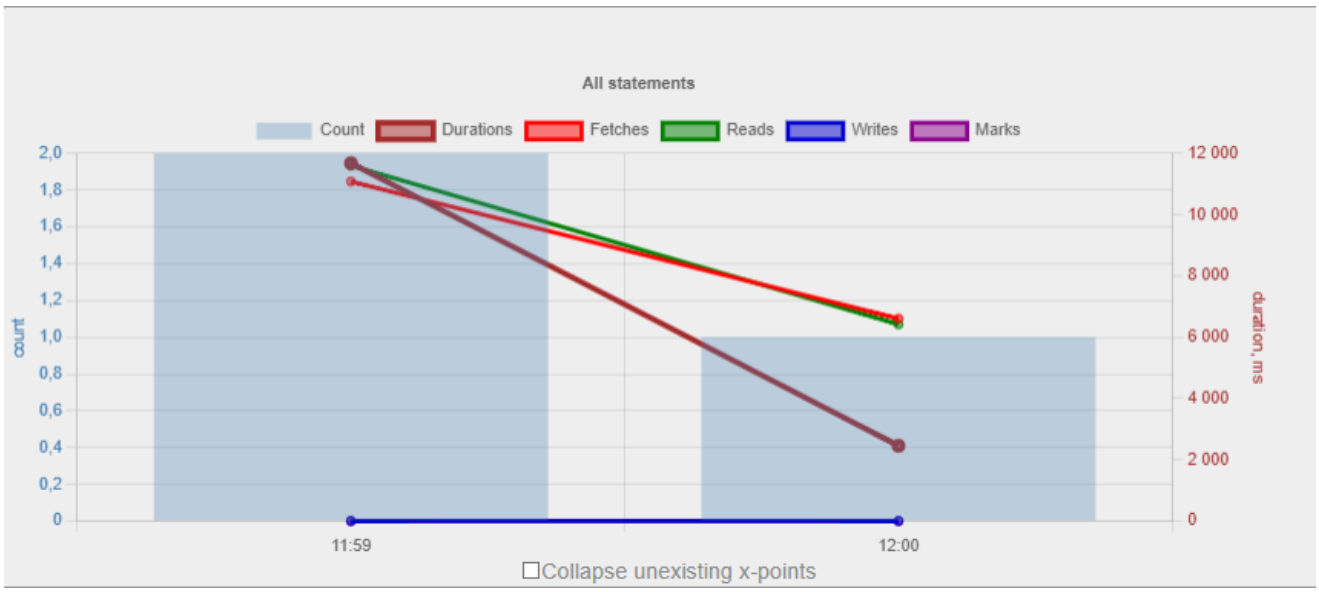

## 6.1.3. Что показывает отчёт о производительности?

Анализ производительности HQbird FBDataGuard предоставляет 3 типа отчетов:

1. список запросов, отсортированный по времени — “Sort by duration”;
2. список запросов, отсортированный по частоте — “Sort by frequency”;
3. список запросов, отсортированный по общему времени (т. е. суммарное время выполнения запросов с одинаковым текстом и различными параметрами) — “Sort by summary”.

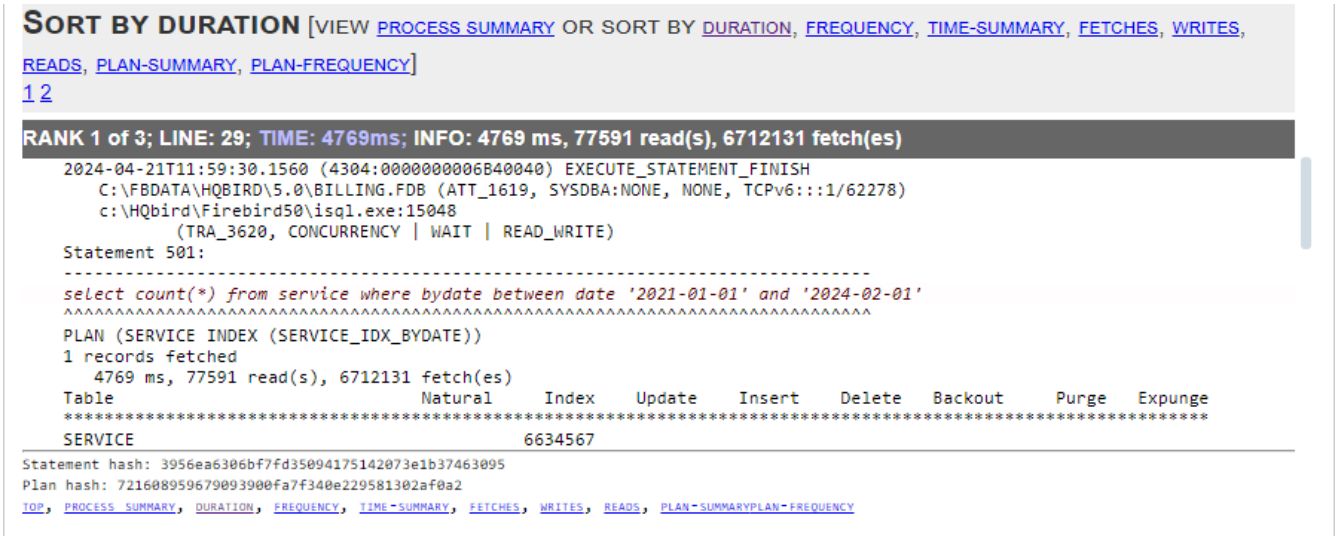
В начале отчета вы увидите графическое представление наиболее проблемных SQL-запросов:




Когда вы нажмете “Sort by duration” (это опция по умолчанию), вы увидите SQL-запросы и хранимые процедуры, выполнение которых в первую очередь заняло больше всего времени.

Обычно они возникают на длительных отчетах и других больших SQL-запросов.



**SORT BY DURATION** [VIEW PROCESS SUMMARY OR SORT BY DURATION, FREQUENCY, TIME-SUMMARY, FETCHES, WRITES, READS, PLAN-SUMMARY, PLAN-FREQUENCY]
1 2

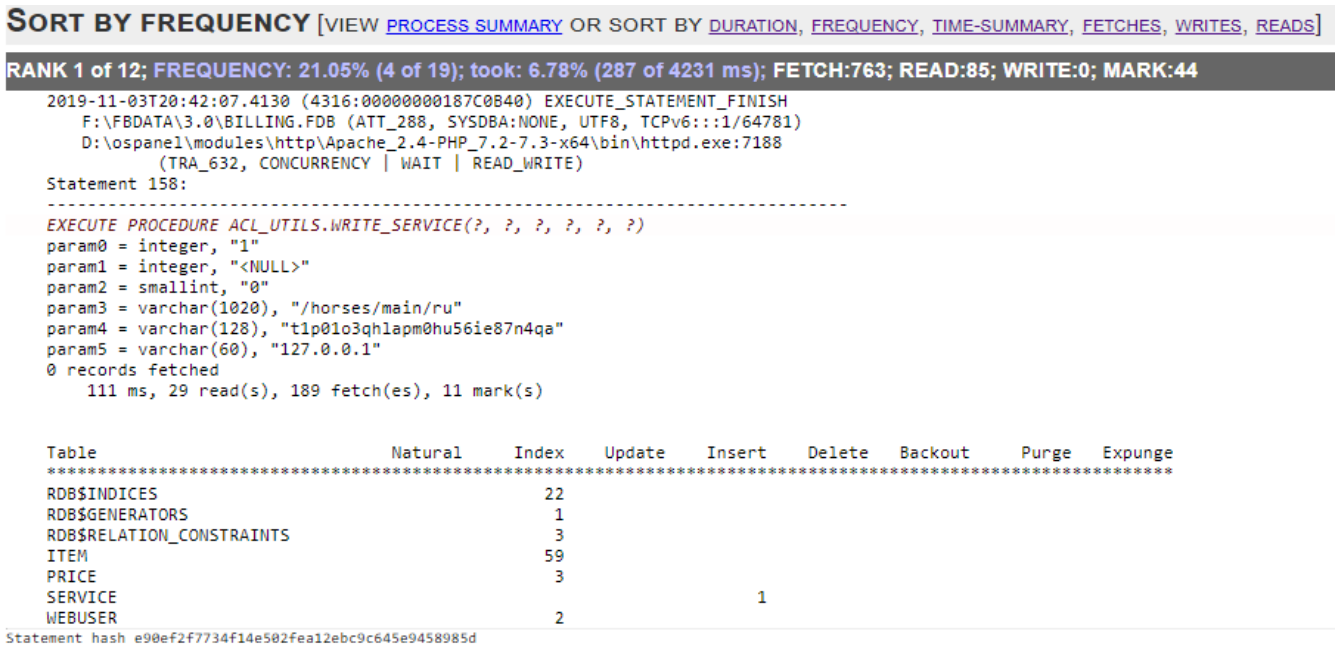
**RANK 1 of 3; LINE: 29; TIME: 4769ms; INFO: 4769 ms, 77591 read(s), 6712131 fetch(es)**

```
2024-04-21T11:59:30.1560 (4304:0000000006B40040) EXECUTE_STATEMENT_FINISH
   C:\FBDATA\HQBIRD\5.0\BILLING.FDB (ATT_1619, SYSDBA:NONE, NONE, TCPv6:::1/62278)
   c:\HQbird\Firebird50\isql.exe:15048
         (TRA_3620, CONCURRENCY | WAIT | READ_WRITE)
Statement 501:
-------------------------------------------------------------------------------
select count(*) from service where bydate between date '2021-01-01' and '2024-02-01'
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
PLAN (SERVICE INDEX (SERVICE_IDX_BYDATE))
1 records fetched
   4769 ms, 77591 read(s), 6712131 fetch(es)
Table                           Natural     Index    Update    Insert    Delete   Backout     Purge   Expunge
*******************************************************************************************************************
SERVICE                                     6634567
```

Statement hash: 3956ea6306bf7fd35094175142073e1b37463095
Plan hash: 721608959679093900fa7f340e229581302af0a2

TOP, PROCESS SUMMARY, DURATION, FREQUENCY, TIME-SUMMARY, FETCHES, WRITES, READS, PLAN-SUMMARYPLAN-FREQUENCY

При нажатии на ссылку “Sort by frequency” в шапке отчета вы увидите наиболее частые запросы: т.е. те запросы, которые запускались часто (среди зарегистрированных запросов).



**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**

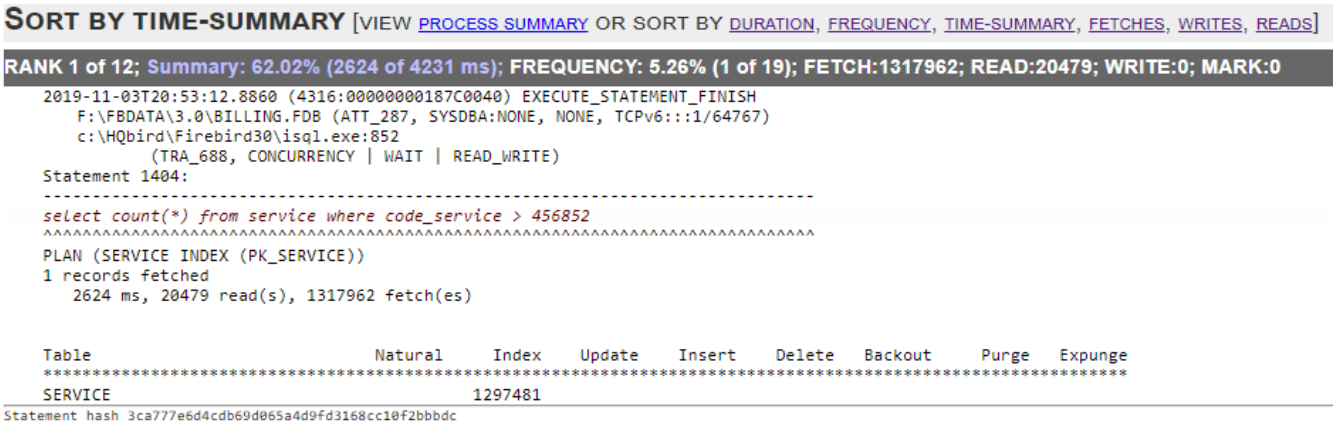
```
2019-11-03T20:42:07.4130 (4316:00000000187C0B40) EXECUTE_STATEMENT_FINISH
   F:\FBDATA\3.0\BILLING.FDB (ATT_288, SYSDBA:NONE, UTF8, TCPv6:::1/64781)
   D:\ospanel\modules\http\Apache_2.4-PHP_7.2-7.3-x64\bin\httpd.exe:7188
         (TRA_632, CONCURRENCY | WAIT | READ_WRITE)
Statement 158:
-------------------------------------------------------------------------------
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

Например, если процедура `SP_GET_INVOICE_REPORT` (или другой запрос) была выполнена 46 раз, то это означает, что данный запрос сильно влияет на общую производительность, и его следует оптимизировать в первую очередь.

При нажатии на “Sort by summary” вы увидите запросы, которые заняли большую часть времени (среди зарегистрированных запросов). Эти запросы обычно являются лучшими кандидатами на оптимизацию.




### Подробная информация по проблемным SQL-запросам.

Чтобы просмотреть детали наиболее частого запроса, нажмите ссылку “View details” внизу текста запроса:


В результате вы увидите самый длинный запрос среди запросов с одинаковым текстом SQL, с его планом выполнения, статистикой выполнения и входными параметрами.

Этой информации достаточно для анализа и оптимизации SQL-запроса в Firebird SQL Studio или другой IDE для разработчиков.

## 6.1.4. Как выбрать инструмент для детального мониторинга

FBDataGuard — это первая линия защиты базы данных Firebird; как только FBDataGuard обнаруживает что-то подозрительное внутри контролируемых областей, он отправляет предупреждение с описанием проблемы.

Если у вас есть несколько серверов Firebird, мы предлагаем приложение HQbird Control Center, которое собирает данные оповещений с серверов и баз данных Firebird и отображает их на одном экране. Свяжитесь с нами для получения более подробной информации.

После получения такого предупреждения от FBDataGuard администратор базы данных должен приступить к детальному изучению проблемы.

Выбор инструмента для детального мониторинга зависит от типа обнаруженной проблемы.

Если FBDataGuard сообщает о длительно выполняемой активной транзакции (Next-OAT), необходимо использовать **HQbird Mon$Logger** для обнаружения источника текущей активной транзакции.

Если сообщается о зависании oldest interesting транзакции, администратор базы данных должен запланировать явный sweep для очистки несобранного мусора с помощью задания Sweep FBDataGuard (если это необходимо), а затем запланировать отслеживание принудительных откатов с помощью Мониторинга производительности в FBDataGuard.

Если пользователи сообщают о проблеме медленности выполнения некоторых запросов, следует настроить Performance Monitoring или использовать Advanced Performance Monitoring.

Проблемы с общей производительностью базы данных и периодические или периодические замедления требуют анализа структуры базы данных, который можно выполнить только с помощью HQbird Database Analyst.

Ниже мы рассмотрим более подробно, как работать с инструментами мониторинга HQbird.

# 6.2. Мониторинг с помощью MON$ таблиц: HQbird MonLogger

HQbird MonLogger — это инструмент для анализа вывода таблиц мониторинга в Firebird, поиска проблем с медленными SQL-запросами, неправильно спроектированных транзакций (длительно выполняемых транзакций, транзакций с неправильным уровнем изоляции и т. д.), а также выявления проблемных приложений.

MonLogger может подключиться к базе данных Firebird с проблемами производительности и определить причину медленности: какое пользовательское подключение, медленный SQL-запрос или длительная транзакция?

MonLogger поддерживает Firebird 2.1, 2.5, 3.0, 4.0 и 5.0 — для более старых версий Firebird или InterBase используйте FBScanner (не входит в состав HQbird, приобретается отдельно).

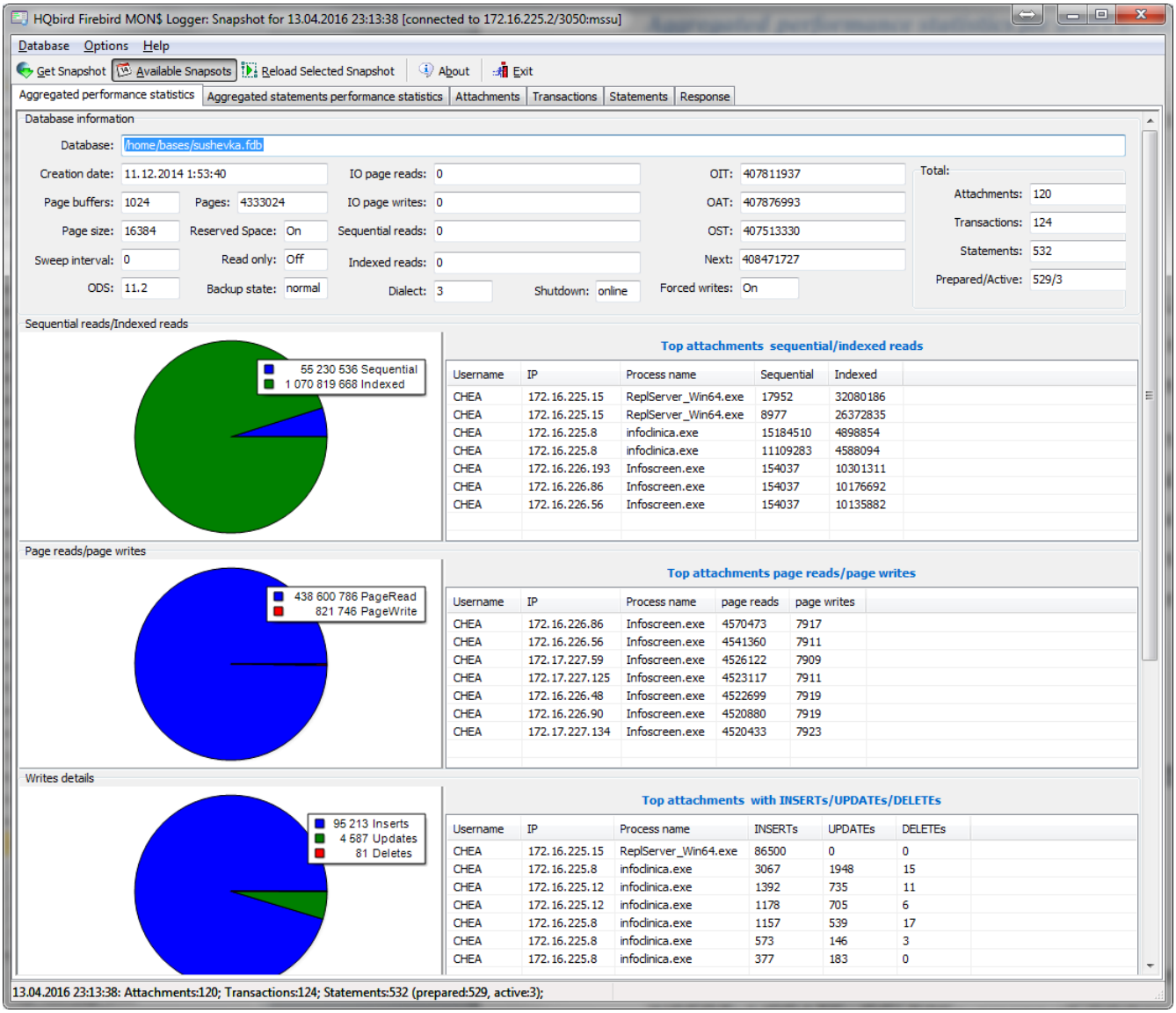
MonLogger может показать вам:

- Топ соединений с наибольшим количеством операций ввода-вывода, не индексированных и индексированных чтений
- Топ SQL запросов с наибольшим количеством операций ввода-вывода, не индексированных и индексированных чтений
- Проблемные транзакции: длительные транзакции, транзакции с ошибочным уровнем изоляции, транзакции чтения/записи и сопутствующая информация: когда они начались, какие приложения запустили эти транзакции, с какого IP-адреса и т. д.
- Соединения и запросы с наиболее интенсивными действиями по сборке мусора
- Соотношение Read/write, соотношение INSERT/UPDATE/DELETE и другое.

После подключения к базе данных, в которой вы хотите обнаружить проблемы с производительностью, необходимо сделать несколько снимков таблиц мониторинга — нажмите “Get Snapshot”, чтобы сделать снимок

## 6.2.1. Агрегированная статистика производительности для пользовательских соединений

На первом экране мы можем увидеть агрегированную статистику подключений к базе данных и выявить соединения с наибольшими проблемами:




## Sequential reads / Indexed reads

“Sequential reads / Indexed reads” показывает нам общее соотношение между последовательными (неиндексированными) чтениями и индексированными чтениями в приложении. Обычно количество неиндексированных чтений должно быть небольшим, поэтому большой процент последовательных чтений является признаком того, что многие SQL-запросы имею план выполнения `NATURAL`, и они могут быть причиной медленного времени ответа.

Щелкните на записи в разделе “TOP attachments: sequential/indexed reads”, чтобы перейти на вкладку “Attachments”, где можно просмотреть более подробную информацию о соединениях с БД, а затем перейти на вкладку “Transactions” или “Statements”, где вы увидите транзакции и запросы, связанные с выбранным соединением (если установлен флажок “Link to selected attachment”, в противном случае будут показаны все транзакции/запросы для всех соединений).

## Write details

“Write details” дает вам обзор операций записи: соотношение между INSERT/UPDATE/DELETE среди всех соединений базы данных. В таблице top writes вы можете увидеть соединения с

наибольшим количеством операций записи. Полезно выявить приложения или программные модули, которые выполняют чрезмерное количество обновлений или удалений (они являются наиболее опасными операциями с точки зрения сборки мусора).

#### Garbage collection details

Что означают операции по сборке мусора?

- Purge — движок удаляет бэк-версии, в базе данных находится только основная версия.
- Expunge — как основная версия, так и все бэк-версии были удалены.
- Back-out — удалена только основную версию (из-за rollback).

Обычно мы можем связать Purge с операцией `UPDATE`, Expunge с `DELETE`, и Backout с откатом `INSERT` или `UPDATE`. Множество backouts могут означать, что в приложении возникла проблема с управлением транзакциями.

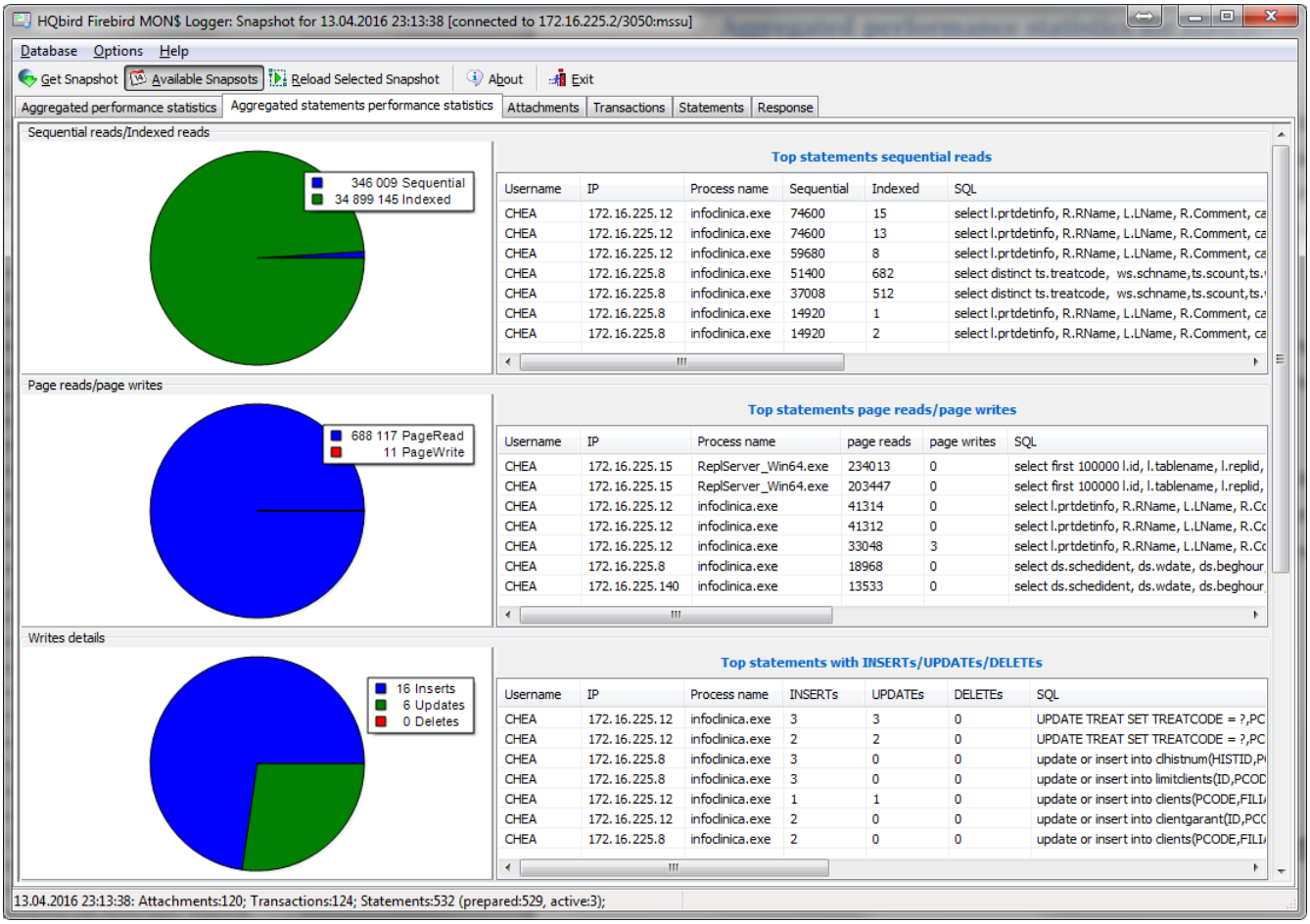
#### Memory usage

График “Memory usage” показывает общий объем памяти, используемый всеми активными соединениями сейчас, и пик выделенной памяти для них в прошлом.

Топ соединений по объему использования памяти показывает, какие ваши соединения больше всего потребляют память. Полезно найти приложения или программные модули с чрезмерным использованием памяти.

### 6.2.2. Aggregated performance statistics for statements

На второй вкладке вы можете найти агрегированную статистику производительности SQL запросов.




Эта статистика лучше отражает текущую ситуацию в базе данных — поскольку таблицы мониторинга собирают информацию с начала жизни каждого объекта, здесь вы можете увидеть операторы, которые выполнялись в момент создания моментального снимка.

### Sequential reads / Indexed reads

В этом списке мы видим топ запросов, которые выполняют множество операций последовательного чтения из базы данных. Обычно такие запросы требуют настройки SQL — либо путем создания индексов индексов, либо путем изменения конструкции SQL-запроса.

Чтобы настроить запрос, проверьте план его выполнения: обычно можно повысить скорость запроса, исключив из планов `NATURAL` с помощью новых индексов или переписывания запроса. Щёлкните по запросу в этом списке, чтобы открыть вкладку “Statements”, где вы сможете найти более подробную информацию о выбранном запросе и перейти к соответствующей транзакции или соединению.

### Page reads/page writes

На этих графиках и в списке представлена краткая информация о топе SQL запросов, которые выполняют много операций чтения — это означает, что они потребляют значительный объем операций ввода-вывода и могут повлиять на производительность других запросов. SQL запросы с пиковыми значениями следует тщательно проверять на предмет оптимальной производительности.

### Write details для запросов

На этом графике вы можете увидеть, что записывали SQL операторы в момент создания снимка таблиц мониторинга, а также определить UPDATEs и DELETEs, которые внесли много изменений в базу данных.

### Garbage collection details для запросов

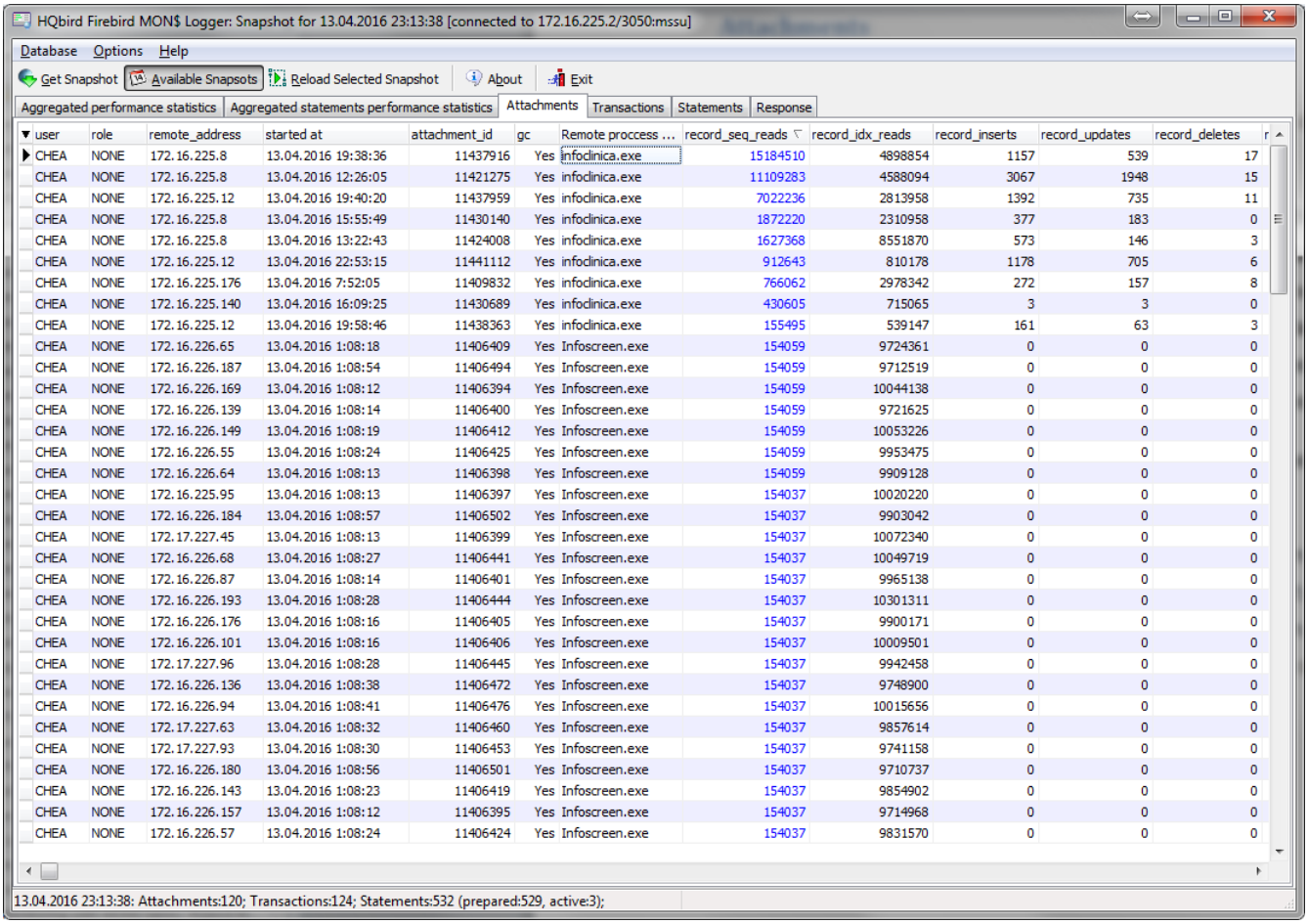
На этом графике мы видим, сколько операций по сборке мусора было выполнено SQL запросами, выполнявшимися в момент создания снимка.

### Memory usage для запросов

В отличие от агрегированной статистики использования памяти для соединений, использование памяти SQL запросами может показать нам список точных операторов, которые в данный момент потребляют много памяти.

## 6.2.3. Attachments

Третья вкладка — “Attachments”. Вы можете открыть эту вкладку и перейти туда, щелкнув одну из записей в разделе “Aggregated performance statistics”.




| user | role | remote_address | started at | attachment_id | gc | Remote proccess ... | record_seq_reads | record_idx_reads | record_inserts | record_updates | record_deletes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CHEA | NONE | 172.16.225.8 | 13.04.2016 19:38:36 | 11437916 | Yes | infoclinica.exe | 15184510 | 4898854 | 1157 | 539 | 17 |
| CHEA | NONE | 172.16.225.8 | 13.04.2016 12:26:05 | 11421275 | Yes | infoclinica.exe | 11109283 | 4588094 | 3067 | 1948 | 15 |
| CHEA | NONE | 172.16.225.12 | 13.04.2016 19:40:20 | 11437959 | Yes | infoclinica.exe | 7022236 | 2813958 | 1392 | 735 | 11 |
| CHEA | NONE | 172.16.225.8 | 13.04.2016 15:55:49 | 11430140 | Yes | infoclinica.exe | 1872220 | 2310958 | 377 | 183 | 0 |
| CHEA | NONE | 172.16.225.8 | 13.04.2016 13:22:43 | 11424008 | Yes | infoclinica.exe | 1627368 | 8551870 | 573 | 146 | 3 |
| CHEA | NONE | 172.16.225.12 | 13.04.2016 22:53:15 | 11441112 | Yes | infoclinica.exe | 912643 | 810178 | 1178 | 705 | 6 |
| CHEA | NONE | 172.16.225.176 | 13.04.2016 7:52:05 | 11409832 | Yes | infoclinica.exe | 766062 | 2978342 | 272 | 157 | 8 |
| CHEA | NONE | 172.16.225.140 | 13.04.2016 16:09:25 | 11430689 | Yes | infoclinica.exe | 430605 | 715065 | 3 | 3 | 0 |
| CHEA | NONE | 172.16.225.12 | 13.04.2016 19:58:46 | 11438363 | Yes | infoclinica.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 | 172.16.225.95 | 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” показывает список пользователей, подключенных к базе данных Firebird, со многими полезными подробностями: `USER` и `ROLE` для соединения, время начала и идентификатор соединения, включена ли сборка мусора для соединения, имя удаленного процесса, установившего соединение, и несколько накопленных счетчиков

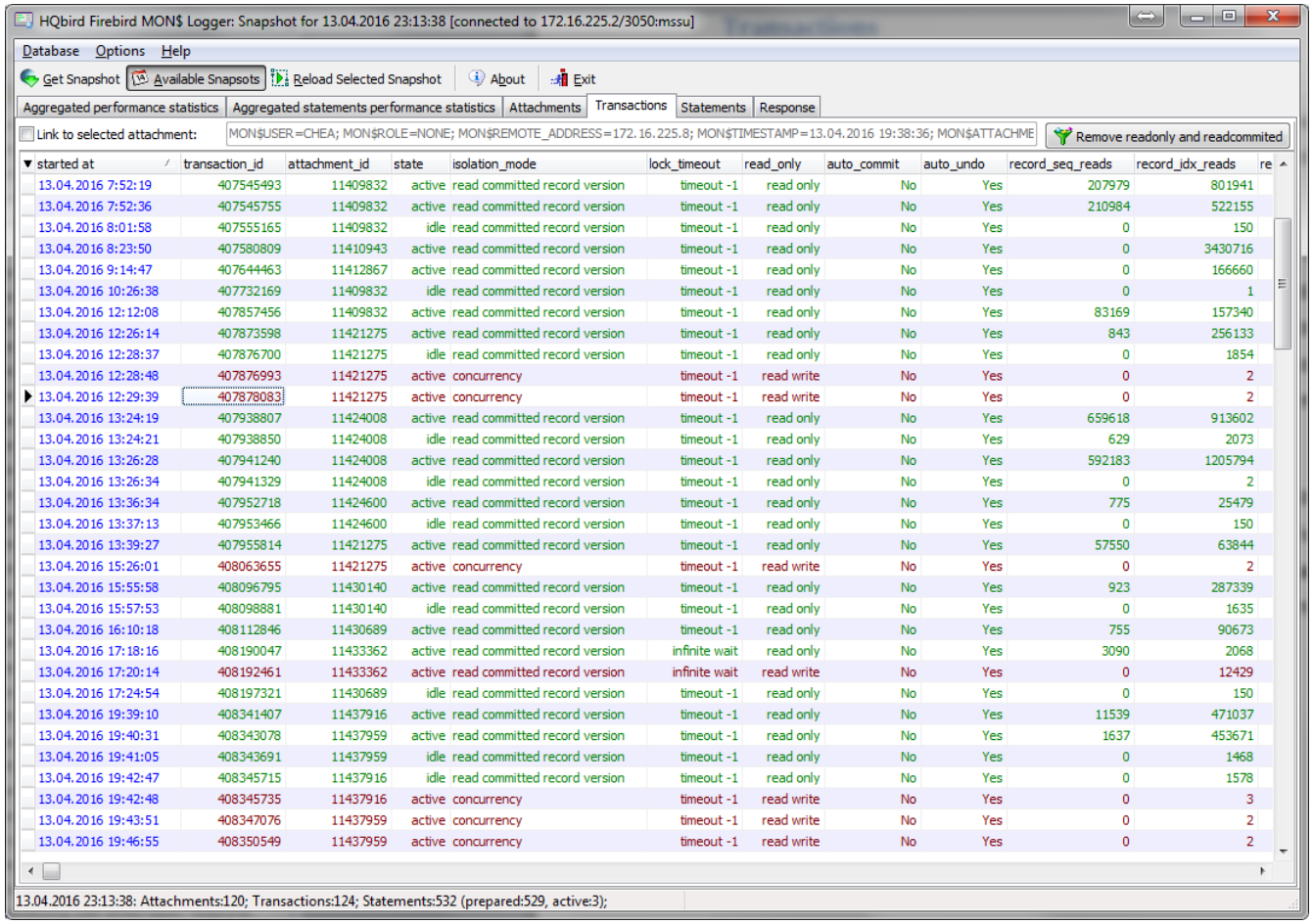
производительности для соединения: количество последовательных чтений (выполненных соединений с момента его запуска), количество индексированных чтений, количество вставок, обновлений и удалений, а также количество backouts, purges и expunges.

По умолчанию некоторые столбцы соединения отключены, чтобы отображалась только самая важная информация.

Конечно, каждый раз, когда вы щёлкаете по соединению, вы можете перейти к транзакциям, выполняемым внутри него, а затем к запросам. В левом верхнем углу вкладок “Transactions” и “Statements” есть флажок, который управляет поведением: если этот флажок установлен, будут отображаться только транзакции и запросы, по выделенному идентификатору соединения.

### 6.2.4. Transactions

Закладка “Transactions” показывает активные транзакции на момент создания снимка.



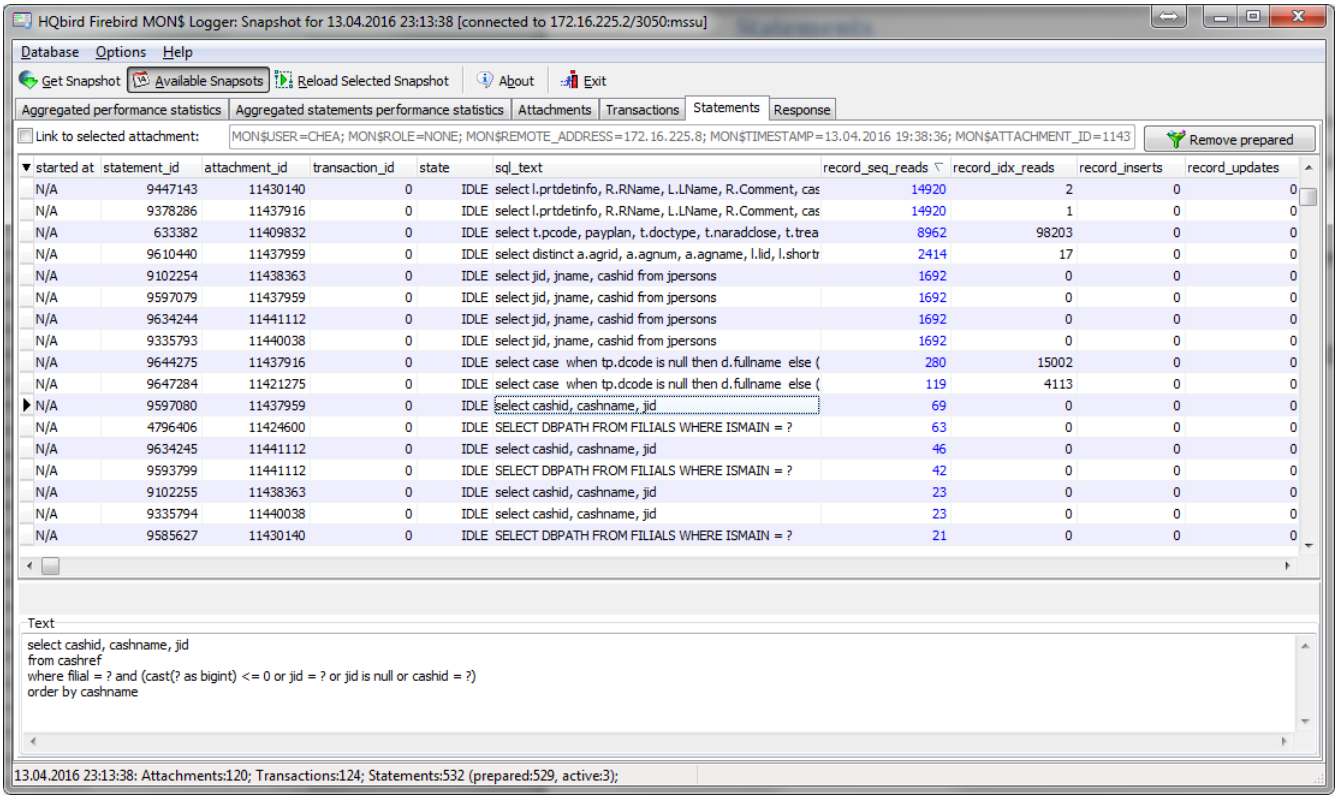

Если установлен флажок “Link to selected attachment” будут показаны только транзакции для выбранного соединения, в противном случае показаны все транзакции.

Одной из наиболее важных характеристик является время жизни транзакций: поскольку Firebird предназначен для работы с короткими пишущими транзакциями, важно сделать их как можно более короткими. MonLogger выделяет транзакции с режимами изоляции и read-write настройками, которые содержат Oldest Active транзакцию и, следовательно, приводят к тому, что они накапливают чрезмерное количество версий записей. Если вы видите такую транзакцию и она началась некоторое время назад, это означает, что она

может быть ответственна за чрезмерное накопление версий записей.

Выполните сортировку по столбцу “started at” и найдите старые транзакции, отмеченные красным: все транзакции, доступные для записи, и snapshot транзакции, доступные только для чтения, приводят к застреванию в Oldest Active Transaction и вызывают удержание чрезмерного количества версий записей. Определите, где начались эти транзакции (щелкните правой кнопкой мыши и выберите “View parent attachment”) и исправьте свой код, чтобы фиксировать эту транзакцию раньше.

### 6.2.5. Statements




На вкладке “Statements” показаны SQL операторы, активные на момент создания снимка: если вам нужно перехватить все операторы, то следует настроить Performance Monitoring или использовать Advanced Performance Monitoring.

Если включена опция “Link to selected attachment”, то будут показаны только SQL запросы для конкретного соединения, в противном случае в списке будут все активные запросы.

Некоторые операторы не имеют связанного идентификатора транзакции (=0): эти запросы подготавливаются, но не выполняются.

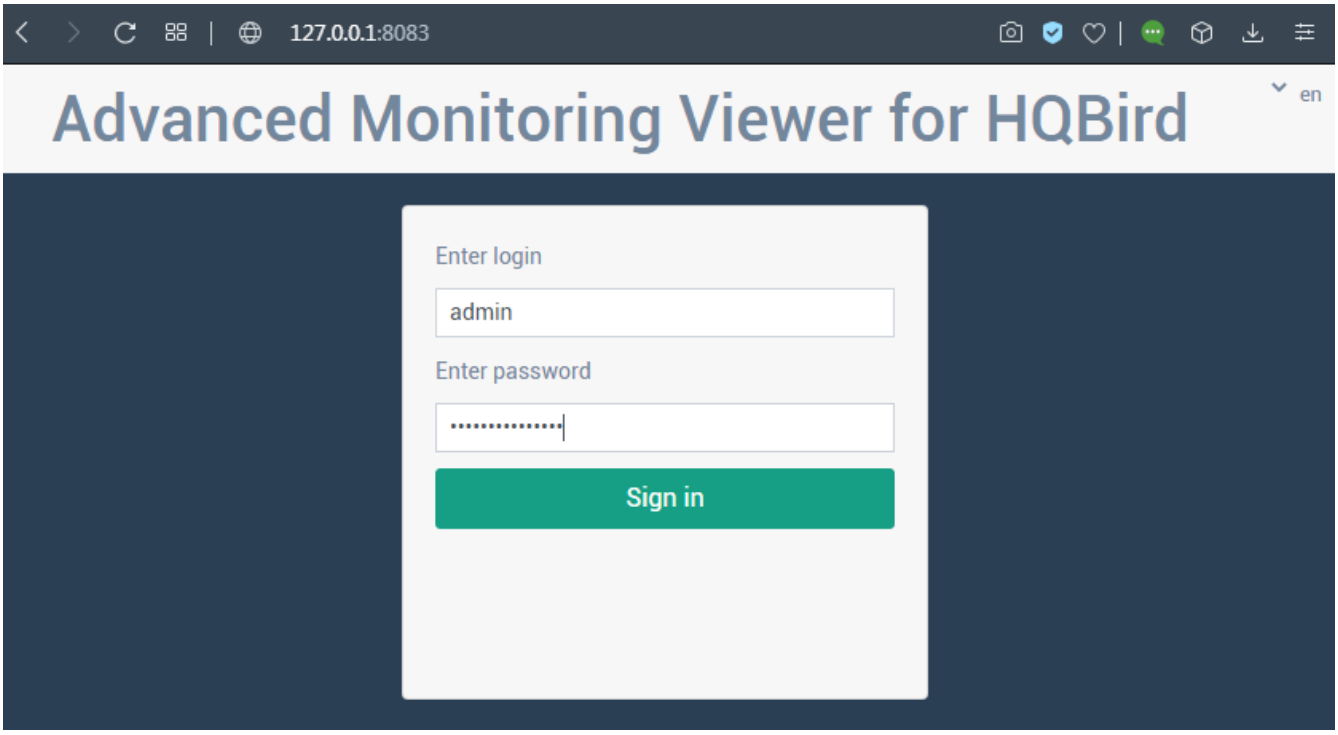
## 6.3. Advanced Monitor Viewer

Advanced Monitor Viewer позволяет графически отображать дополнительные счетчики производительности. Они основаны как на данных трассировки, так и на данных таблиц мониторинга, плюс используются дополнительные системные утилиты, такие как wmic (Windows).

Для запуска “Advanced Monitor Viewer” нажмите на соответствующий пункт меню “Пуск” **IBSurgeon › HQbird Server Side 2024 › Advanced Monitor Viewer** или запустите скрипт `AVM/quick_start.cmd`.

После успешного запуска в браузере по умолчанию откроется страница `http://127.0.0.1:8083`.

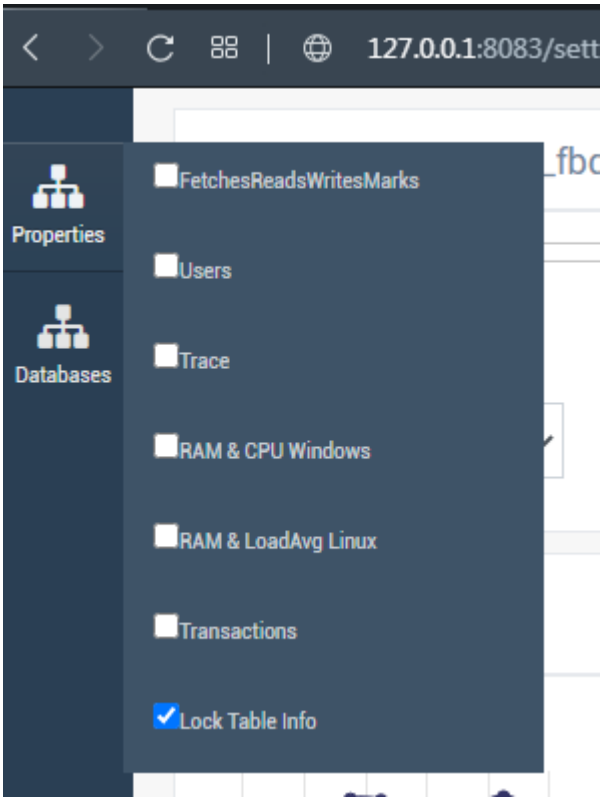
Вам будет предложено войти в систему:



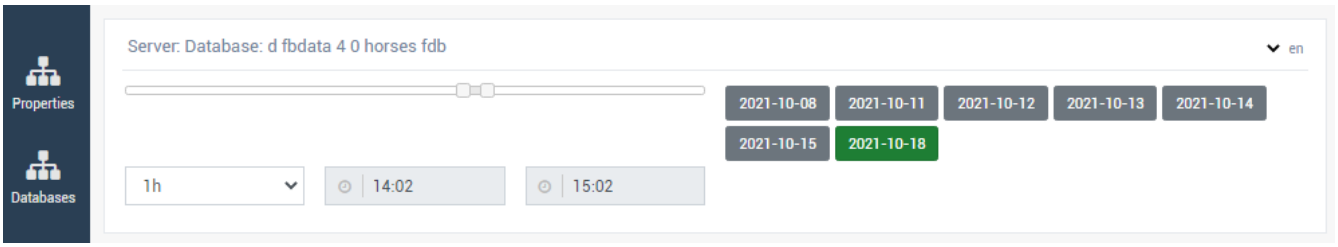

Логин и пароль по умолчанию такие же, как и для DataGuard: "admin / strong password".

После успешной аутентификации откроется страница с панелью, на которой расположены различные графики, отображающие загрузку системы в разные моменты времени.

В левой части страницы вы увидите две кнопки: “Properties” и “Databases”. Первая открывает контекстное меню для выбора счетчиков, которые будут отображаться на графиках. Вторая, открывает контекстное меню, в котором можно выбрать базу данных, для которой отображаются эти счетчики. База данных должна быть зарегистрирована для мониторинга с помощью DataGuard.



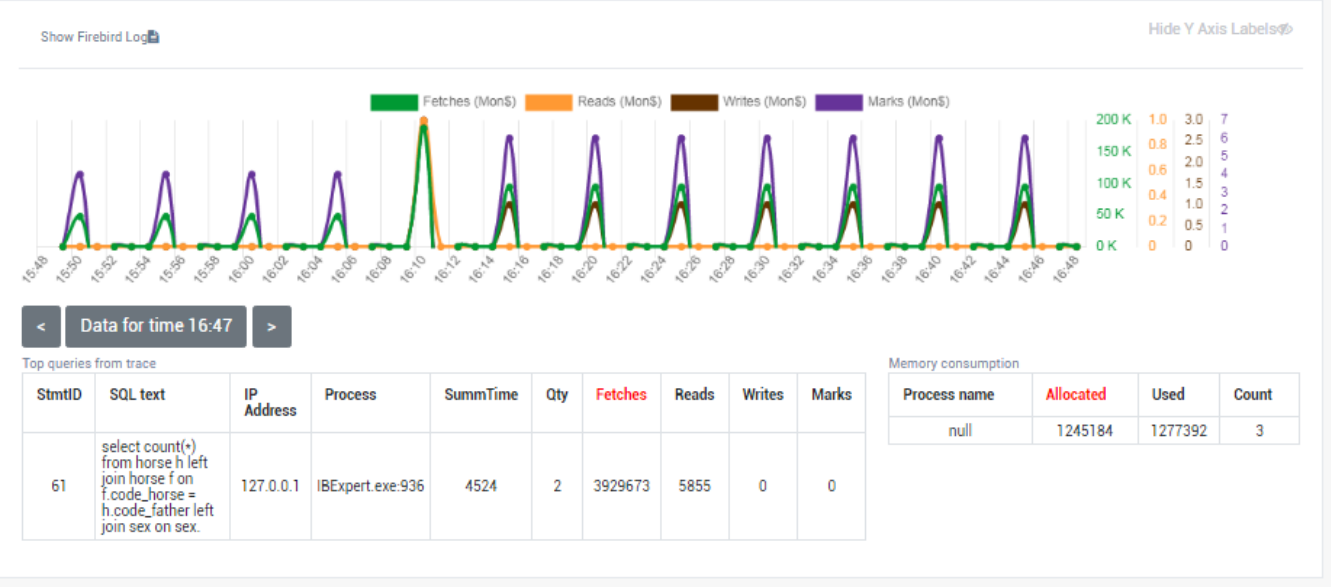

Вверху страницы отображается название базы данных, закладки с датами, а также интервал времени, за который отображаются счетчики производительности. Вы можете изменить дату просмотра и выбрать нужный интервал.




Следующие счетчики могут быть отображены графически:

### 6.3.1. Fetches, Reads, Writes, Marks

На графике отображаются счетчики производительности Fetches, Reads, Writes, Marks на основе таблиц мониторинга. Вы можете перейти к каждому моменту времени, щелкнув по нему или выбрав “Data for time” из списка.

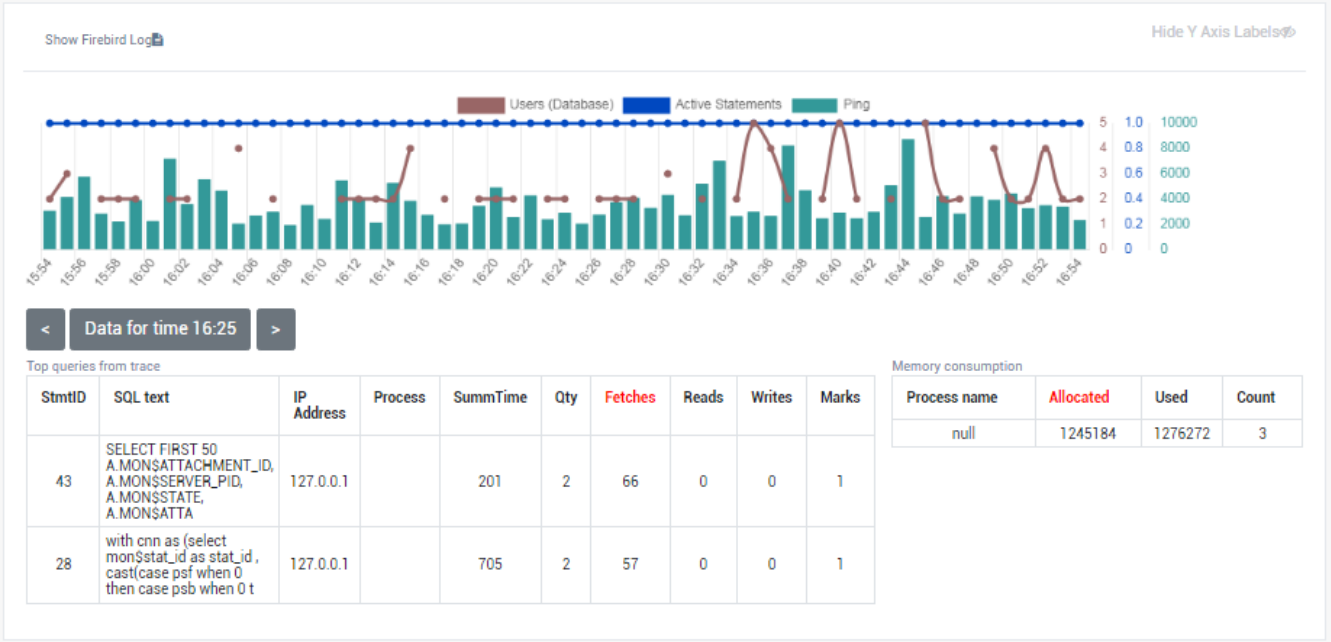



| StmtID | SQL text | IP Address | Process | SummTime | Qty | Fetches | Reads | Writes | Marks |
|---|---|---|---|---|---|---|---|---|---|
| 61 | select count(*) from horse h left join horse f on f.code_horse = h.code_father left join sex on sex. | 127.0.0.1 | IBExpert.exe:936 | 4524 | 2 | 3929673 | 5855 | 0 | 0 |

| Process name | Allocated | Used | Count |
|---|---|---|---|
| null | 1245184 | 1277392 | 3 |

## 6.3.2. Users

На графике отображается количество активных пользователей и запросов, а также время пинга. Вы можете перейти к каждому моменту времени, щелкнув по нему или выбрав “Data for time” из списка.




| StmtID | SQL text | IP Address | Process | SummTime | Qty | Fetches | Reads | Writes | Marks |
|---|---|---|---|---|---|---|---|---|---|
| 43 | SELECT FIRST 50 A.MON$ATTACHMENT_ID, A.MON$SERVER_PID, A.MON$STATE, A.MON$ATTA | 127.0.0.1 | | 201 | 2 | 66 | 0 | 0 | 1 |
| 28 | with cnn as (select mon$stat_id as stat_id , cast(case psf when 0 then case psb when 0 t | 127.0.0.1 | | 705 | 2 | 57 | 0 | 0 | 1 |

| Process name | Allocated | Used | Count |
|---|---|---|---|
| null | 1245184 | 1276272 | 3 |

## 6.3.3. Traces

На графике отображаются счетчики производительности: Fetches, Reads, Writes, Marks и время выполнения запроса на основе данных из журналов трассировки. Вы можете перейти к каждому моменту времени, щелкнув по нему или выбрав “Data for time” из списка.


Top queries from trace

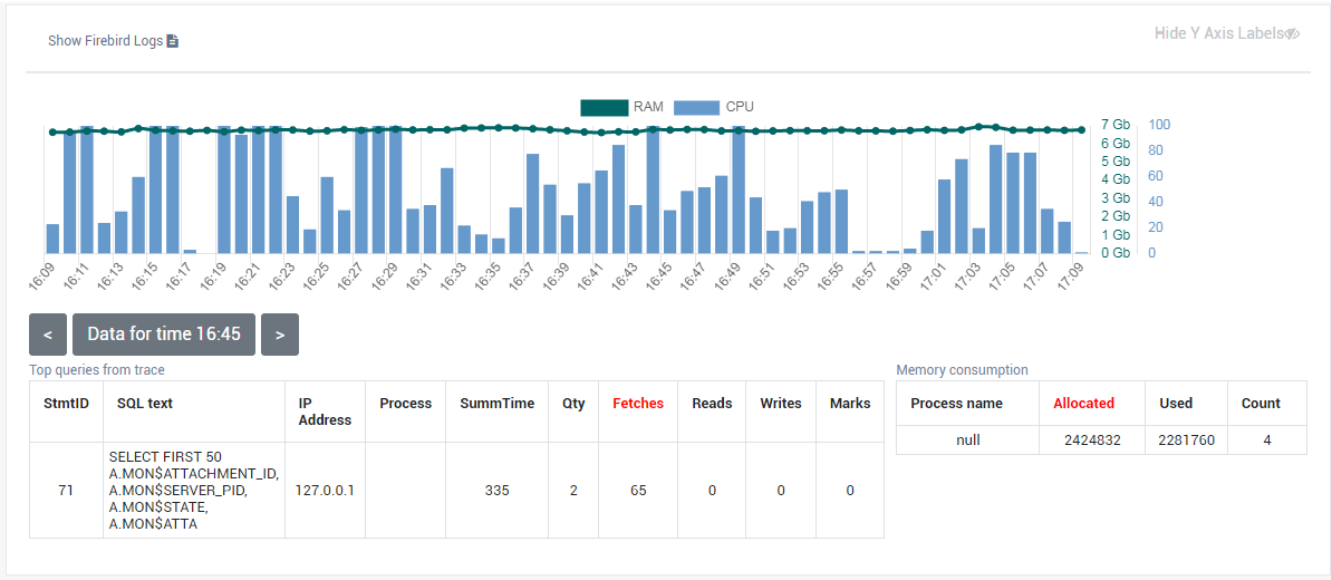
| StmtID | SQL text | IP Address | Process | SummTime | Qty | Fetches | Reads | Writes | Marks |
|---|---|---|---|---|---|---|---|---|---|
| 43 | SELECT FIRST 50 A.MON$ATTACHMENT_ID, A.MON$SERVER_PID, A.MON$STATE, A.MON$ATTA | 127.0.0.1 | | 201 | 2 | 66 | 0 | 0 | 1 |
| 28 | with cnn as (select mon$stat_id as stat_id , cast(case psf when 0 then case psb when 0 t | 127.0.0.1 | | 705 | 2 | 57 | 0 | 0 | 1 |

Memory consumption

| Process name | Allocated | Used | Count |
|---|---|---|---|
| null | 1245184 | 1276272 | 3 |

## 6.3.4. RAM and CPU Windows

На графике отображается потребляемая память, а также загрузка процессора на основе отслеживания утилитой wmic.




Top queries from trace

| StmtID | SQL text | IP Address | Process | SummTime | Qty | Fetches | Reads | Writes | Marks |
|---|---|---|---|---|---|---|---|---|---|
| 71 | SELECT FIRST 50 A.MON$ATTACHMENT_ID, A.MON$SERVER_PID, A.MON$STATE, A.MON$ATTA | 127.0.0.1 | | 335 | 2 | 65 | 0 | 0 | 0 |

Memory consumption

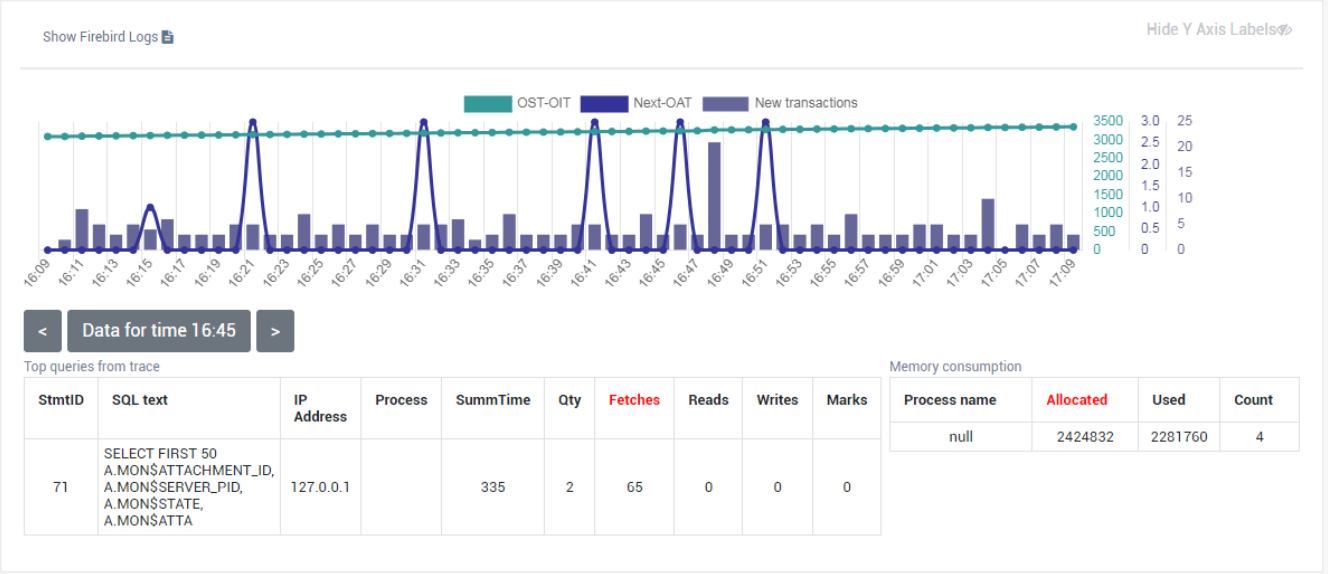
| Process name | Allocated | Used | Count |
|---|---|---|---|
| null | 2424832 | 2281760 | 4 |

## 6.3.5. RAM and LoadAvg Linux

То же, что “RAM and CPU Windows”, только в Linux.

## 6.3.6. Transactions

На графике отображается количество активных транзакций и разрыв между счетчиками OST-OIT, Next-OAT.

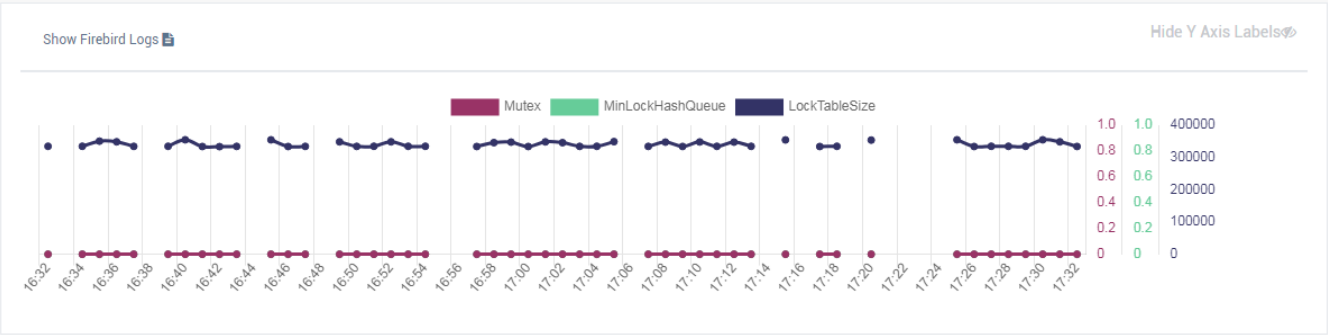



| StmtID | SQL text | IP Address | Process | SummTime | Qty | Fetches | Reads | Writes | Marks |
|---|---|---|---|---|---|---|---|---|---|
| 71 | SELECT FIRST 50 A.MON$ATTACHMENT_ID, A.MON$SERVER_PID, A.MON$STATE, A.MON$ATTA | 127.0.0.1 | | 335 | 2 | 65 | 0 | 0 | 0 |

| Process name | Allocated | Used | Count |
|---|---|---|---|
| null | 2424832 | 2281760 | 4 |

## 6.3.7. Lock Table Info

На графике отображаются данные по нагрузке на менеджер блокировок (актуально в Classic и SuperClassic).



# Глава 7. Анализ структуры базы данных

## 7.1. Обзор структуры базы данных Firebird

Первое, что следует сказать о структуре базы данных Firebird, это то, что она представляет собой набор страниц строго определенного размера: 4096, 8192, 16384 или 32768 (начиная с Firebird 4.0).

Страницы могут быть разных типов, каждая из которых служит определенной цели.

Страницы одного типа не идут строго одна за другой — они могут быть перемешаны, и размещены в файле в том порядке, в котором они были созданы сервером при расширении или создании базы данных.

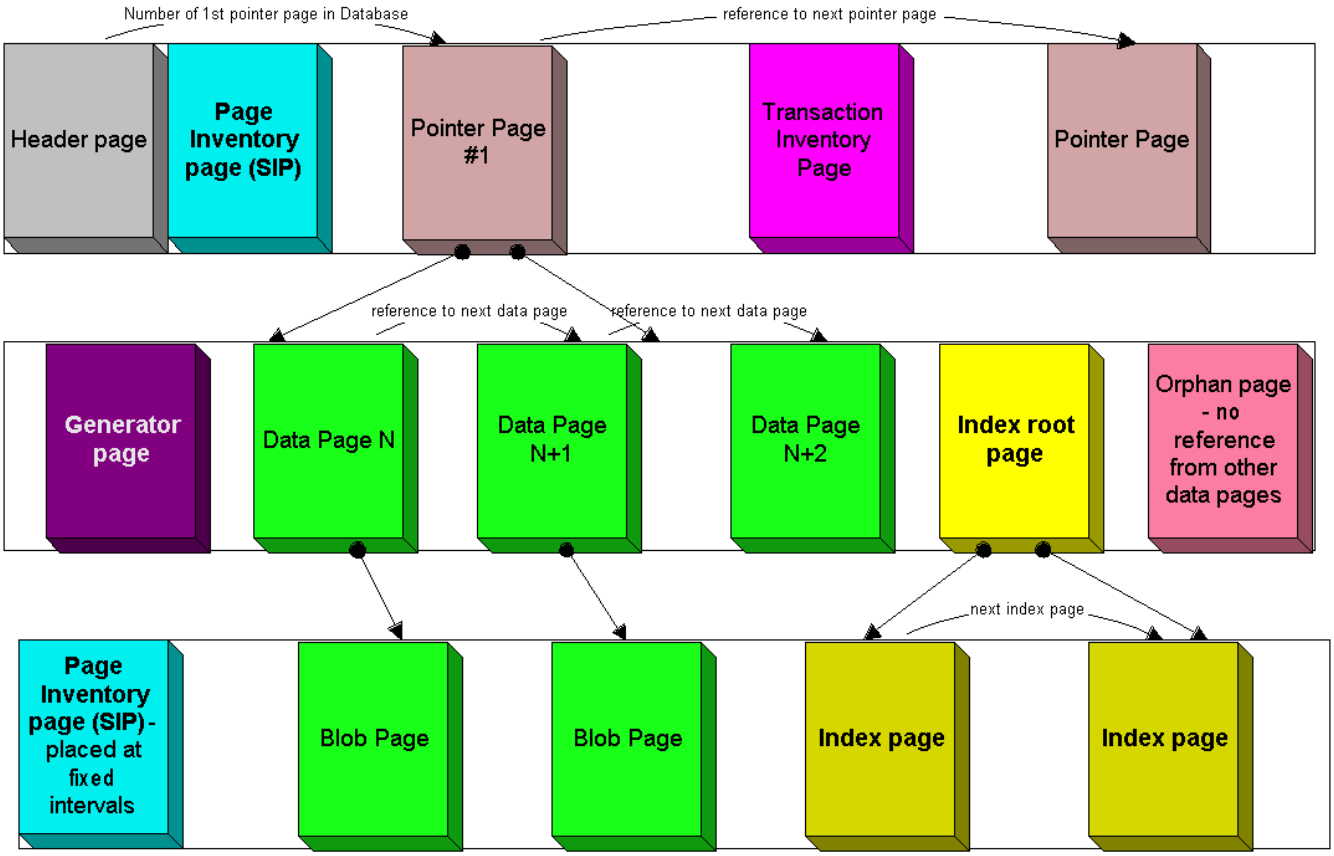



*Таблица 2. Типы страниц*

| Тип страницы | ID | Описание |
|---|---|---|
| **pag_undefined** | 0 | Undefined — Если страница имеет этот тип страницы, скорее всего, она пуста. |
| **pag_header** | 1 | Database header page |
| **pag_pages** | 2 | Page inventory page (or Space inventory page — SIP) |
| **pag_transactions** | 3 | Transaction inventory page (TIP) |
| **pag_pointer** | 4 | Pointer page |

| Тип страницы | ID | Описание |
|---|---|---|
| **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 |

# 7.2. Как анализировать структуру базы данных с помощью HQbird Database Analyst (IBAnalyst)

**IBAnalyst** — это инструмент, который помогает пользователю детально анализировать статистику базы данных Firebird и выявлять возможные проблемы с производительностью базы данных, ее обслуживанием и взаимодействием приложения с базой данных.

IBAnalyst графически отображает статистику базы данных Firebird в удобной для пользователя форме и выявляет следующие проблемы:

- фрагментирование таблиц и BLOB,
- версии записей,
- сборка мусора,
- эффективность индексов

Более того, IBAnalyst может автоматически вносить разумные предложения по улучшению производительности и обслуживанию базы данных.

IBAnalyst может получать статистику из действующих производственных баз данных через Services API (рекомендуется) или анализировать текстовый вывод команд `gstat -a -r`. Статистика в периоды пиковой нагрузки может дать много информации о реальных проблемах с производительностью в производственных базах данных.

Основные возможности IBAnalyst перечислены ниже:

- Получение статистики базы данных через Service API и из вывода `gstat`.
- Сводка всех актуальных и возможных проблем в базе данных
- Цветная сетка для представления таблиц, индексов и таблиц→индексов, которая выделяет фрагментированные таблицы, плохие индексы и т. д.
- Автоматическая экспертиза статистики базы данных предоставляет рекомендации и инструкции по следующим вопросам:
  - Оптимальный размер страницы базы данных
  - Состояние транзакций и критический разрыв между транзакциями.
  - Различные флаги базы данных
  - Глубина индекса
  - Дубликаты ключей индекса
  - Фрагментированные таблицы
  - Версии записи
  - Очень большие таблицы
- и другое

## 7.2.1. Как правильно получать статистику из базы данных Firebird

### Правильное время и место

Звучит странно, но просто взять статистику через gstat или Services API недостаточно. Статистику необходимо собирать в нужный момент, чтобы показать, как приложения влияют на данные и транзакции в базе данных.

Худшее время для сбора статистики

- Сразу после восстановления
- После снятия резервной копии (`gbak -b db.gdb`) без указания переключателя `-g`
- После ручного sweep (`gfix -sweep`)

Также верно, что во время работы могут быть моменты, когда база данных находится в правильном состоянии, например, когда приложения нагружают базу данных меньше, чем обычно (пользователи обедают, ужинают или в определенное время бизнес-процесса).

Как обнаружить, что в базе данных что-то не так?

Да, ваши приложения могут быть спроектированы настолько идеально, что они всегда будут корректно работать с транзакциями и данными, не создавая sweep gap, большого количества активных транзакций, длительных снимков и так далее. Обычно этого не происходит. Хотя бы потому, что некоторые разработчики тестируют свои приложения под управлением 2-3 пользователей одновременно, не более. Таким образом, при настройке написанных приложений для 15 и более одновременных пользователей база данных может вести себя непредсказуемо. Конечно, многопользовательский режим может работать нормально, поскольку большинство многопользовательских конфликтов можно протестировать с помощью 2-3 одновременно работающих приложений. Но затем, когда будет запускаться больше одновременно работающих приложений, могут возникнуть проблемы со сборкой мусора (по крайней мере). И это можно поймать, если в правильные моменты снимать статистику.

### Если у вас не возникают периодические проблемы с производительностью

Это может произойти, если ваши приложения спроектированы правильно, база данных не загружена или ваше оборудование современное и очень мощное (достаточно для хорошей обработки текущего количества пользователей и данных).

Самая ценная информация — это транзакционная нагрузка и накопление версий. Это можно увидеть, только если настроить регулярное сохранение статистики.

Лучшая настройка — получать почасовую статистику транзакций. Это можно сделать, запустив

```
gstat -h db.gdb > db_stat_<time>.txt
```

где

- `db.gdb` — имя вашей базы данных,
- `db_stat_<time>.txt` — текстовый файл, в котором будет сохраняться статистика,
- `<time>` — текущая дата и время, когда была снята статистика.

Также вы можете запланировать сбор статистики базы данных с помощью HQbird FBDataGuard, задание “Database: Statistics”.

### Если у вас периодически возникают проблемы с производительностью

Эти проблемы обычно вызваны автоматическим запуском sweep. Сначала вам нужно определить период времени между такими падениями производительности. Далее разделите этот интервал минимум на 4 (8, 16 и так далее). Сейчас в информационных системах много одновременно работающих пользователей, и большинство проблем с производительностью при ненастроенных сервере и базе данных происходят по 2 или 3 раза в день.

Например, если проблемы с производительностью возникают каждые 3 часа, вам необходимо собирать

```
gstat -h db.gdb
```

статистику каждые 30-45 минут, и

```
gstat -a -r db.gdb -user SYSDBA -pass masterkey
```

каждые 1-1.5 часа.

Лучше всего, когда вы соберёте статистику `gstat -a -r` прямо перед предстоящим падением производительности. Она покажет, где настоящий мусор и сколько скопилось устаревших версий записей.

### Что делать с этой статистикой?

Если ваше приложение явно использует транзакции и использует их хорошо, т. е. вы знаете, что такое `read read_commit` и когда его использовать, ваши SNAPSHOT транзакции длятся не дольше, чем необходимо, и транзакции активны минимальный период времени, вы можете настроить sweep interval, или отключить его, а затем заботьтесь только о том, сколько обновлений делает приложение(я) и какие таблицы нужно меньше обновлять или заботиться об обновлениях.

Что это значит, спросите вы? Приведем пример некоторой системы, в которой каждое утро в течение 20-30 минут возникали проблемы с производительностью. Этого вполне обычно для “утренних” запусков приложений, и дольше оно не может продолжаться.

Администратору базы данных задали правильные вопросы, и вот картина:

Ежедневная работа была разделена на части: утром анализ данных, затем данные

вставляются и редактируются обычными операторами, а в конце дня специальные процедуры начинают сбор данных, которые будут использоваться для аналитики на следующий день (как минимум).

Последняя работа над базой данных в конце дня заключалась в большом количестве обновлений, причем обновлений тех таблиц, которые аналитики использовали утром. Итак, появилось много мусорных версий, которые начало собирать приложение, работающее с утра.

Решение это проблемы оказалось простым — запускать `gfix -sweep` в конце дня.

Sweep читает все таблицы в базе данных и пытается собрать в них все мусорные версии для зафиксированных и откаченных транзакций. После очистки базы данных стало ясно, почему улучшение происходит после восстановления.

И “утренняя проблема” ушла.

Итак, вам нужно понимать статистику с учетом множества других факторов:

- сколько одновременно пользователей (в среднем) работает в течение дня
- продолжительность рабочего дня (8, 12, 16, 24 часа)
- какие приложения работают в разное время суток и как они влияют на данные, используемые другими приложениями, работающими в то же время или в последующее время. Т.е. вы должны понимать бизнес-процессы, происходящие в течение всего дня и всей недели.

### Когда DBA не может ничего сделать

К сожалению, такие ситуации случаются. И снова пример:

Некоторая система установлена примерно для 15 пользователей. Периодически производительность настолько падает, что администратору базы данных приходится перезагружать сервер. После перезагрузки сервера какое-то время все работает нормально, затем производительность снова падает. Статистика показала, что среднее количество транзакций в день составляет около 75000, и есть активные транзакции, выполняемые с начала дня до момента падения производительности.

К сожалению, приложения были написаны с использованием BDE и вообще без использования транзакций; то есть вся обработка транзакций была автоматической и использовалась самим BDE. Из-за этого некоторые транзакции оставались активными в течение длительного времени, а мусор (версии записей) накапливался до тех пор, пока администратор базы данных не перезапустил сервер. После перезапуска начнется автоматическая очистка, и мусор будет собран (устранен).

Все это было вызвано приложениями, потому что они тестировались только с 2-3 одновременными пользователями, а когда их стало ~15, приложения начали очень сильно нагружаться.

Надо сказать, что в этой конфигурации 70% пользователей только читали данные, а остальные 30% вставляли и обновляли какие-то (!) данные.

В этой ситуации единственное, что может улучшить производительность, — это перепроектировать управление транзакциями в этом приложении.

#### Как IBAnalyst может помочь найти проблемы в вашей базе данных Firebird

Давайте рассмотрим ключевые возможности IBAnalyst. Когда вы впервые просматриваете статистику своей базы данных в IBAnalyst, все может быть неясно, особенно если IBAnalyst показывает множество предупреждений в виде окрашенных в красный и желтый цвет ячеек в представлениях Summary, Tables и Index. Рассмотрим несколько примеров из реальной статистики.

### 7.2.2. Закладка Summary

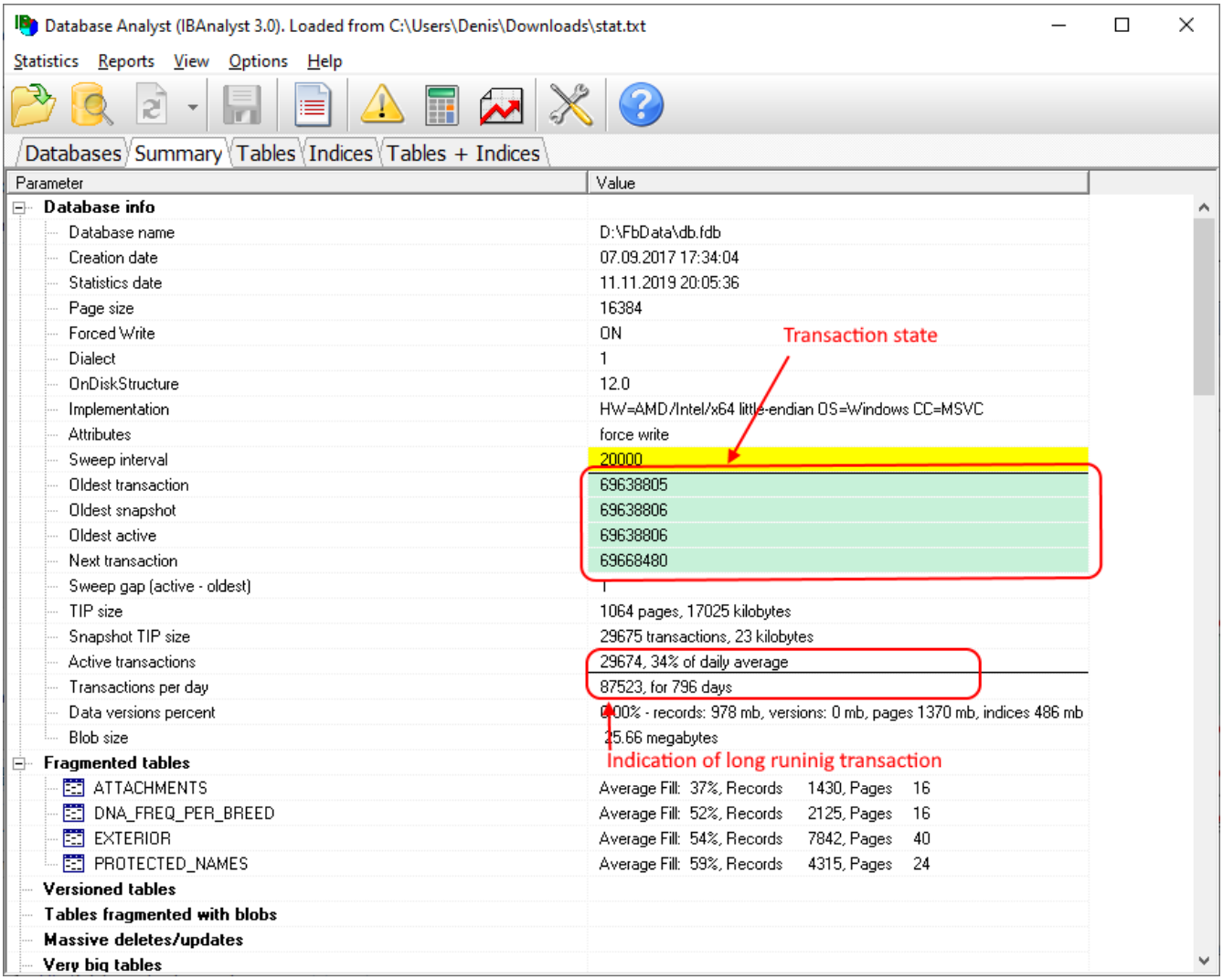
Summary содержит наиболее важную информацию, извлеченную из статистики базы данных. Обычно полная статистика базы данных содержит сотни Кбайт и распознать важную информацию непросто.

Ниже приведено описание объектов и параметров базы данных, которые вы можете увидеть в разделе Summary. Описание видимых проблем (отмеченных **красным** или **желтым**) смотри в подсказках для столбцов или выводе рекомендаций.

| Объект или параметр | Описание |
|---|---|
| **Database name** | Имя анализируемой базы данных. |
| **Creation date** | Дата создания базы данных. Когда она был создан оператором `CREATE DATABASE` или восстановлена (`gbak -c` или `gbak -r`). |
| **Statistics date** | Когда была получена статистика — дата файла статистики или дата вызова service API (сейчас). |
| **Page size** | Размер страницы — это физический параметр базы данных. В современных версиях Firebird размер страницы может быть 4096, 8192 или 16384 байта (Firebird *4.0+ может использовать размер страницы 32 КБ*). Для повышения производительности восстановите базу данных из резервной копии, используя размер страницы 8 или 16 КБ. |
| **Forced Write** | Показывает режим записи измененных страниц: синхронизированный или асинхронный — соответствующая настройка ON или OFF. OFF не рекомендуется, так как сбой сервера, сбой питания или другие проблемы могут привести к повреждению базы данных. |
| **Dialect** | Текущий диалект базы данных. |
| **Sweep interval** | Текущее значение sweep interval. Отмечено желтым, если оно не равно 0, и отмечено красным, если sweep gap больше sweep interval. |

| Объект или параметр | Описание |
|---|---|
| **On Disk Structure** | ODS. Это физический формат базы данных. Смотри подсказку, чтобы узнать номер ODS для конкретных версий IB/FB. |
| **Transaction block** | |
| **Oldest transaction** | Oldest interesting transaction. Идентификатор старейшей транзакции которая откачена или находится в состоянии limbo. |
| **Oldest snapshot** | Oldest snapshot transaction Идентификатор транзакции, которая была самой старой активной на момент запуска самого старого моментального снимка (snapshot). |
| **Oldest active** | Oldest active transaction Идентификатор самой старой все еще активной транзакции. |
| **Next transaction** | Идентификатор следующей транзакции. |
| **Sweep gap (snapshot - oldest)** | Для ODS 10.x. Разница между Oldest Snapshot и Oldest Interesting transaction. Если она больше sweep interval и sweep interval > 0, то Firebird попытается запустить sweep, и это может замедлить производительность. |
| **Snapshot gap (active - oldest)** | Разница между Oldest Active и Oldest transaction. То же самое что и sweep gap. |
| **TIP size** | Размер Transaction Inventory Pages, в страницах и килобайтах. TIP сохраняет состояние транзакции для каждой транзакции, запущенной с момента создания (или восстановления) базы данных. Он рассчитывается как Next transaction div 4 (байты). |
| **Snapshot TIP Size** | Размер Transaction Inventory Pages необходимых для snapshot транзакций. Отображает сколько памяти потребуется каждой snapshot транзакции для проверки состояния параллельных транзакций. |
| **Active transactions** | Количество активных транзакций (на момент взятия статистики из базы данных) `Next - Oldest Active`. Это неточная оценка, потому что может быть одна активная транзакция и множества завершённых после неё. В любом случае активные транзакции предотвращают сборку мусора. |
| **Transactions per day** | Просто делит Next transaction на количество дней между датой создания базы данных и датой сбора статистики. Показывает среднее количество транзакций в день. Это значение бесполезно, если это не рабочая база данных. Предупреждения о транзакциях в основном основаны на среднем количестве транзакций за день. |

| Объект или параметр | Описание |
|---|---|
| **Data versions percent** | Процент версий записей в базе данных. Также отображается общий размер записей и размер версий для всех таблиц, а также общий размер индексов. Строка не отображается, если статистика не содержит информации о количестве записей (`gstat -a` без опции `-r`). Обратите внимание, что в вашей базе данных может быть много других данных (transaction inventory pages, empty pages и т. д.). |
| **Table/Index lists** (также отображается в рекомендациях) | |
| **Fragmented Tables** | Здесь вы можете просмотреть таблицы (с данными > 200 килобайт), средняя заполненность которых составляет менее 60 % (File/Options/Table average fill). |
| **Versioned Tables** | Список таблиц Versions которых больше чем Records, установленной в Options/Tables. |
| **Tables fragmented with blobs** | Список таблиц, в которых есть поля BLOB с размером данных меньше размера страницы базы данных. |
| **Massive deletes/updates** | Список таблиц, в которых большое количество данных было удалено/обновлено одним оператором delete/update. |
| **Very big tables** | Таблицы, близкие к техническому лимиту InterBase (36 гигабайт на таблицу). Вы увидите предупреждение, прежде чем может возникнуть проблема. |
| **Deep Indices** | Индексы с глубиной больше 3 (Options/Index) |
| **Bad Indices** | Индексы с большими значениями MaxDup и TotalDup |
| **Broken or incomplete indices** | Индексы с количеством ключей меньше количества записей. Это может произойти, когда индекс сломан или когда статистика собирается во время создания или повторной активации индекса. |
| **Useless Indices** | Индексы с уникальным количество ключей = 1. Могут быть удалены или деактивированы, поскольку они бесполезны для операций поиска по индексу или сортировки. |
| **Tables with no records** | Список таблиц с Records = 0. Это может быть задумано (временные таблицы), а может быть просто забыто разработчиком базы данных. |




На странице сводки отображается много информации, но наиболее ценным является состояние транзакций (*пожалуйста, прочтите описание возможных состояний транзакций в справке IBAnalyst, его можно получить, нажав F1 или в меню Help*).

На этом скриншоте вы можете видеть, что какая-то транзакция активна в течение длительного времени, “60% of daily average”. IBAnalyst помечает состояние такой транзакции красным, поскольку эта транзакция может помешать серверу рассматривать накопленные версии как мусор и, следовательно, собирать мусор. Это возможная причина замедления: чем больше версий существует для некоторой записи, тем больше времени потребуется на ее чтение.

Чтобы найти эту длительную транзакцию, вы можете использовать `MON$Logger` или выполнить прямой запрос к таблицам `MON$`. Затем, чтобы узнать, на какие таблицы повлияли длительные транзакции (таблицы с большим количеством версий записей), вам нужно перейти на закладку “Tables” в IBAnalyst.

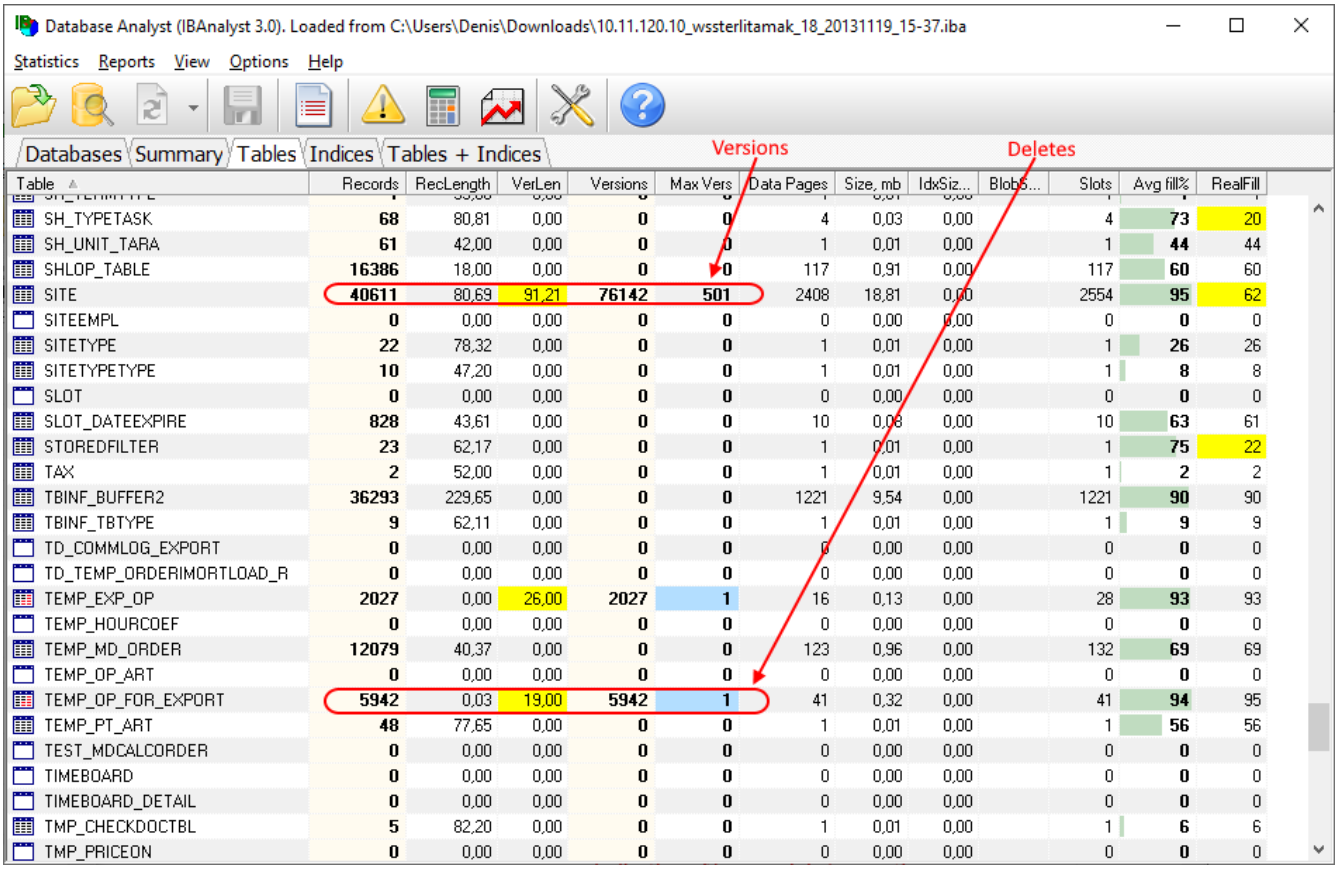
### 7.2.3. Закладка Tables

Закладка **Таблицы** содержит информацию обо всех таблицах базы данных. Она предоставляет важную статистическую информацию о каждой таблице. Все таблицы с предупреждениями отмечаются (подробности см. ниже).

Вы можете увидеть следующие столбцы (столбцы **Records, RecLength, VerLen, Versions, Max Vers** видны только в том случае, если статистика была сгенерирована с помощью `gstat -r` или с установленным флажком “Include record/rec versions”):

| Столбец | Описание |
|---|---|
| **Records** | Количество записей. Помечено розовым, если таблица фрагментирована полями BLOB, данные в которых меньше размера страницы базы данных. Подсказка показывает реальную фрагментацию таблицы и среднее количество записей, как если бы полей BLOB не было. Такая фрагментация может привести к снижению производительности при соединении больших таблиц или полном сканировании. |
| **RecLength** | Средняя длина записи. Зависит от данных записи, особенно от данных столбцов типов char/varchar. Минимальная физическая длина записи — 17 байт (заголовок записи + все поля равны `NULL`), максимальная — как указано в таблице. Статистика показывает эти данные без подсчета заголовков записей, в этом случае RecLength может быть равен 0 (если удалены почти все записи). |
| **VerLen** | Средняя длина версии записи. Если оно близко к RecLength, обновляются почти все записи. Если VerLen составляет 40-80% и не превышает RecLength, то версии в основном представляют собой обновления. Если VerLen превышает 80-90% от RecLength, то возможно, версии в основном удаляются или обновляются столбцы char/varchar с новыми, более крупными данными. Отмечено **желтым**, если его размер превышает указанный процент (Options/Record/Version size) от среднего размера записи. |
| **Versions** | Текущее количество версий записи. Чем больше версий тем медленней читается таблица. Также большое количество версий означает, что сборка мусора не выполняется или записи никем не читаются. Отмечено **красным**, если количество версий превышает количество записей (Options/Record Versions). |
| **Max Vers** | Максимальное количество версий записи для одной конкретной записи. Отмечено **синим**, если оно равно 1, а значение Versions не равно нулю. Это означает, что произошла массовая операция обновления/удаления. Смотри Options, Table, Massive deletes updates option. |
| **Data Pages** | Выделено страниц данных |
| **Size, Mb** | `DataPages * Page Size`, в мегабайтах. То есть это общий размер таблицы, records + versions. График показывает процент этой таблицы от общего объема данных. |

| Столбец | Описание |
|---|---|
| **Idx Size, Mb** | Сумма всех размеров индексов для этой таблицы. График показывает процент этого значения от общего размера всех индексов. |
| **Slots** | Количество ссылок на страницы данных. Пустые ссылки — это Slots-Data Pages. Не влияет на дисковое пространство и производительность. |
| **Average Fill** | Средний процент заполнения страницы данных. Может быть вычислен как `(DataPages * Page_Size)/ Records * RecLength`. Низкая заполнение страницы означает, что таблица “фрагментирована”. Частые обновления/удаления могут фрагментировать страницы данных. Отмечено **красным**, если средний коэффициент заполнения менее 60% (чтобы изменить это зайдите в Options/Average Fill). Отмечается **желтым**, если это артефакт высокой фрагментации таблицы, когда ее запись слишком мала (11-13 байт). |
| **Real Fill** | Поскольку мы обнаружили, что среднее заполнение, рассчитанное с помощью `gstat`, иногда дает неправильные результаты (по крайней мере, для таблиц с небольшими BLOB), мы разместили здесь вычисляемый столбец, который считает среднее заполнение не по страницам данных, а по записям + версиям, включая заголовок записи. |
| **20%, 40%, 60% and 100% fill** | Показывает количество страниц с соответствующим коэффициентом заполнения. Можно включить/выключить в диалоговом окне Options. |
| **Total %** | Насколько велика эта таблица плюс ее индексы в % по отношению к другим таблицам. |




| Table | Records | RecLength | VerLen | Versions | Max Vers | Data Pages | Size, mb | IdxSiz... | BlobS... | Slots | Avg fill% | RealFill |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SH_TYPETASK | 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 |

На закладке “Tables” вы можете увидеть таблицы и их важные параметры: количество записей, количество версий записей, длину записи, максимальное количество версий и т. д.

Вы можете отсортировать эту таблицу, чтобы найти самые большие таблицы. Особенно нас интересуют таблицы с большим количеством версий записей — большое количество версий записей замедлят сборку мусора для затронутых таблиц. Обычно необходимо изменить алгоритмы обновления и удаления, чтобы избавиться от множества версий записей.

Столбец Versions показывает общее количество версий для конкретной таблицы, а столбец Max Vers показывает максимальное количество версий, достигнутых некоторой записью. Например, если вы посмотрите на таблицу SITE, то в ней 40611 записей, а общее количество версий — 76142, но одна запись имеет 501 версию. Чтение и анализ такой цепочки версий с диска занимает больше времени, поэтому чтение этой записи происходит медленнее, чем чтение других.

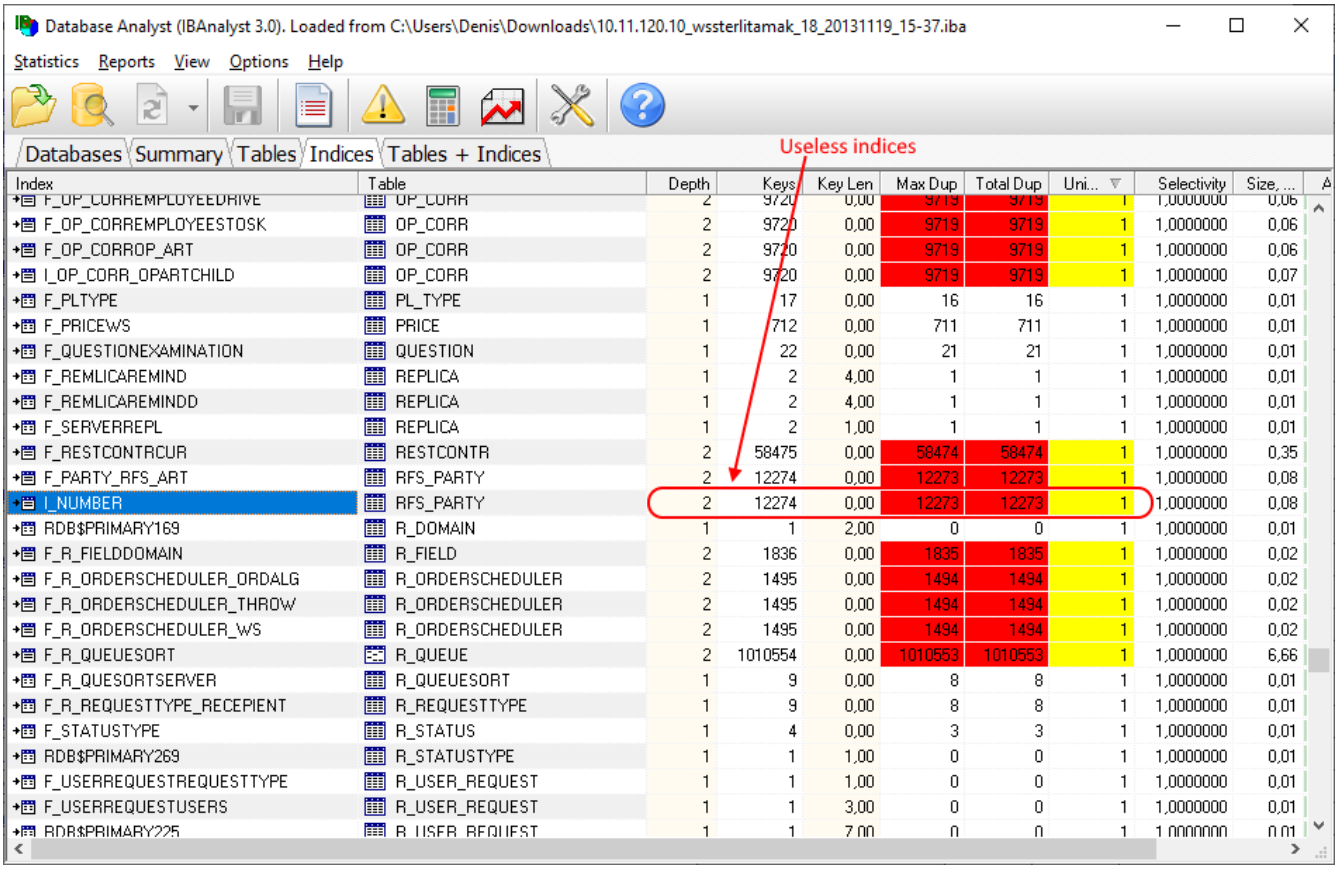
На этом изображении также показано множество таблиц, данные из которых были удалены. Но из-за длительной транзакции сервер не может удалить эти версии, и они все еще находятся на диске, все еще индексируются и все еще читаются сервером при чтении данных.

### 7.2.4. Закладка Indices

Закладка **Indices** отображает все индексы в вашей базе данных. Оценить эффективность индексов можно по следующим параметрам (проблемные индексы отмечены **красным** — подробности смотри в умных подсказках)

| Столбец | Описание |
|---|---|
| **Depth** | Глубина индекса — это количество страниц, которые движок считывает с диска для перехода от корня индекса к указателю на записи. Оптимальная глубина индекса составляет 3 или меньше. Если глубина равна 4 и выше, рекомендуется увеличить размер страницы базы данных (создать резервную копию, а затем восстановить с использованием опции `-page_size`). Этот столбец будет отмечен красным, если глубина индекса превышает 3 (Options/Index/Index Depth). Наибольший шанс превысить оптимальную глубину имеют индексы, построенные на столбцах с длинными char/varchar. |
| **Keys** | Количество ключей индекса. Обычно равно Records. Если Keys больше чем Records и Versions больше 0, то это означает, что значение конкретного поля было изменено в этих версиях записи. Если Table.RecVersions больше чем Keys, то эти поля индекса не изменяются во время обновлений. |
| **KeyLen** | Средняя длина индексного ключа. Чем меньше KeyLen, тем больше равных или похожих (постфиксных) значений (ключей) хранится в индексе. |
| **Max Dup** | Максимальное количество дубликатов для определенного значения ключа. Отмечено **красным**, если количество дубликатов составляет 30 % от всех ключей. (Options/Index/Lot of key duplicates). |
| **Total Dup** | Общее количество ключей с одинаковыми значениями.<br><br>Чем ближе это значение к Keys count, , тем менее эффективным будет поиск по этому индексу, особенно если поиск выполняется с использованием более чем одного индекса. Значение Total Dup можно подсчитать как количество ключей (Keys) минус количество уникальных ключей (Uniques) (статистика индекса нелинейна).<br><br>Отмечено **желтым**, если `1/(Keys - TotalDup)` больше чем 0.01, и **красным**, если в дополнение MaxDup тоже отмечен красным. Эта константа (0.01) используется оптимизатором (см. исходники в `opt.cpp`) в качестве полезной границы селективности индекса. Оптимизатор по-прежнему будет использовать этот индекс, если для какого-либо условия не существует другого индекса с лучшей избирательностью. |

| Столбец | Описание |
| --- | --- |
| **Uniques** | Количество различных значений ключа. Индексы первичного и уникального ключа будут иметь то же значение, что и в столбце Keys. Полезно понимать, сколько разных значений хранится в индексе. Индекс бесполезен, если в столбце Unique указано 1 (отмечено желтым). |
| **Selectivity** | Информация из `rdb$indises.rdb$statistics`, видна только в том случае, если параметр “load table/index metadata” включен. Если селективность, сохраненная в базе данных, отличается от вычисленной селективности, отображается предупреждение **желтого** цвета (разница менее 20%) или **красного** (разница более 20%). **Синее** предупреждение отображается, когда индекс пуст, но его селективность не равна 0. Селективность неактивных индексов игнорируется. |
| **Size, Mb** | Размер индекса в мегабайтах. Gap показывает процент размера этого индекса по отношению к общему размеру всех индексов. |
| **Average Fill** | Средняя заполняемость индексных страниц, в %. Отмечено **красным**, если средняя скорость заполнения менее 50% (для настройки этого перейдите в Options/Average Index Fill). Фрагментированный индекс приводит к большему количеству прочитанных страниц, и обычно его глубина (Depth) может быть выше. Это можно исправить, выполнив `alter index active`, если он не является индексом, созданным для ограничения первичного, уникального или внешнего ключа. |
| **Leafs** | Количество листовых страниц (страницы с ключами и указателями на записи). |
| **20%, 40%, 60% and 100% fill** | Показывает количество страниц с соответствующей коэффициентом заполнения. Можно включить/выключить в диалоговом окне Options. |




Некоторые производственные базы данных могут иметь индексы с индексированным единственным значением ключа. Это может произойти потому, что база данных была разработана “с целью расширения в будущем”, или кто-то просто экспериментировал с индексами во время разработки или тестирования. Вы можете увидеть эти индексы как “Useless” в IBAnalyst: I_NUMBER и другие, построенные на столбце, который имеет только одно значение для всех записей. Эти индексы действительно бесполезны, потому что

- Оптимизатор может использовать этот индекс, если вы укажете “where field =...”. Поскольку поле содержит только одно значение, использование индекса приведет к бесполезному чтению страниц индекса с диска в память и потребует памяти (и времени), когда сервер будет подготавливать записи для отображения для этого запроса.
- Создание индексов является частью процесса восстановления. Дополнительные индексы замедляют восстановление базы данных.

Конечно, это еще не все, что вы можете узнать о своей базе данных в IBAnalyst. Вы также можете найти

- среднее количество транзакций в день
- были ли откаты или потеря соединения и когда
- размер (в мегабайтах) каждой таблицы и индекса
- таблицы, в которых записи перемешаны с BLOB, из-за чего чтение только записей происходит медленнее
- пустые таблицы — просто забыты или пусты на момент снятия статистики
- индексы с большим количеством повторяющихся ключей (нужно задуматься о распределении значений для столбца)

- индексы с глубиной 4 и выше — возможно, вам нужно увеличить размер страницы для ускорения

# Глава 8. Поддержка шифрования

HQbird включает плагин шифрования и обеспечивает поддержку работы с базами данных шифрования в веб-интерфейсе. Эта функция поддерживается только в HQBird с Firebird 3.0 и выше.

**Обратите внимание!**

Плагин шифрования требует отдельного файла лицензии, он отправляется в электронном письме о покупке.

Плагин шифрования можно приобрести отдельно и использовать с community версией Firebird: https://ib-aid.com/download-demo-firebird-encryption-plugin

## 8.1. Файлы OpenSSL

Этот продукт включает программное обеспечение, разработанное OpenSSL Project использующееся в OpenSSL Toolkit (http://www.openssl.org/).

HQbird для Windows уже включает в себя необходимые двоичные файлы OpenSSL 1.1. Чтобы использовать функции шифрования в Linux, необходимо установить **OpenSSL 1.1**.

## 8.2. Как зашифровать и дешифровать базу данных Firebird

В этом кратком руководстве ниже мы продемонстрируем ключевые особенности шифрования: как зашифровать базу данных Firebird на сервере, как реализовать зашифрованное клиентское соединение и выполнить резервное копирование/восстановление зашифрованной базы данных. Демо-версия FEPF полностью функциональна, за единственным исключением — она ограничена до 30 декабря 2019 года.

### 8.2.1. Демо-пакет с примерами клиентских приложений

Загрузите демо-пакет с примерами клиентских приложений по ссылке https://ib-aid.com/download/crypt/CryptTest.zip

### 8.2.2. Этап 1. Первоначальное шифрование базы данных.

На данный момент мы предполагаем, что у вас есть некоторая база данных, которую нужно зашифровать. Поместите незашифрованную базу данных по какому-нибудь пути, например, в `c:\temp\employee30\employee.fdb`

Мы предполагаем, что все действия производятся с 64-битной версией Firebird 3.0.3+, в случае 32-битной версии просто используйте файлы из папки `WinSrv32bit_ServerPart`.

1. Создайте следующий псевдоним в `databases.conf`

```
crypt = C:\Temp\EMPLOYEE30\EMPLOYEE30.FDB
{
   KeyHolderPlugin = KeyHolder
}
```

Также вы можете объявить плагин KeyHolder для всех баз на сервере, для этого добавьте в firebird.conf следующий параметр:

```
KeyHolderPlugin = KeyHolder
```

или просто скопируйте firebird.conf на шаге 2 (см. ниже).

2. Убедитесь, что в server/plugins скопированы следующие файлы из папки WinSrv64Bit_ServerPart\plugins
   - DbCrypt.dll
   - DbCrypt.conf
   - KeyHolder.dll
   - KeyHolder.conf — этот текстовый файл с ключами предназначен только для использования разработчиками, его нельзя отправлять конечным пользователям!
3. Поместите следующие файлы в корень Firebird из папки WinSrv64Bit_ServerPart
   - fbcrypt.dll
   - libcrypto-1_1-x64.dll
   - libssl-1_1-x64.dll
   - gbak.exe
   - firebird.msg
   - firebird.conf (необязательно, может использоваться в качестве примера)
4. Подключитесь к незашифрованной базе данных с помощью isql и зашифруйте базу данных:

```
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
```

Давайте рассмотрим команду шифрования:

```
alter database encrypt with dbcrypt key red;
```

Здесь `dbcrypt` — это имя плагина шифрования, а `red` — имя используемого ключа. Ключи определяются в файле `KeyHolder.conf`.

Обратите внимание: в Linux необходимо использовать кавычки и имя плагина с учетом регистра:

```
alter database encrypt with "DbCrypt" key Red;
```

После этого база данных шифруется с аутентификацией на стороне сервера: ключи находятся в файле `KeyHolder.conf`.

На рисунке ниже вы можете увидеть файлы, которые необходимы на сервере для включения шифрования, и файлы, которые необходимы на стороне клиента:

| **На серверной стороне:** | **На стороне клиента (Демо - `CryptTest.exe`) - 32bit** |
|---|---|
| **Обязательные файлы:** | **Обязательные файлы:** |
| `plugins/dbcrypt.dll` | `fbclient.dll` |
| `plugins/keyholder.dll` | `fbcrypt.dll` |
| `DbCrypt.conf` | `libcrypto-1_1.dll` |
| `libssl-1_1-x64.dll` | **Необязательные файлы:** |
| `libcrypto-1_1-x64.dll` | `firebird.conf` |
| `fbcrypt.dll` | |
| **Файлы для gbak с шифрованием:** | |
| `gbak.exe` | |
| `firebird.msg` | |
| **Необязательные файлы:** | |
| `plugins/KeyHolder.conf` (для первоначального шифрования в режиме разработки) | |
| `firebird.conf` (содержит параметр для установки плагина шифрования) | |

### 8.2.3. Этап 2. Подключение к зашифрованной базе данных с помощью клиентского приложения.

Предполагаем, что после первоначального шифрования база данных будет скопирована в среду заказчика, где доступ к ней будет осуществляться только через авторизованное приложение.

Чтобы имитировать такую среду, нам нужно удалить (или просто переименовать) файл с ключами (`KeyHolder.conf`) из папки плагинов.

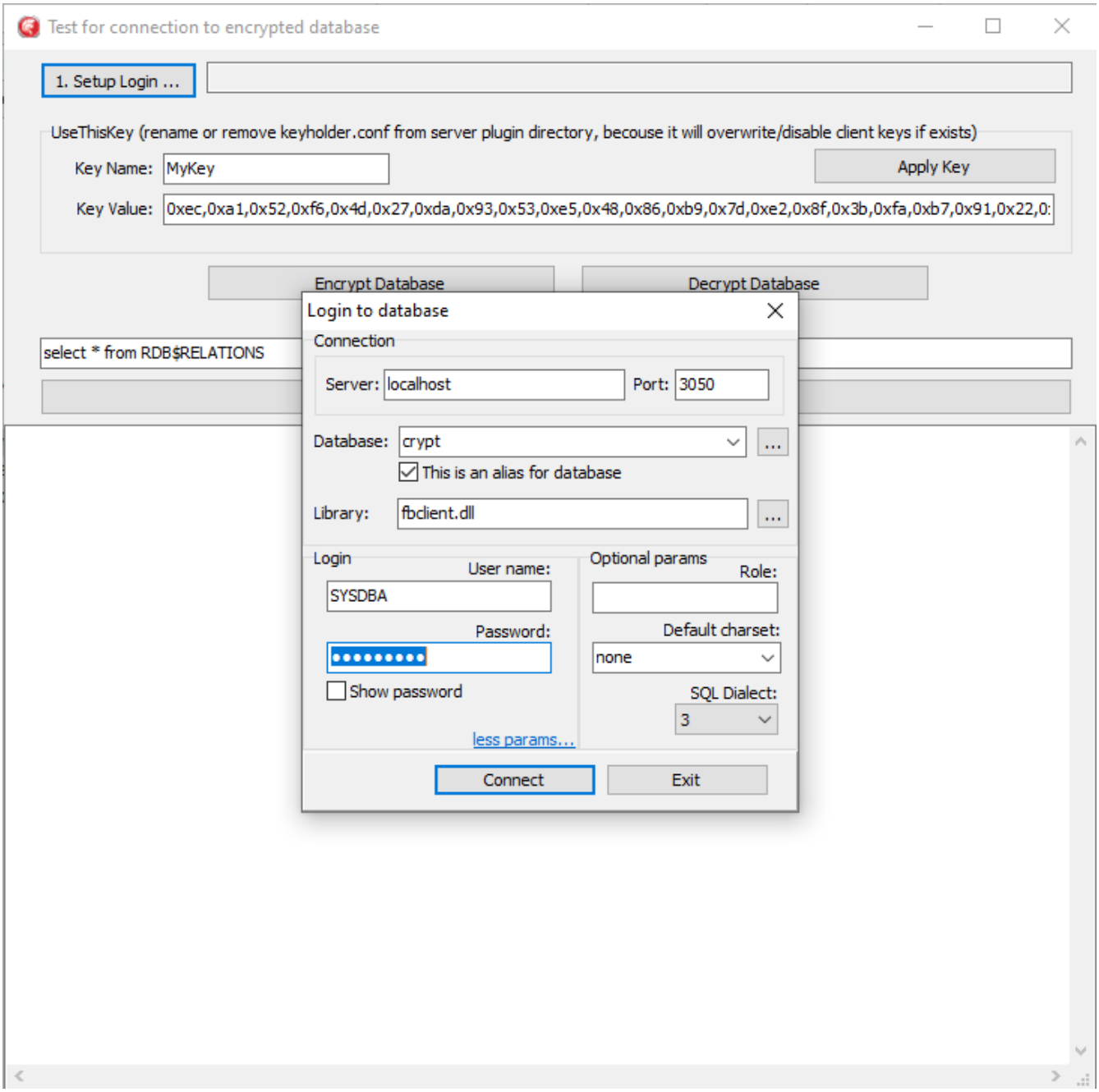
Без `KeyHolder.conf` плагин шифрования потребует получения ключа от подключенного приложения. Пример такого приложения включен в архив с демонстрационным плагином — для него есть скомпилированная версия и полные исходники на Delphi XE8.

Код инициализации зашифрованного соединения очень прост — перед обычным соединением необходимо выполнить несколько вызовов для отправки соответствующего ключа. После этого клиентское приложение работает с Firebird в обычном режиме.

Запустите демо-приложение для проверки работы с зашифрованной базой данных, оно находится в папке `CryptTest\EnhancedCryptTestClient\Win32\Debug`.

Выполните следующие шаги:

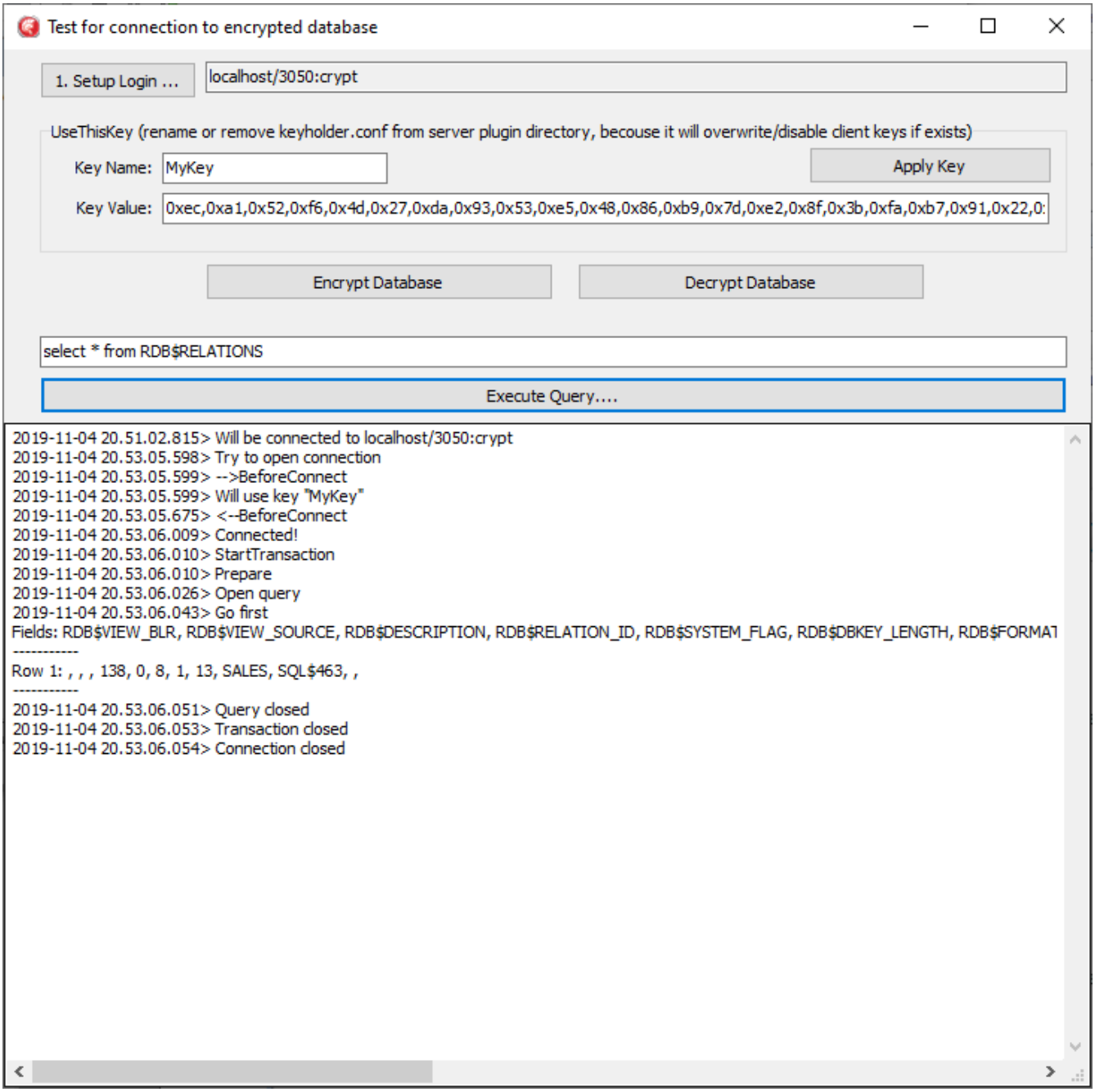
1. Укажите путь к базе данных или псевдоним в “1. Setup Login”. Эта база данных будет использоваться на следующих этапах.
2. Укажите имя и значение ключа, которые будут использоваться. Если вы ранее использовали ключ RED, установите `Key Name = RED` и скопируйте значение ключа из файла `KeyHolder.conf`.
3. Вы можете зашифровать и расшифровать базу данных с помощью указанного ключа. Обратите внимание: шифрование требует времени и активного подключения к базе данных.
4. Нажмите Execute query, чтобы проверить соединение с зашифрованной базой данных




**Обратите внимание!**

Тестовое приложение может подключаться к зашифрованной базе данных только через TCP/IP, xnet не поддерживается.

В примере клиентского приложения все операции с базой данных (подключение, запуск транзакции, подтверждение транзакции, запуск запроса и т. д.) выполняются очень простым способом, чтобы продемонстрировать все этапы операций с зашифрованной базой данных. Вы можете использовать этот код в качестве примера реализации шифрования в ваших приложениях.




### 8.2.4. Этап 3 — резервное копирование и восстановление зашифрованной базы данных.

Полная проверенная резервная копия сделанная с помощью `gbak.exe` является основным методом резервного копирования баз данных Firebird. В стандартный дистрибутив Firebird входит инструмент командной строки `gbak.exe` для её выполнения, однако он не будет работать с зашифрованной базой данных в рабочем режиме (без ключей на сервере). После шифрования только авторизованные приложения могут получить доступ к зашифрованной базе данных, а стандартный `gbak` не является авторизованным приложением.

Мы все знаем, насколько важно резервное копирование и восстановление для работоспособности и производительности базы данных, поэтому для выполнения резервного копирования и восстановления зашифрованных баз данных мы разработали `gbak.exe` с поддержкой шифрования и включили его в FEPF.

Важно отметить, что этот `gbak.exe` создает зашифрованный файл резервной копии: он шифрует резервную копию тем же ключом, что и для шифрования базы данных.

Если вы запустите `gbak.exe` из файлов плагина с ключом `-?`, вы увидите новые параметры `gbak.exe`, которые используются для работы с зашифрованными базами данных:

```
-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
```

Давайте рассмотрим, как использовать `gbak.exe` с зашифрованными базами данных и резервными копиями.

### Резервное копирование зашифрованной базы данных Firebird

Для резервного копирования зашифрованной базы данных Firebird `gbak.exe` должен предоставить ключ для сервера. Этот ключ будет использоваться для подключения и чтения базы данных, а также для шифрования файла резервной копии.

Есть два способа предоставить ключ для `gbak.exe`: сохранить ключ в файле ключей или явно указать его в командной строке:

*Пример 1. Пример создания резервной копии с ключом шифрования в файле ключей*

```
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
```

Здесь в параметре `-KEYFILE` мы указываем расположение файлов с ключами, а в `-KEYNAME` — имя используемого ключа. Обратите внимание, что файл `examplekeyfile.txt` имеет ту же структуру, что и `KeyHolder.conf`.

Если вы запустите создание резервной копии с шифрованием (`gbak -b -keyfile … -keyname …`) к незашифрованной базе данных, то резервная копия будет зашифрована.

*Пример 2. Пример резервного копирования с явным ключом*

```
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
```

Здесь мы указываем значение ключа в параметре `-KEY` и имя ключа в параметре `-KEYNAME`. Необходимо указать имя ключа, даже если мы указываем явное значение ключа.

### Восстановление резервной копии зашифрованной базы данных Firebird.

gbak также может восстанавливать данные из файлов резервных копий в зашифрованные базы данных. Подход тот же: нам нужно указать имя ключа и значение ключа для восстановления файла резервной копии.

Ниже приведены примеры команд восстановления:

*Пример 3. Пример восстановления с использованием ключа шифрования в файле ключа*

```
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
```

*Пример 4. Пример восстановления с явным ключом*

```
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
```

Если вы выполняете восстановление из незашифрованного файла резервной копии с ключами шифрования (gbak -c -keyfile … -keyname …), то восстановленная база данных будет зашифрована.

# Глава 9. Authentication plugin for EXECUTE STATEMENT ON EXTERNAL

В Firebird есть удобный механизм запросов к другим базам данных: `EXECUTE STATEMENT ON EXTERNAL` (ESOE). Например, типовой ESOE может выглядеть так:

```
EXECUTE STATEMENT 'SELECT * FROM RDB$DATABASE'
ON EXTERNAL 'server:db1' AS USER 'MYUSER' PASSWORD 'mypassword'
```

Как видите, оператор содержит имя пользователя и пароль в открытом виде, что небезопасно: например, если ESOE вызывается из кода хранимой процедуры, подключенные пользователи смогут увидеть пароль.

HQbird включает плагин аутентификации для ESOE, позволяющий устанавливать доверительные отношения между серверами Firebird и выполнять аутентификацию ESOE без пароля:

```
EXECUTE STATEMENT 'SELECT * FROM RDB$DATABASE'
ON EXTERNAL 'server:db1' AS USER 'MYUSER';
```

Рассмотрим, как установить и настроить плагин HQbird Authentication для ESOE.

## 9.1. Установка плагина аутентификации для ESOE

### 9.1.1. Файлы плагина аутентификации

- Windows
  - `plugins\cluster.dll`
  - `clusterkeygen.exe`
- Linux
  - `plugins\libcluster.so`
  - `bin\ClusterKeygen` # executable

**Обратите внимание!**

Файлы плагина уже включены в HQbird, поэтому копировать их не нужно.

### 9.1.2. Конфигурация

#### В firebird.conf

Прежде всего, необходимо добавить имя плагина (Cluster) в параметры `AuthServer` и `AuthClient` в `firebird.conf` на всех серверах, которые будут доверять друг другу:

```
AuthServer = Srp, Legacy_Auth, Cluster
AuthClient = Srp, Srp256, Legacy_Auth, Cluster
```

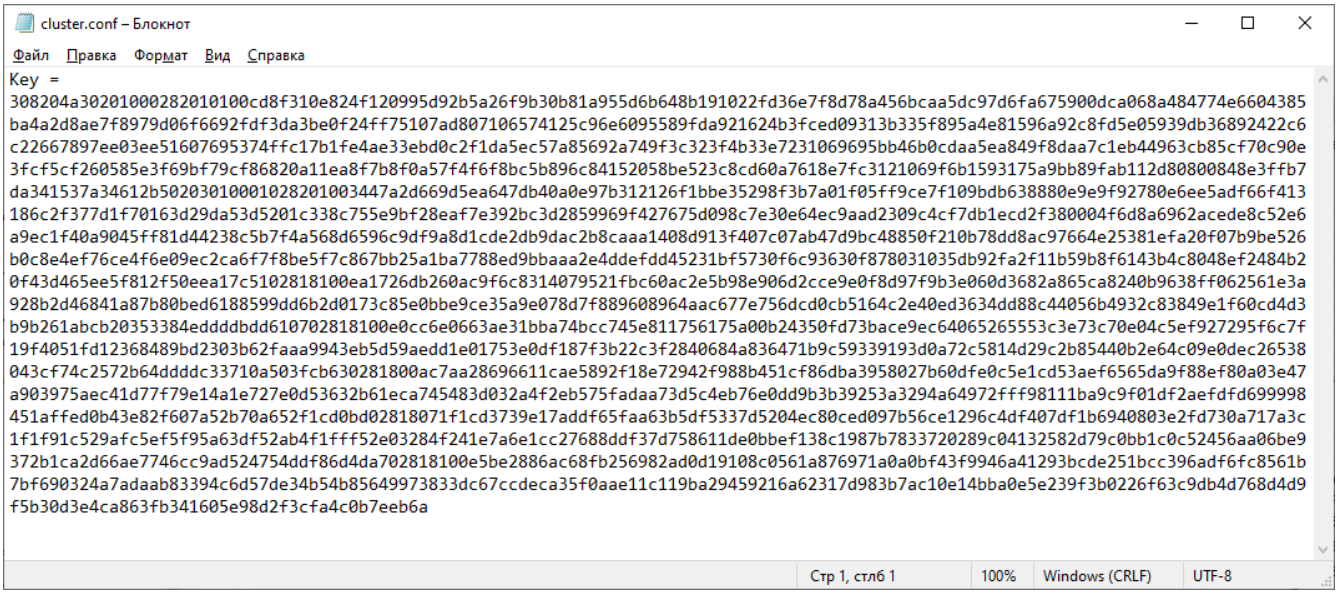
### Файл ключа

Затем необходимо сгенерировать файл ключа для плагина. Этот ключ должен быть размещен на всех серверах Firebird, которые должны доверять друг другу.

Чтобы сгенерировать файл ключа cluster.conf, выполните следующую команду:

```
C:\HQbird\Firebird30>clusterkeygen.exe > cluster.conf
```

В результате будет сгенерирован файл ключа `cluster.conf`. Он содержит 2048-значный ключ, примерно такой:




Затем нам нужно скопировать файл ключа на все серверы Firebird с доверенными отношениями в папку `plugins`. Ключ, созданный в Windows, можно использовать в Linux и наоборот.

**Обратите внимание!**

Имя файла ключа должно быть именно `cluster.conf`. Он должен расположен в папке `plugins` Firebird.

### Отображение

Чтобы использовать плагин аутентификации внутри конкретной базы данных, необходимо создать отображение (mapping) между пользователями плагина Cluster и обычными пользователями Firebird.

Например, если мы запустим следующую запрос ESOE

```
EXECUTE STATEMENT 'SELECT * FROM RDB$DATABASE'
ON EXTERNAL 'server:db1' AS USER 'MYUSER';
```

нам нужно сопоставить пользователя MYUSER с реальным пользователем в целевой базе данных db1.

Предположим, у нас есть пользователь MYUSER2 в целевой базе данных. В этом случае нам нужно выполнить следующую команду в целевой базе данных db1:

```
CREATE MAPPING usr_mapping_cluster1 USING PLUGIN CLUSTER
FROM USER MYUSER TO user MYUSER2;
```

В результате в db1 будет создано отображение usr_mapping_cluster1 для сопоставления пользователя MYUSER с MYUSER2.

**Обратите внимание!**

Оба пользователя должны существовать, даже если у них одинаковое имя. В противном случае будет следующая ошибка:

```
Execute statement error at attach:

335544472: Your user name and password are not defined. Ask your
database
administrator to set up a Firebird login.
```

Вы можете создать столько отображений, сколько вам нужно. Существующее отображения можно найти в таблице RDB$AUTH_MAPPING с помощью следующего запроса:

```
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
```

### Глобальные отображения

Можно создать отображение между пользователями для всех баз данных на сервере — в этом случае следует использовать следующий запрос:

```
CREATE GLOBAL MAPPING global_usr_mapping_cluster1 USING PLUGIN CLUSTER
FROM USER MYUSER TO user MYUSER2;
```

В этом случае отображения будут храниться в базе данных безопасности, чтобы увидеть их,

используйте следующий запрос:

```
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
```

#### Отображение ролей

Чтобы сопоставить пользователя с ролью в целевой базе данных, необходимо создать 2 отображения:

```
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. Как протестировать

Следующий запрос можно использовать для проверки работы плагина аутентификации для 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
```

В результате этот запрос вернет имя пользователя, идентификатор соединения и роль пользователя из целевой базы данных `db1`.

# Глава 10. Работа с внешними источниками данных (другие СУБД)

В HQBird для Firebird 4.0 и выше появилась возможность работать с внешними источниками данных, то есть не только с другими базами данных СУБД Firebird, но и другими СУБД. Для этого используются следующие плагины:

- `MySQLEngine` для работы с MySQL и MariaDB;
- `ODBCEngine` для работы с любым источником данных через соответствующий ODBC драйвер.

В настоящее время доступно только под Windows.

## 10.1. MySQLEngine

Плагин `MySQLEngine` предназначен для доступа к базам данных MySQL и MariaDB.

Для того чтобы плагин был известен Firebird необходимо отредактировать файл конфигурации `firebird.conf` добавив в список провайдеров (параметр `Providers`) плагин `MySQLEngine`:

```
Providers = Remote,Engine13,MySQLEngine,Loopback
```

Теперь можно попробовать выполнить простейший запрос к базе данных MySQL

```
execute block
returns (i integer)
as
  declare dsn_mysql varchar(128);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';

  for
    execute statement 'select 1'
    on external dsn_mysql
    as user null password 'sa'
    into i
  do
    suspend;
end
```

### 10.1.1. Формат строки соединения

Строка соединения для плагина `MySQLEngine` должна начинаться с префикса `:mysql:` за которым следуют параметры подключения в виде `<param>=<value>` разделённых точкой с

запятой.

Возможные параметры:

- host — хост или IP адрес на котором находится сервер MySQL;
- port — номер порта, который слушает MySQL сервер. Можно не указывать если используется порт по умолчанию (3306);
- database — имя базы данных в которой будут выполняться запросы;
- user — имя пользователя;
- password — пароль.

**Замечания**

- Не указывайте в строке подключения пароль пользователя (это не безопасно), лучше передавать его с использованием ключевого слова `password`;
- Не указывайте имя пользователя с помощью ключевого слова `user` (оставьте `NULL` или пустую строку), поскольку это не работает совместно с провайдером `Remote`.

### 10.1.2. Поддерживаемые типы запросов

Плагин MySQLEngine поддерживает следующие типы запросов:

- `SELECT`
- `INSERT`, `UPDATE`, `DELETE`
- `CALL procedure(<params>)`

### 10.1.3. Выходные параметры запросов

Типы данных возвращаемые из запроса

| Тип данных MySQL | Тип данных Firebird |
|---|---|
| TINYINT | SMALLINT |
| SMALLINT | SMALLINT |
| MEDIUMINT | INTEGER |
| INT | INTEGER |
| BIGINT | BIGINT |
| FLOAT | FLOAT |
| DOUBLE | DOUBLE PRECISION |
| DECIMAL | VARCHAR(N) |
| YEAR | SMALLINT |
| TIME | TIME |
| DATE | DATE |

| Тип данных MySQL | Тип данных Firebird |
|---|---|
| TIMESTAMP | TIMESTAMP |
| DATETIME | TIMESTAMP |
| CHAR(N), BINARY(N) | CHAR(N), BINARY(N) |
| VARCHAR(N), VARBINARY(N) | VARCHAR(N), VARBINARY(N) |
| TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT | BLOB SUB_TYPE TEXT |
| TYNYBLOB, BLOB, MEDIUMBLOB, LONGBLOB | BLOB SUB_TYPE BINARY |
| JSON | BLOB SUB_TYPE TEXT |
| BIT(N) | VARBINARY(N) |

Пример запроса с несколькими полями:

```
execute block
returns (
  emp_no bigint,
  birth_date date,
  first_name varchar(14),
  last_name varchar(16),
  gender char(1),
  hire_date date
)
as
  declare dsn_mysql varchar(128);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';

  for
    execute statement q'{
      select
        emp_no,
        birth_date,
        first_name,
        last_name,
        gender,
        hire_date
      from employees
      order by birth_date desc
      limit 5
    }'
    on external dsn_mysql
    as user null password 'sa'
    into
      emp_no, birth_date, first_name, last_name, gender, hire_date
  do
    suspend;
end
```

Запрос SELECT всегда возвращает курсор.

В MySQL запросы CALL могут возвращать значения через параметры типа OUT и INOUT. Синтаксическому анализатору Firebird ничего неизвестно об операторе типа CALL, поэтому возврат параметров типа OUT и INOUT не поддерживается.

Это может изменится в будущем. В Firebird 6.0 добавлена возможность вызова хранимых процедур через оператор CALL.

Однако вы можете возвратить параметры типа OUT и INOUT с помощью локальных переменных и последовательного выполнения нескольких запросов.

Предположим у вас есть следующая хранимая процедура:

```
CREATE PROCEDURE `sp_test_add`(
  IN `A` INT,
  IN `B` INT,
  OUT `C` INT
)
LANGUAGE SQL
NOT DETERMINISTIC
NO SQL
SQL SECURITY DEFINER
BEGIN
  SET C = A + B;
END
```

Тогда результат выполнения процедуры можно вернуть следующим образом:

```
execute block
returns (
  c int
)
as
  declare dsn_mysql varchar(128);
  declare psw_mysql varchar(25);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';
  psw_mysql = 'sa';

  execute statement 'SET @C=NULL'
  on external dsn_mysql
  as user null password psw_mysql;

  execute statement
  ('CALL sp_test_add(?, ?, @C)')
  (1, 2)
  on external dsn_mysql
  as user null password psw_mysql;
```

```
  execute statement
  'SELECT @C'
  on external dsn_mysql
  as user null password psw_mysql
  into c;

  suspend;
end
```

Запросы CALL также могут возвращать курсор или несколько курсоров. В текущей версии возврат курсора из запросов CALL не поддерживается. Работа с несколькими наборами данных с использованием оператора EXECUTE STATEMENT ⋯ ON EXTERNAL не поддерживается.

### 10.1.4. Входные параметры запросов

Плагин MySQLEngine поддерживает использование параметров в запросах. Параметры могут быть безымянные (позиционные) или именованные.

Пример использования безымянных параметров:

```
execute block
returns (
  emp_no bigint,
  birth_date date,
  first_name varchar(14),
  last_name varchar(16),
  gender char(1),
  hire_date date
)
as
  declare dsn_mysql varchar(128);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';

  for
    execute statement (q'{
      select
        emp_no,
        birth_date,
        first_name,
        last_name,
        gender,
        hire_date
        from employees
      where emp_no = ?
    }')
    (10020)
    on external dsn_mysql
    as user null password 'sa'
```

```
    into
      emp_no, birth_date, first_name, last_name, gender, hire_date
  do
    suspend;
end
```

Пример использования именованных параметров:

```
execute block
returns (
  emp_no bigint,
  birth_date date,
  first_name varchar(14),
  last_name varchar(16),
  gender char(1),
  hire_date date
)
as
  declare dsn_mysql varchar(128);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';

  for
    execute statement (q'{
      select
        emp_no,
        birth_date,
        first_name,
        last_name,
        gender,
        hire_date
      from employees
      where emp_no = :emp_no
    }')
    (emp_no := 10020)
    on external dsn_mysql
    as user null password 'sa'
    into
      emp_no, birth_date, first_name, last_name, gender, hire_date
  do
    suspend;
end
```

### 10.1.5. Ограничение использования входных параметров

Для работы именных параметров подсистема EDS (EXTERNAL DATA SOURCE) использует внутренний предварительный анализатор запросов, который заменяет все параметры вида :<name> на ? и сохраняет привязку имени параметра к его номеру. Поэтому это работает только для запросов синтаксис которых схож с синтаксисом Firebird. Например, для

запросов CALL именованные параметры работать не будут. В этом случае необходимо использовать безымянные параметры.

```
set term ;#

execute block
as
  declare dsn_mysql varchar(128);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';

  execute statement
  ('CALL sp_conn_audit(:A_CONN_ID, :A_USER, :A_DT)')
  (
    A_CONN_ID := current_connection,
    A_USER := current_user,
    A_DT := localtimestamp
  )
  on external dsn_mysql
  as user null password 'sa';
end#
```

```
Statement failed, SQLSTATE = 42000
Execute statement error at isc_dsql_prepare :
335544382 : You have an error in your SQL syntax; check the manual that corresponds to
your MariaDB server version for the right syntax to use near ':A_CONN_ID, :A_USER,
:A_DT)' at line 1
Statement : CALL sp_conn_audit(:A_CONN_ID, :A_USER, :A_DT)
Data source : Firebird:::mysql:host=localhost;port=3306;database=employees;user=root
-At block line: 7, col: 3
```

Если же заменить именованные параметры безымянными, то запрос успешно отработает

```
execute block
as
  declare dsn_mysql varchar(128);
begin
  dsn_mysql = ':mysql:host=localhost;port=3306;database=employees;user=root';

  execute statement
    ('CALL sp_conn_audit(?, ?, ?)')
    (current_connection, current_user, localtimestamp)
  on external dsn_mysql
  as user null password 'sa';
end#
```

Это может изменится в будущем. В Firebird 6.0 добавлена возможность

вызова хранимых процедур через оператор `CALL`.

Firebird при выполнении `prepare` получает типы, размеры и другие свойства входных и выходных параметров запросов. Далее исходя из этих данных строятся входные и выходные сообщения, выделяются буферы для обмена данными. MySQL умеет возвращать типы, размеры и свойства выходных параметров (столбцов), а для входных параметров возвращается только их общее количество. MySQL C-API устроен так, что типы, размеры и другие атрибуты для входных параметров задаются клиентским приложением. Однако в Firebird API невозможно целиком и полностью определить входное сообщение самостоятельно, возможно только преобразовать входное сообщение возвращённое после `prepare` в другое сообщение (совместимое по типам).

Поскольку невозможно узнать типы входных параметров, мы задаём всем параметрам тип `VARCHAR(8191) CHARACTER SET UTF8`. Большинство типов Firebird могут быть преобразованы в строку и обратно. Однако вы не можете передавать в такие параметры бинарные данные (типы `BINARY(N)`, `VARBINARY(N)` и `BLOB SUB_TYPE BINARY`), поскольку они будут искажены. Кроме того, нельзя передавать в качестве параметров `BLOB SUB_TYPE TEXT`, если текст превышает длину 8191 символов.

# 10.2. ODBCEngine

Плагин `ODBCEngine` предназначен для доступа к различным базам данных через интерфейс ODBC.

Для того чтобы плагин был известен Firebird необходимо отредактировать файл конфигурации `firebird.conf` добавив в список провайдеров (параметр `Providers`) плагин `ODBCEngine`:

```
Providers = Remote,Engine13,ODBCEngine,Loopback
```

## 10.2.1. Формат строки соединения

Строка соединения для плагина ODBCEngine должна начинаться с префикса `:odbc:` за которым следуют параметры подключения. Возможно два варианта подключения к базе данных: с помощью DNS или с помощью полной строки соединения для указанного драйвера.

Возможные параметры строки подключения:

- DSN — DSN источника данных (обязателен, если нет DRIVER);
- DRIVER — имя ODBC драйвера (обязателен, если нет DSN);
- UID или USER — имя пользователя;
- PWD или PASSWORD — пароль;
- другие параметры специфичные для выбранного драйвера.

В Windows, если Firebird работает как служба, видны только **системные** DSN.

Приведём примеры подключения к базе данных MySQL. Допустим вы настроили системный DSN с именем `test_dsn`, тогда строка подключения будет выглядеть следующим образом:

```
execute block
returns (
  i integer
)
as
begin
  for
    execute statement q'{
       select 1
    }'
    on external ':odbc:dsn=test_dsn;user=root'
    as user null password '12345'
    into
      i
  do
    suspend;
end
```

Другой вариант подключения к той же базе данных с помощью полной строки соединения. Набор допустимых параметров в строке соединения зависит от выбранного драйвера.

Например для драйвера MariaDB подключение будет выглядеть следующим образом:

```
execute block returns (
  i integer
)
as
begin
  for
    execute statement 'select 1'
    on external ':odbc:DRIVER={MariaDB ODBC 3.1
Driver};SERVER=127.0.0.1;PORT=3306;DATABASE=test;TCPIP=1;CHARSET=utf8mb4;UID=root'
    as user null password '12345'
    into i
  do
    suspend;
end
```

**Замечания**

- Не указывайте в строке подключения пароль пользователя (это не безопасно), лучше передавать его с использованием ключевого слова `password`;
- Не указывайте имя пользователя с помощью ключевого слова `user` (оставьте `NULL` или пустую строку), поскольку это не работает совместно с

провайдером Remote.

- Для корректной работы со строками не ASCII кодировки всегда указывайте в DSN набор символов подключения совместимый с UTF8. В разных драйверах ODBC это делается по разному.

### 10.2.2. Таблица соответствия типов данных между ODBC и Firebird

| Тип данных ODBC | Тип данных Firebird |
|---|---|
| SQL_CHAR, SQL_WCHAR | VARCHAR(N), если длина не превышает 32765 байт, в противном случае BLOB SUB_TYPE TEXT |
| SQL_VARCHAR, SQL_WVARCHAR | VARCHAR(N), если длина не превышает 32765 байт, в противном случае BLOB SUB_TYPE TEXT |
| SQL_BINARY | VARBINARY(N), если длина не превышает 32765 байт, в противном случае BLOB SUB_TYPE BINARY |
| SQL_VARBINARY | VARBINARY(N), если длина не превышает 32765 байт, в противном случае BLOB SUB_TYPE BINARY |
| SQL_TINYINT, SQL_SMALLINT | SMALLINT. Если SQL_SMALLINT беззнаковый, то INTEGER. |
| SQL_INTEGER | INTEGER. Если SQL_INTEGER беззнаковый, то BIGINT. |
| SQL_BIGINT | BIGINT. Если SQL_BIGINT беззнаковый, то VARCHAR(20). |
| SQL_REAL | FLOAT |
| SQL_DOUBLE, SQL_FLOAT | DOUBLE PRECISION |
| SQL_TYPE_DATE | DATE |
| SQL_TYPE_TIME | TIME |
| SQL_TYPE_TIMESTAMP | TIMESTAMP |
| SQL_DECIMAL | VARCHAR(N), где N = precision + 2 |
| SQL_NUMERIC | VARCHAR(N), где N = precision + 2 |
| SQL_LONGVARCHAR | BLOB SUB_TYPE TEXT |
| SQL_WLONGVARCHAR | BLOB SUB_TYPE TEXT |
| SQL_LONGVARBINARY | BLOB SUB_TYPE BINARY |
| SQL_BIT | BOOLEAN |
| SQL_GUID | VARBINARY(16) |

### 10.2.3. Типы запросов

Запрос SELECT всегда возвращает курсор.

Запросы CALL могут возвращать значения через параметры типа OUT и INOUT.

Синтаксическому анализатору Firebird ничего неизвестно об операторе типа CALL, поэтому возврат параметров типа OUT и INOUT не поддерживается.

Это может изменится в будущем. В Firebird 6.0 добавлена возможность вызова хранимых процедур через оператор CALL.

Запросы CALL также могут возвращать курсор или несколько курсоров. В текущей версии возврат курсора из запросов CALL не поддерживается. Работа с несколькими наборами данных с использованием оператора EXECUTE STATEMENT ⋯ ON EXTERNAL не поддерживается.

Запросы типа INSERT, UPDATE, DELETE обычно не возвращают данных, если не указано предложение RETURNING, в противном случае возвращается курсор.

Пример выполнения SELECT запроса.

```
execute block
returns (
  id     integer,
  title  varchar(255),
  body blob sub_type text,
  bydate varchar(50)
)
as
declare sql varchar(8191);
begin
  sql = Q'{
    SELECT
      id,
      title,
      body,
      bydate
    FROM article
  }';
  for
    execute statement (:sql)
    on external ':odbc:DRIVER={MariaDB ODBC 3.1
Driver};SERVER=127.0.0.1;PORT=3306;DATABASE=test;TCPIP=1;CHARSET=utf8mb4;UID=root'
    as user null password 'root'
    into
      id,
      title,
      body,
      bydate
  do
    suspend;
end
```

### 10.2.4. Входные параметры запросов

Плагин ODBCEngine поддерживает использование параметров в запросах. Параметры могут быть безымянные (позиционные) или именованные.

Пример использования безымянных параметров:

```
execute block
returns(
   CODE_SEX INTEGER,
   NAME VARCHAR(70),
   NAME_EN VARCHAR(70)
)
as
declare xSQL varchar(8191);
declare xCODE_SEX INT = 1;
begin
  xSQL = '
SELECT
  CODE_SEX,
  NAME,
  NAME_EN
FROM sex
WHERE CODE_SEX = ?
';
  for
    execute statement (:xSQL)
    (xCODE_SEX)
    on external ':odbc:DRIVER={MariaDB ODBC 3.1
Driver};SERVER=127.0.0.1;PORT=3306;DATABASE=test;TCPIP=1;CHARSET=utf8mb4;UID=test'
    as user null password '12345'
    into CODE_SEX, NAME, NAME_EN
  do
    suspend;
end
```

Пример использования именованных параметров:

```
execute block
returns(
   CODE_SEX INTEGER,
   NAME VARCHAR(70),
   NAME_EN VARCHAR(70)
)
as
declare xSQL varchar(8191);
declare xCODE_SEX INT = 1;
begin
  xSQL = '
```

```
SELECT
  CODE_SEX,
  NAME,
  NAME_EN
FROM sex
WHERE CODE_SEX = :A_CODE_SEX
';
  for
    execute statement (:xSQL)
    (A_CODE_SEX := xCODE_SEX)
    on external ':odbc:DRIVER={MariaDB ODBC 3.1
Driver};SERVER=127.0.0.1;PORT=3306;DATABASE=test;TCPIP=1;CHARSET=utf8mb4;UID=test'
    as user null password '12345'
    into CODE_SEX, NAME, NAME_EN
  do
    suspend;
end
```

### 10.2.5. Ограничение использования входных параметров

Для работы именных параметров подсистема EDS (EXTERNAL DATA SOURCE) использует внутренний предварительный анализ запросов, который заменяет все параметры вида `:<name>` на ? и сохраняет привязку имени параметра к его номеру. Поэтому это работает только для запросов синтаксис которых схож с синтаксисом Firebird. Например, для запросов `CALL` именованные параметры работать не будут. В этом случае необходимо использовать безымянные параметры.

Firebird при выполнении `prepare` получает типы, размеры и другие свойства входных и выходных параметров запросов. Далее исходя из этих данных строятся входные и выходные сообщения, выделяются буферы для обмена данными.

Не все ODBC драйверы поддерживают описание входных параметров функцией `SQLDescribeParam`. Некоторые ODBC драйверы формально поддерживают эту функцию, а на деле описание не соответствует действительности. Например ODBC драйвер для MySQL для всех входных параметров возвращает тип `SQL_VARCHAR` с длинной 255 символов.

# Глава 11. RSA-UDR — функции безопасности для подписания документов и проверки подписей

HQbird включает RSA-UDR, который содержит набор полезных функций безопасности. Эти функции могут быть полезны для защиты документов (хранящихся в виде VARCHAR, BLOB и даже внешних файлов) с помощью цифровых подписей.

**Важно**

RSA-UDR не требуется при использовании Firebird 4.0 и выше, поскольку эти функции встроены.

**Внимание!**

Чтобы использовать функции RSA-UDR, вам необходимо зарегистрировать их в вашей базе данных с помощью соответствующего SQL скрипта. Скрипт расположен в `plugin/UDR/crypto.sql`

Рассмотрим функции из этой библиотеки и примеры их использования.

| Функция | Описание |
|---|---|
| BIN2HEX | Преобразование двоичного представления в HEX (шестнадцатеричное) |
| BIN2HEXB | Преобразование двоичного BLOB в шестнадцатеричное представление |
| CRC32 | Вычисляет контрольную сумму |
| CRC32B | Вычисляет контрольную сумму BLOB |
| DECODE_BASE64 | Декодирует из Base64 |
| DECODE_BASE64B | Декодирует BLOB из Base64 |
| ENCODE_BASE64 | Кодирует в Base64 |
| ENCODE_BASE64B | Кодирует BLOB в Base64 |
| HEX2BIN | Преобразование шестнадцатеричного представления в двоичное |
| HEX2BINB | Преобразование шестнадцатеричного представления в двоичное для BLOB |
| MD5 | Вычисляет хэш MD5 |
| MD5B | Вычисляет хэш MD5 для BLOB |
| RSA_PUBLIC_KEY | Генерирует открытый ключ |
| RSA_PRIVATE_KEY | Генерирует закрытый ключ |
| RSA_SIGN | Подписывает объект |

| **Функция** | **Описание** |
|---|---|
| RSA_VERIFY | Проверяет подпись |
| SHA1 | Вычисляет хэш SHA |
| SHA1B | Вычисляет хэш SHA для BLOB |
| SHA256 | Вычисляет хэш SHA256 |
| SHA256B | Вычисляет хэш SHA256 для BLOB |

Рассмотрим на упрощенном примере, как использовать эти функции для подписания какого-либо документа и проверки подписи.

Для простоты мы поместим документ, закрытый ключ, открытый ключ, дайджест (хэш-сумму документа) и подпись в одну таблицу:

```
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
```

Рассмотрим пример:

```
-- Сначала подключимся к базе данных:

C:\HQbird\Firebird30>isql localhost:c:\temp\rsatest.fdb -user SYSDBA -pass masterkey
Database: localhost:c:\temp\rsatest.fdb, User: SYSDBA

-- а затем проверим, что функции зарегистрированы
SQL> show functions;
Global functions

Function Name Invalid Dependency, Type
=============================== ======= ====================================
RSA_PRIVATE_KEY
RSA_PUBLIC_KEY
RSA_SIGN
RSA_VERIFY
SHA256


-- очистим тестовую таблицу
SQL>delete from tbl
SQL>commit;
```

```
-- генерируем приватный ключ и записываем
-- записываем его в таблицу TBL (обычно приватный ключ хранится в секретном месте)

SQL>insert into tbl(PRIVATE_KEY) values(rsa_private_key(1024));

-- генерируем публичный ключ
SQL>update tbl set PUBLIC_KEY = rsa_public_key(PRIVATE_KEY);

-- создаём BLOB документ
SQL>update TBL set DOC='testtesttest';

-- и вычисляем его дайджест
SQL>update tbl set digest = sha256(doc);

—- подписываем документ и сохраняем его подпись
SQL>update tbl set sign = rsa_sign(digest, PRIVATE_KEY, 8);

-- проверка подписи
SQL> select RSA_VERIFY(SIGN, DIGEST, PUBLIC_KEY, SALTLEN) from tbl;

RSA_VERIFY
==========
<true>
-- как видите подпись действительна

—- изменяем документ (BLOB)
SQL> update TBL set DOC='testtesttest222';

-- перевычисляем его дайджест
SQL> update tbl set digest = sha256(doc);

-- проверяем подпись
SQL> select rsa_verify(sign, digest, PUBLIC_KEY, 8) from tbl;

RSA_VERIFY
==========
<false>
-- мы видим что защищённый документ был изменён
```

Примеры использования функций `BIN2HEX` и `HEX2BIN`

```
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
```

## 11.1. Как использовать функции безопасности и преобразования RSA-UDR

В основном функции RSA-UDR позволяют подписывать электронные документы всех типов (DOC, PDF, XML, JPG, PNG и т. д.), а затем обнаруживать несанкционированные изменения.

Функции преобразования упрощают преобразования BIN→HEX и HEX→BIN, а также кодирование и декодирование Base64.

# Глава 12. SPLIT-UDR — процедуры разбиения строк по разделителю

HQbird включает в себя SPLIT-UDR, который содержит набор полезных хранимых процедур. Эти процедуры могут быть полезны для разделения строки (хранящейся в виде VARCHAR или BLOB) по разделителю.

**Замечание**

Чтобы использовать функции SPLIT-UDR, вам необходимо зарегистрировать их в вашей базе данных с помощью соответствующего SQL скрипта. Скрипт расположен в `plugin/UDR/split-udr.sql`

Для удобства процедуры разделения текста по разделителю упакованы в пакет `SPLIT_UTILS`.

Рассмотрим процедуры из этой библиотеки и примеры их использования.

| Имя процедуры | Описание |
|---|---|
| `SPLIT_BOOLEAN` | Разбивает строку по разделителю и возвращает `BOOLEAN` значения. |
| `SPLIT_SMALLINT` | Разбивает строку по разделителю и возвращает `SMALLINT` значения. |
| `SPLIT_INT` | Разбивает строку по разделителю и возвращает `INTEGER` значения. |
| `SPLIT_BIGINT` | Разбивает строку по разделителю и возвращает `BIGINT` значения. |
| `SPLIT_STR` | Разбивает строку по разделителю и возвращает `VARCHAR(8191)` значения. |
| `SPLIT_DATE` | Разбивает строку по разделителю и возвращает `DATE` значения. |
| `SPLIT_TIME` | Разбивает строку по разделителю и возвращает `TIME` значения. |
| `SPLIT_TIMESTAMP` | Разбивает строку по разделителю и возвращает `TIMESTAMP` значения. |
| `SPLIT_DOUBLE` | Разбивает строку по разделителю и возвращает `DOUBLE PRECISION` значения. |

Первый аргумент этих процедур — это `BLOB` с подтипом `TEXT`, второй — строка типа `VARCHAR(10)`. Тип выходного значения зависит от имени процедуры.

Рассмотрим простые примеры использования этих процедур.

```
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
```

Кроме того, SPLIT-UDR также содержит процедуры разделения текста на токены (текст разбивается по нескольким разделителям). Эти процедуры объявляются следующим образом:

```
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;
```

Входные параметры:

- `IN_TXT` - входной текст типа `BLOB SUB_TYPE TEXT` или `VARCHAR (8191)`
- `IN_SEPARATORS` - список разделителей (строка с символами-разделителями), если не указан, то используются разделители " \n\r\t,.?!:;/\|<>[]{}()@#$%^&*-+='"~`"

Пример:

```
  SELECT
    w.WORD
  FROM DOCS
  LEFT JOIN SPLIT_WORDS(DOCS.CONTENT) w ON TRUE
```

```
WHERE DOCS.DOC_ID = 4
```

# Глава 13. OCR-UDR — функции распознавания текста с изображений

HQbird включает OCR-UDR, который содержит функцию распознавания текста на изображениях.

UDR OCR основан на бесплатной библиотеке распознавания текста Tesseract OCR 4.1, выпущенной по лицензии Apache, версия 2.0.

Чтобы зарегистрировать OCR-UDR, вам необходимо выполнить следующий скрипт:

```
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.';
```

Обратите внимание! В `EXTERNAL NAME` после имени модуля и точки входа через "!" указывается название словаря.

Где `eng` словарь из каталога `tessdata`, который используется для распознавания. Здесь можно указать несколько словарей, например `rus+eng`. Если словарь отсутствует, то используется `eng`.

В дистрибутиве HQBird содержится ограниченное количество словарей. Полный список словарей доступен по адресам: https://github.com/tesseract-ocr/tessdata_best (высокое качество распознования) или https://github.com/tesseract-ocr/tessdata_fast (высокая скорость распознования).

Библиотека поддерживает распознавание текста с изображений в форматах PNG, JPEG, GIF.

## 13.1. Пример использования OCR-UDR

Предположим, у вас есть таблица `DOCS`, в которой хранятся сканы документов, определенная следующим образом:

```
CREATE TABLE DOCS (
    ID         BIGINT GENERATED BY DEFAULT AS IDENTITY,
    NAME       VARCHAR(127) NOT NULL,
    PIC        BLOB SUB_TYPE 0,
    TXT        BLOB SUB_TYPE TEXT,
    PROCESSED  BOOLEAN DEFAULT FALSE NOT NULL,
    CONSTRAINT PK_DOCS PRIMARY KEY (ID)
);
```

Изображения для распознавания сохраняются в поле `PIC`, распознанный текст — в поле `TXT`.

Чтобы распознать текст из одного изображения, вы можете использовать запрос:

```
SELECT GET_OCR_TEXT(pic, 200) as txt
FROM docs
WHERE docs.id = 2;
```

Следующий запрос распознает текст на изображениях и сохраняет его в поле `TXT`.

```
UPDATE DOCS
SET TXT = GET_OCR_TEXT(PIC, 200),
    PROCESSED = TRUE
WHERE PROCESSED IS FALSE;
```

# Глава 14. LK-JSON-UDR — сборка и анализ JSON

HQbird включает библиотеку UDR udr-lkJSON для создания и анализа JSON на стороне сервера. Библиотека распространяется с открытым исходным кодом по лицензии MIT и бесплатна для использования. Она написана на Free Pascal. Исходный код доступен по адресу https://github.com/mnf71/udr-lkJSON

## 14.1. Установка UDR lkJSON

Чтобы использовать библиотеку UDR lkJSON, вам необходимо зарегистрировать ее в своей базе данных. Для этого запустите один из скриптов `plugin/UDR/udrJSON.sql` или `plugin/UDR/udrJSON-utf8.sql`, в зависимости от кодировки вашей базы данных (первый будет работать для любой однобайтовой кодировки).

После установки UDR её можно проверить с помощью скрипта https://github.com/mnf71/udr-lkJSON/blob/main/verify.sql.

Скрипт вызывает функцию для анализа JSON и преобразования его обратно в строку. Если исходный JSON такой же, как только что собранный JSON, то все в порядке. В реальности строки не будут полностью совпадать, так как JSON собран без учета красивого форматирования. Но содержание должно быть идентичным.

Проверка происходит для двух наборов (процедура + функция):

- `js$func.ParseText` — анализ JSON, заданного как BLOB. `js$func.GenerateText` — сборка JSON с возвратом BLOB.
- `js$func.ParseString` — анализ JSON, заданного как `VARCHAR(N)`. `js$func.GenerateString` — сборка JSON, возвращающая `VARCHAR(N)`.

## 14.2. Как это работает?

Библиотека udr-lkJSON основана на бесплатной библиотеке lkJSON для генерации и анализа JSON. Поэтому, чтобы иметь хорошее представление о том, как работать с UDR-lkJSON, желательно ознакомиться с lkjson (см. https://sourceforge.net/projects/lkjson/).

При анализе JSON некоторые элементы могут быть простыми типами, существующими в Firebird (`INTEGER`, `DOUBLE PRECISION`, `VARCHAR (N)`, `BOOLEAN`), а некоторые сложные — объектами и массивами. Сложные объекты возвращаются как указатель на внутренний объект из библиотеки lkJSON. Указатель сопоставляется с доменом `TY$POINTER`. Этот домен определяется следующим образом:

```
CREATE DOMAIN TY$POINTER AS
CHAR(8) CHARACTER SET OCTETS;
```

Кроме того, если в JSON встречается `null`, то для простых типов он не будет возвращен! Вам

придется рассматривать это значение отдельно. Это связано с тем, что библиотека UDR-lkJSON просто копирует методы классов библиотеки lkJSON в пакеты PSQL. И, как вы знаете, простые типы в Pascal не имеют отдельного состояния для NULL.

# 14.3. Описание пакетов PSQL из UDR-lkJSON

## 14.3.1. Пакет JS$BASE

Заголовок этого пакета выглядит следующим образом:

```
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 - success */

  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
```

Как видно из комментария этот пакет является калькой с класса TlkJSONbase. Он содержит базовые функции для работы с JSON.

Функция Dispose предназначена для освобождения указателя на объект JSON. Указатели, подлежащие обязательному освобождению, являются результатом анализа или генерации JSON. Не следует вызывать его на промежуточных объектах при разборе или сборке JSON. Это требуется только для объекта верхнего уровня.

Функция Field возвращает указатель на поле объекта. Первый параметр — указатель на объект, второй — имя поля. Если поле не существует, функция вернет нулевой указатель

(Это не NULL, а x'0000000000000000').

Функция Count_ возвращает количество элементов в списке или количество полей в объекте. В качестве параметра задаётся указатель на объект или список.

Функция Child возвращает или устанавливает значение элемента по индексу Idx в объекте или списке Self. Если параметр Obj не указан, то он возвращает указатель на элемент по индексу Idx. Если указан Obj, то присваивает свое значение элементу с индексом Idx. Obj — это указатель на одного из потомков TlkJSONbase.

Функция Value_ возвращает или устанавливает значение в виде строки JSON (VARCHAR) для объекта, указанного в параметре Self. Если параметр Val не указан, то возвращается значение; в противном случае оно устанавливается.

Функция WideValue_ возвращает или устанавливает значение в виде строки JSON (BLOB SUB_TYPE TEXT) для объекта, указанного в параметре Self. Если параметр Val не указан, то возвращается значение; в противном случае оно устанавливается.

Функция SelfType возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как число, где

- 0 — jsBase
- 1 — jsNumber
- 2 — jsString
- 3 — jsBoolean
- 4 — jsNull
- 5 — jsList
- 6 — jsObject

Если параметр Self не задан, то вернёт 0.

Функция SelfTypeName возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как строка. Если параметр Self не задан, то вернёт 'jsBase'.

### 14.3.2. Пакет JS$BOOL

Заголовок этого пакета выглядит следующим образом:

```
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
```

Как видно из комментария этот пакет является калькой с класса `TlkJSONboolean`. Он предназначен для работы с типом `BOOLEAN`.

Функция `Value_` возвращает или устанавливает в значение логического типа для объекта заданного в параметре `Self`. Если параметр `Bool` не задан, то значение будет возвращено, если задан — установлено. Обратите внимание, `NULL` не возвращается и не может быть установлено этим методом, для этого существует отдельный пакет `JS$NULL`.

Функция `Generate` возвращает указатель на новый объект `TlkJSONboolean`, который представляет собой значение логического типа в JSON. Параметр `Self` — указатель на JSON объект на основе которого создаётся объект `TlkJSONboolean`. Значение логического типа указывается в параметре `Bool`.

Функция `SelfType` возвращает тип объекта для указателя заданного в параметре `Self`. Тип объекта возвращается как число. Если параметр `Self` не задан, то вернёт 3.

Функция `SelfTypeName` возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как строка. Если параметр Self не задан, то вернёт `'jsBoolean'`.

### 14.3.3. Пакет JS$CUSTLIST

Заголовок этого пакета выглядит следующим образом:

```
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
```

Как видно из комментария этот пакет является калькой с класса `TlkJSONcustomlist`. Этот тип является базовым при работе с объектами и списками. Все процедуры и функции этого пакета можно использовать как JSON типа объект, так и JSON типа список.

Процедура `ForEach` извлекает каждый элемент списка или каждое поле объекта из указателя на JSON заданного в `Self`. Возвращаются следующие значения:

- `Idx` — индекс элемента списка или номер поля в объекте. Начинается с 0.
- `Name` — имя очередного поля, если `Self` — объект. Или индекс элемента списка, начиная с 0, если `Self` — список.
- `Obj` — указатель на очередной элемент списка или поля объекта.

Функция `Field` возвращает указатель на поле по его имени из объекта заданного в `Self`. Вместо имени поля можно задать номер элемента в списке или номер поля. Нумерация начинается с 0.

Функция `Count_` возвращает количество элементов в списке или полей в объекте, заданного в параметре `Self`.

Функция `Child` возвращает или устанавливает значение для элемента с индексом `Idx` в объекте или списке `Self`. Индексация начинается с 0. Если параметр `Obj` не задан, то возвращает указатель на элемент с индексов `Idx`. Если `Obj` указан, то устанавливает его значение в элемент с индексов `Idx`. Обратите внимание `Obj` это указатель на один из потомков `TlkJSONbase`.

Функция `GetBoolean` возвращает логическое значение поля объекта или элемента массива с индексом `Idx`. Индексация начинается с 0.

Функция `GetDouble` возвращает значение с плавающей точкой поля объекта или элемента массива с индексом `Idx`. Индексация начинается с 0.

Функция `GetInteger` возвращает целочисленное значение поля объекта или элемента массива с индексом `Idx`. Индексация начинается с 0.

Функция `GetString` возвращает символьное значение (`VARCHAR`) поля объекта или элемента массива с индексом `Idx`. Индексация начинается с 0.

Функция `GetWideString` возвращает значение типа `BLOB SUN_TYPE TEXT` поля объекта или элемента массива с индексом `Idx`. Индексация начинается с 0.

Функции `GetBoolean`, `GetDouble`, `GetInteger`, `GetString`, `GetWideString` не могут вернуть `NULL`. Для обработки значения `NULL` существует отдельный набор функций в пакете `JS$NULL`.

### 14.3.4. Пакет JS$FUNC

Заголовок этого пакета выглядит следующим образом:

```
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
```

Этот пакет содержит набор функций для разбора JSON или преобразование JSON в строку.

Функция `ParseText` разбирает JSON заданный в виде строки типа `BLOB SUB_TYPE TEXT` в параметре `Text`. Если в параметре Conv передать значение TRUE, то текст JSON строки будет преобразован из кодировки UTF8 в обычную.

Функция `ParseString` разбирает JSON заданный в виде строки типа `VARCHAR(N)` в параметре `String`. Если в параметре Conv передать значение TRUE, то текст JSON строки будет преобразован из кодировки UTF8 в обычную.

Функция `GenerateText` возвращает JSON в виде строки типа `BLOB SUB_TYPE TEXT`. Если в параметре Conv передать значение `TRUE`, то текст возвращаемой этой функцией будет преобразован в UTF8.

Функция `GenerateString` возвращает JSON в виде строки типа `VARCHAR(N)`. Если в параметре Conv передать значение `TRUE`, то текст возвращаемой этой функцией будет преобразован в UTF8.

Функция `ReadableText` возвращает JSON в виде человекочитаемой строки типа `BLOB SUB_TYPE TEXT`. Параметр `Level` - задаёт количество отступов для первого уровня. Это требуется если генерируемая строка является частью другого JSON. Если в параметре Conv передать значение `TRUE`, то текст возвращаемой этой функцией будет преобразован в UTF8.

Использование параметра `Conv` со значением `TRUE` оставлено для совместимости с исходной библиотекой lkJSON. Особой необходимости в нем нет, так как внешние сервисы самостоятельно умеют конвертировать исходную строку в нужный для СУБД формат и обратно.

### 14.3.5. Пакет JS$LIST

Заголовок этого пакета выглядит следующим образом:

```
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, Dbl 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
```

Как видно из комментария этот пакет является калькой с класса `TlkJSONlist`. Он предназначен для работы со списком.

Процедура `ForEach` извлекает каждый элемент списка или каждое поле объекта из указателя на JSON заданного в `Self`. Возвращаются следующие значения:

- `Idx` — индекс элемента списка или номер поля в объекте. Начинается с 0.
- `Name` — имя очередного поля, если `Self` — объект. Или индекс элемента списка, начиная с

0, если Self — список.

- Obj — указатель на очередной элемент списка или поля объекта.

Функция Add_ добавляет новый элемент в конец списка, указатель на который указан в параметре Self. Добавляемый элемент указывается в параметре Obj, который должен быть указателем на один из потомков TlkJSONbase. Функция возвращает индекс вновь добавленного элемента.

Функция AddBoolean добавляет новый элемент логического типа в конец списка, указатель на который указан в параметре Self. Функция возвращает индекс вновь добавленного элемента.

Функция AddDouble добавляет новый элемент вещественного типа в конец списка, указатель на который указан в параметре Self. Функция возвращает индекс вновь добавленного элемента.

Функция AddInteger добавляет новый элемент целочисленного типа в конец списка, указатель на который указан в параметре Self. Функция возвращает индекс вновь добавленного элемента.

Функция AddString добавляет новый элемент строкового типа (VARCHAR(N)) в конец списка, указатель на который указан в параметре Self. Функция возвращает индекс вновь добавленного элемента.

Функция AddWideString добавляет новый элемент типа BLOB SUB_TYPE TEXT в конец списка, указатель на который указан в параметре Self. Функция возвращает индекс вновь добавленного элемента.

Функция Delete_ удаляет элемент из списка с индексом Idx. Функция возвращает 0.

Функция IndexOfObject возвращает индекс элемента в списке. Указатель на список задаётся в параметре Self. В параметре Obj задаётся указатель на элемент индекс которого определяется.

Функция Field возвращает указатель на поле по его имени из объекта заданного в Self. Вместо имени поля можно задать номер элемента в списке или номер поля. Нумерация начинается с 0.

Функция Count_ возвращает количество элементов в списке или полей в объекте, заданного в параметре Self.

Функция Child возвращает или устанавливает значение для элемента с индексом Idx в объекте или списке Self. Индексация начинается с 0. Если параметр Obj не задан, то возвращает указатель на элемент с индексов Idx. Если Obj указан, то устанавливает его значение в элемент с индексов Idx. Обратите внимание Obj это указатель на один из потомков TlkJSONbase.

Функция Generate возвращает указатель на новый объект TlkJSONlist, который представляет собой пустой список. Параметр Self — указатель на JSON объект на основе которого создаётся TlkJSONlist.

Функция SelfType возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как число. Если параметр Self не задан, то вернёт 5.

Функция SelfTypeName возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как строка. Если параметр Self не задан, то вернёт 'jsList'.

### 14.3.6. Пакет JS$METH

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
```

Как видно из комментария этот пакет является калькой с класса TlkJSONobjectmethod. Он представляет собой пару ключ — значение.

Функция MethodObjValue возвращает указатель на значение из пары ключ-значение, указанной в параметре Self.

Функция MethodName возвращает или устанавливает имя ключа для пары ключ-значение, указанной в параметре Self. Если параметр Name не указан, то возвращает имя ключа, если указан, то устанавливает новое имя ключа.

Функция MethodGenerate создаёт новую пару ключ-значение и возвращает указатель на неё. В параметре Name указывается имя ключа, в параметре Obj — указатель на значение ключа.

### 14.3.7. Пакет JS$NULL

Заголовок этого пакета выглядит следующим образом:

```
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
```

Как видно из комментария этот пакет является калькой с класса `TlkJSONnull`. Он предназначен для обработки значения `NULL`.

Функция `Value_` возвращает 0, если значение объекта в `Self` представляет собой значение `null` (`jsNull`), и 1 в противном случае.

Функция `Generate` возвращает указатель на новый объект `TlkJSONnull`, который представляет собой значение `null`. Параметр `Self` — указатель на JSON объект на основе которого создаётся `TlkJSONnull`.

Функция `SelfType` возвращает тип объекта для указателя заданного в параметре `Self`. Тип объекта возвращается как число. Если параметр `Self` не задан, то вернёт 4.

Функция `SelfTypeName` возвращает тип объекта для указателя заданного в параметре `Self`. Тип объекта возвращается как строка. Если параметр `Self` не задан, то вернёт `'jsNull'`.

### 14.3.8. Пакет JS$NUM

Заголовок этого пакета выглядит следующим образом:

```
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
```

Как видно из комментария этот пакет является калькой с класса `TlkJSONnumber`. Он предназначен для обработки числовых значений.

Функция Value_ возвращает или устанавливает в значение числового типа для объекта заданного в параметре Self. Если параметр Num не задан, то значение будет возвращено, если задан — установлено. Обратите внимание, NULL не возвращается и не может быть установлено этим методом, для этого существует отдельный пакет JS$NULL.

Функция Generate возвращает указатель на объект TlkJSONnumber, который представляет собой значение числового типа в JSON. Параметр Self — указатель на JSON объект на основе которого создаётся объект TlkJSONnumber. В параметре Num передаётся значение числового типа.

Функция SelfType возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как число. Если параметр Self не задан, то вернёт 1.

Функция SelfTypeName возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как строка. Если параметр Self не задан, то вернёт 'jsNumber'.

### 14.3.9. Пакет JS$OBJ

Заголовок этого пакета выглядит следующим образом:

```
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, Dbl 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
```

Как видно из комментария этот пакет является калькой с класса `TlkJSONobject`. Он предназначен для обработки объектных значений.

Функция `New_` создаёт и возвращает указатель на новый пустой объект. Если `UseHash` установлен в `TRUE` (значение по умолчанию), то для поиска полей внутри объекта будет использована HASH таблица, в противном случае поиск будет осуществляться простым перебором.

Функция `Dispose` предназначена для освобождения указателя на JSON объект. Указатели, которые надо принудительно освобождать, появляются в результате анализа или создания JSON. Не следует вызывать его для промежуточных объектов при разборе или сборке JSON.

Он требуется только для объекта верхнего уровня.

Процедура ForEach извлекает каждое поле объекта из указателя на JSON заданного в Self. Возвращаются следующие значения:

- Idx — индекс элемента списка или номер поля в объекте. Начинается с 0.
- Name — имя очередного поля, если Self — объект. Или индекс элемента списка, начиная с 0, если Self — список.
- Obj — указатель на пару ключ-значение (для обработки такой пары необходимо использовать пакет JS$METH).

Функция Add_ добавляет новое поле в объект, указатель на который указан в параметре Self. Добавляемый элемент указывается в параметре Obj, который должен быть указателем на один из потомков TlkJSONbase. Имя поля указывается в параметре Name. Функция возвращает индекс вновь добавленного поля.

Функция AddBoolean добавляет новое поле логического типа в объект, указатель на который указан в параметре Self. Имя поля указывается в параметре Name. Значение поля указывается в параметре Bool. Функция возвращает индекс вновь добавленного поля.

Функция AddDouble добавляет новое поле вещественного типа в объект, указатель на который указан в параметре Self. Имя поля указывается в параметре Name. Значение поля указывается в параметре Dbl. Функция возвращает индекс вновь добавленного поля.

Функция AddInteger добавляет новое поле целочисленного типа в объект, указатель на который указан в параметре Self. Имя поля указывается в параметре Name. Значение поля указывается в параметре Int_. Функция возвращает индекс вновь добавленного поля.

Функция AddString добавляет новое поле строкового типа (VARCHAR(N)) в объект, указатель на который указан в параметре Self. Имя поля указывается в параметре Name. Значение поля указывается в параметре Int_. Функция возвращает индекс вновь добавленного поля.

Функция AddWideString добавляет новое поле типа BLOB SUB_TYPE TEXT в объект, указатель на который указан в параметре Self. Имя поля указывается в параметре Name. Значение поля указывается в параметре Int_. Функция возвращает индекс вновь добавленного поля.

Функция Delete_ удаляет поле из объекта с индексом Idx. Функция возвращает 0.

Функция IndexOfName возвращает индекс поля по его имени. Указатель на объект задаётся в параметре Self. В параметре Obj задаётся указатель на элемент индекс которого определяется.

Функция IndexOfObject возвращает индекс значения поля в объекте. Указатель на объект задаётся в параметре Self. В параметре Obj задаётся указатель на значения поля индекс которого определяется.

Функция Field возвращает или устанавливает значение поля по его имени. Указатель на объект задаётся в параметре Self. Имя поля указывается в параметре Name. Вместо имени поля можно задать номер элемента в списке или номер поля. Нумерация начинается с 0. Если в параметре Obj указано значение отличное от NULL, то новое значение будет

прописано в поле, в противном случае функция вернёт указатель на значение поля.

Функция Count_ возвращает количество полей в объекте, заданного в параметре Self.

Функция Child возвращает или устанавливает значение для элемента с индексом Idx в объекте Self. Индексация начинается с 0. Если параметр Obj не задан, то возвращает указатель на элемент с индексов Idx. Если Obj указан, то устанавливает его значение в элемент с индексов Idx. Обратите внимание Obj это указатель на один из потомков TlkJSONbase.

Функция Generate возвращает указатель на объект TlkJSONobject, который представляет собой объект в JSON. Если UseHash установлен в TRUE (значение по умолчанию), то для поиска полей внутри объекта будет использована HASH таблица, в противном случае поиск будет осуществляться простым перебором. В параметре Self передаётся указатель на объект на основе которого создаётся новый объект типа TlkJSONobject.

Функция SelfType возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как число. Если параметр Self не задан, то вернёт 6.

Функция SelfTypeName возвращает тип объекта для указателя заданного в параметре Self. Тип объекта возвращается как строка. Если параметр Self не задан, то вернёт 'jsObject'.

Функция FieldByIndex возвращает или устанавливает свойство как пару ключ-значение по заданному индексу Idx. Указатель на объект задаётся в параметре Self. Для обработки пары ключ-значение необходимо использовать пакет JS$METH. Если в параметре Obj указано значение отличное от NULL, то новое значение будет поле будет записано по заданному индексу, в противном случае функция вернёт указатель на поле.

Функция NameOf возвращает имя поля по его индексу заданному в параметре Idx. Указатель на объект задаётся в параметре Self.

Функция GetBoolean возвращает логическое значение поля объекта с индексом Idx. Индексация начинается с 0.

Функция GetDouble возвращает значение с плавающей точкой поля объекта с индексом Idx. Индексация начинается с 0.

Функция GetInteger возвращает целочисленное значение поля объекта с индексом Idx. Индексация начинается с 0.

Функция GetString возвращает символьное значение (VARCHAR) поля объекта с индексом Idx. Индексация начинается с 0.

Функция GetWideString возвращает значение типа BLOB SUN_TYPE TEXT поля объекта с индексом Idx. Индексация начинается с 0.

Функция GetBooleanByName возвращает логическое значение поля объекта по его имени Name.

Функция GetDoubleByName возвращает значение с плавающей точкой поля объекта по его имени Name.

Функция GetIntegerByName возвращает целочисленное значение поля объекта по его имени Name.

Функция GetStringByName возвращает символьное значение (VARCHAR) поля объекта по его имени Name.

Функция GetWideStringByName возвращает значение типа BLOB SUN_TYPE TEXT поля объекта по его имени Name.

### 14.3.10. Пакет JS$PTR

Заголовок этого пакета выглядит следующим образом:

```
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
```

Этот пакет помогает следить за указателями, которые возникают при создании объектов JSON.

Функция New_ создаёт и возвращает указатель на новый пустой объект. Если в параметр UsePtr передано значение 'Tra', то указатель будет привязан к транзакции, и по её завершении он будет автоматически удалён. Если в параметр UsePtr передано значение 'Att', то указатель будет привязан к соединению, и при его закрытии он будет автоматически удалён. Если UseHash установлен в TRUE (значение по умолчанию), то для поиска полей внутри объекта будет использована HASH таблица, в противном случае поиск будет осуществляться простым перебором.

Функция Dispose удаляет указатель на JSON объект привязанный к транзакции или соединению. Если в параметр UsePtr передано значение 'Tra', то будет удалён указатель привязанный к транзакции. Если в параметр UsePtr передано значение 'Att', то будет удалён указатель привязанный к соединению.

Функция Tra возвращает указатель привязанный к транзакции.

Функция Att возвращает указатель привязанный к соединению.

Функция isNull проверяет не является ли указатель нулевым (с нулевым адресом). Нулевой

указатель возвращает функции js$func.ParseText и js$func.ParseString в случае некорректного JSON на входе, обращение к несуществующему полю через метод Field и другое. Эут функцию можно использовать для детектирования таких ошибок.

### 14.3.11. Пакет JS$STR

Заголовок этого пакета выглядит следующим образом:

```
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
```

Как видно из комментария этот пакет является калькой с класса TlkJSONstring. Он предназначен для обработки строковых значений.

Функция Value_ возвращает или устанавливает в значение строкового типа (VARCHAR(N)) для объекта заданного в параметре Self. Если параметр Str не задан, то значение будет возвращено, если задан — установлено. Обратите внимание, NULL не возвращается и не может быть установлено этим методом, для этого существует отдельный пакет JS$NULL.

Функция WideValue_ возвращает или устанавливает в значение типа BLOB SUB_TYPE TEXT для объекта заданного в параметре Self. Если параметр Str не задан, то значение будет возвращено, если задан — установлено. Обратите внимание, NULL не возвращается и не может быть установлено этим методом, для этого существует отдельный пакет JS$NULL.

Функция Generate возвращает указатель на объект TlkJSONstring, который представляет собой значение строкового типа в JSON. Параметр Self — указатель на JSON объект на основе которого создаётся новый объект TlkJSONstring. Строковое значение задаётся в параметре Str.

Функция WideGenerate возвращает указатель на объект TlkJSONstring, который представляет

собой значение строкового типа в JSON. Параметр `Self` — указатель на JSON объект для которого устанавливается длинное строковое значение (`BLOB SUB_TYPE TEXT`) в параметре `Str`. Значение параметра `Self` будет возвращено функцией, если оно отлично от `NULL`, в противном случае вернёт указатель на новый объект `TlkJSONstring`.

Функция `SelfType` возвращает тип объекта для указателя заданного в параметре `Self`. Тип объекта возвращается как число. Если параметр `Self` не задан, то вернёт 2.

Функция `SelfTypeName` возвращает тип объекта для указателя заданного в параметре `Self`. Тип объекта возвращается как строка. Если параметр `Self` не задан, то вернёт `'jsString'`.

# 14.4. Примеры

## 14.4.1. Сборка JSON

Для примера возьмём базу данных employee.

В примерах используется модифицированную базу данных employee преобразованная в кодировку UTF8.

Функция `MAKE_JSON_DEPARTMENT_TREE` выводит список подразделений в формате JSON в иерархическом виде.

```
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
  -- для каждого нового подразделения заполняем значение полей JSON объекта
  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());
  -- тут возможно ADDSTRING лучше, так как гарантированно сохранит точность
  JS$OBJ.ADDDOUBLE(C_DEP.JSON, 'budget', C_DEP.BUDGET);
  JS$OBJ.ADDSTRING(C_DEP.JSON, 'location', C_DEP.LOCATION);
  JS$OBJ.ADDSTRING(C_DEP.JSON, 'phone_no', C_DEP.PHONE_NO);
  -- в каждое подразделение добавляем список, в который будут
  -- вносится подчинённые подразделения
  JS$OBJ.ADD_(C_DEP.JSON, 'departments', JS$LIST.GENERATE());
  IF (C_DEP.PARENT_JSON IS NOT NULL) THEN
  BEGIN
    -- там где есть подразделения, есть и объект родительского объекта JSON
    -- получаем из этого родительского объекта поле со списком
    JSON_SUB_DEPS = JS$OBJ.FIELD(C_DEP.PARENT_JSON, 'departments');
    -- и добавляем в него текущее подразделение
    JS$LIST.ADD_(JSON_SUB_DEPS, C_DEP.JSON);
  END
END
-- генерируем JSON в виде текста
```

```
  JSON_TEXT = JS$FUNC.READABLETEXT(JSON);
  -- не забываем очистить указатель
  JS$OBJ.DISPOSE(JSON);
  RETURN JSON_TEXT;
  WHEN ANY DO
  BEGIN
    -- если была ошибка всё равно очищаем указатель
    JS$OBJ.DISPOSE(JSON);
    EXCEPTION;
  END
END
```

Здесь применена следующая хитрость: на самом верхнем уровне рекурсивного запроса используется указатель на ранее созданный корневой объект JSON. В рекурсивной части запроса, создаётся JSON объект для родительского подразделения `PARENT_JSON` и JSON объект для текущего подразделения `PARENT_JSON`. Таким образом, мы всегда знаем в какой JSON объект добавлять подчинённое подразделение.

Далее пробегаем циклом по курсору и на каждой итерации добавляем значения полей для текущего подразделения. Обратите внимание для того, чтобы добавить значение `NULL`, приходится использовать вызов `JS$NULL.GENERATE()`. Если вы не будете делать этого, то при вызове `JS$OBJ.ADDSTRING(C_DEP.JSON, 'head_dept', C_DEP.HEAD_DEPT)`, когда ` C_DEP.HEAD_DEPT` равно `NULL` поле `head_dept` просто не будет добавлено.

Также для каждого подразделения необходимо добавить JSON список, в который будут добавляться подчинённые подразделения.

Если JSON объект родительского подразделения не `NULL`, то получаем ранее добавленный для него список с помощью функции `JS$OBJ.FIELD` и добавляем в него текущий объект JSON.

Далее для JSON объекта самого верхнего уровня можно сгененрировать текст, после чего сам объект нам больше не нужен и необходимо очистить выделенный для него указатель с помощью функции `JS$OBJ.DISPOSE`.

Обратите внимание на блок обработки исключений `WHEN ANY DO`. Он обязателен, поскольку даже когда произошла нам надо освободить указатель, чтобы избежать утечки памяти.

### 14.4.2. Анализ JSON

Разбирать JSON несколько сложнее, чем собирать его. Дело в том, что вам надо учитывать, что на вход может поступить некорректный JSON, не только сам по себе, но и со структурой не отвечающей вашей логике.

Предположим у вас есть JSON в котором содержится список людей с их характеристиками.

Этот JSON выглядит следующим образом:

```
[
  {"id": 1, "name": "John"},
```

```
  {"id": 2, "name": null}
]
```

Напишем хранимую процедуру, которая возвращает список людей из этого 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);
  -- если JSON некорректный js$func.parsetext не сгененрирует исключение,
  -- а вернёт нулевой указатель
  -- поэтому надо обработать такой случай самостоятельно
  if (js$ptr.isNull(json)) then
    exception e_custom_error 'invalid json';
  -- Опять же функции из этой библиотеки не проверяют корректность типов элементов
  -- и не возвращают ошибку понятную. Нам надо проверить тот ли тип мы обрабатываем.
  -- Иначе js$list.foreach вернёт "Access violation"
  if (js$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$list.foreach(:json)
    as cursor c
  do
  begin
    -- Проверяем, что элемент массива - это объект, иначе
    -- js$obj.GetIntegerByName вернёт "Access violation"
    if (js$base.SelfTypeName(c.Obj) != 'jsObject') then
      exception e_custom_error 'Element of list is not object';
    -- js$obj.GetIntegerByName не проверяет существования элемента с заданным именем
    -- она просто молча вернёт 0!!!!! Надо самому проверить
    -- А js$obj.Field вернёт нулевой указатель
    if (js$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 ;^
```

Для проверки правильности выполните следующий запрос

```
select id, name
from parse_peoples_json( '[{"id": 1, "name": "John"}, {"id": 2, "name": null}]' )
```

Посмотрим внимательно на скрипт разбора JSON. Первая особенность состоит в том, что функция `js$func.parsetext` не сгенерирует исключение, если вместо JSON на вход подана любая другая строка. Она просто вернёт пустой указатель. Но, это не `NULL` как вам казалось, а указатель с содержимым `x'0000000000000000'`. Поэтому после выполнения этой функции надо проверить, а что же вам было возвращено, иначе вызовы последующий функций будут возвращать ошибку "Access violation".

Далее важно проверять, какого типа объект JSON был возвращён. Если на входе вместо списка окажется объект или любой другой тип, то вызов `js$list.foreach` вернёт "Access violation". То же самое произойдёт если вы вызовите любую другую функцию, которая ожидает указатель на другой, не предназначенный для неё тип.

Следующая особенность состоит в том, что всегда надо проверять наличие полей (свойств объекта). Если поля с заданным именем нет, то в некоторых случаях может быть возвращено не корректное значение (как в случае с `js$obj.GetIntegerByName`), в других приведёт к ошибке преобразования типа.

Обратите внимание, функции вроде `js$obj.GetIntegerByName` или `js$obj.GetStringByName` не могут вернуть значение `NULL`. Для распознавания значения `NULL`, вам надо проверять тип поля функцией `js$base.selftypename`.

Как и в случае со сборкой JSON не забывайте освобождать указатель на JSON верхнего уровня, а также делать это в блоке обработки исключений `WHEN ANY DO`.

Далее приведём пример разбора JSON, который был собран функцией `MAKE_JSON_DEPARTMENT_TREE` в примере выше. В тексте примера приведены комментарии поясняющие принцип разбора.

```
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);
  -- если JSON некорректный js$func.parsetext не сгененрирует исключение,
  -- а просто вернёт нулевой указатель
  -- поэтому надо обработать такой случай самостоятельно
  IF (JS$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, 'mngr_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
      -- получаем список подчинённых подразделений
      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;
      -- обходим этот список и рекурсивно вызываем для него процедуру извлечения
      -- информации о каждом подразделении
      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 ; ^
```

# Глава 15. NANODBC-UDR — работа с данными через ODBC

СУБД Firebird начиная с версии 2.5 имеет возможность работать с внешними данными через оператор EXECUTE STATEMENT .. ON EXTERNAL DATA SOURCE. К сожалению работа с внешними источниками данных ограничена только базами данных Firebird.

Для устранения этого недостатка была написана UDR nanodbc. Библиотека с полностью открытыми исходными кодами под лицензией MIT и свободна для использования. Она написана на языке C++. Исходный код доступен по адресу https://github.com/mnf71/udr-nanodbc

Библиотека основана на тонкой обёртке для C++ вокруг нативного C ODBC API https://github.com/nanodbc/nanodbc

UDR nanodbc для взаимодействия с ODBC драйверами предоставляет низкоуровневый интерфейс, который сложен в использовании.

Вместо UDR nanodbc вы можете воспользоваться более высокоуровневым и привычным интерфейсом `EXECUTE STATEMENT ON EXTERNAL`. Подробнее см. ODBCEngine

## 15.1. Установка UDR nanodbc

Библиотеку необходимо зарегистрировать в вашей базе данных. Для этого необходимо выполнить скрипт `plugin/UDR/nanodbc_install.sql`.

## 15.2. Как это работает?

UDR nanodbc основана на свободной библиотеке **nanodbc**, поэтому для полного понимания рекомендуем изучить API этой библиотеки в её исходных кодах и документации (см. https://github.com/mnf71/udr-nanodbc).

При работе с объектами библиотеки используются так называемые дескрипторы (указатели на объекты nanodbc). Указатели описываются доменом определённым как:

```
CREATE DOMAIN TY$POINTER AS
CHAR(8) CHARACTER SET OCTETS;
```

в Firebird 4.0 он может быть описан следующим способом

```
CREATE DOMAIN TY$POINTER AS BINARY(8);
```

После завершения работы с объектом указатель на него необходимо освободить при помощи функций release_(), которые расположены в соответствующих PSQL пакетах. Какой

пакет использовать зависит от типа объекта, указатель на который необходимо освободить.

В Firebird невозможно создать функцию не возвращающую результат, поэтому для C++ функций с типом возврата `void`, UDR функции возвращают тип описанный доменом `TY$NANO_BLANK`. Не имеет смысл анализировать результат таких функций. Домен `TY$NANO_BLANK` описан как:

```
CREATE DOMAIN TY$NANO_BLANK AS SMALLINT;
```

Перед началом работы с UDR необходимо провести инициализацию библиотеки nanodbc. Это делается с помощью вызова функции `nano$udr.initialize()`. А по завершению работу вызвать функцию финализации `nano$udr.finalize()`. Функцию `nano$udr.initialize()` рекомендуется вызывать в `ON CONNECT` триггере, а функцию `nano$udr.finalize()` в `ON DISCONNECT` триггере.

## 15.3. Описание PSQL пакетов из UDR nanodbc

### 15.3.1. Пакет NANO$UDR

Заголовок этого пакета выглядит следующим образом:

```
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
```

Пакет `NANO$UDR` содержит функции инициализации и финализации UDR.

Функция `initialize()` инициализирует UDR nanodbc. Эту функции рекомендуется вызывать в `ON CONNECT` триггере. Её необходимо вызывать перед первым обращением к любой другой функции из UDR nanodbc.

Функция `finalize()` завершает работу UDR nanodbc. После её вызова работа с UDR nanodbc невозможна. При вызове функция автоматически освобождает все ранее выделенные ресурсы. Эту функцию рекомендуется вызывать в `ON DISCONNECT` триггере.

Функция `expunge()` автоматически освобождает все ранее выделенные ресурсы (соединения, транзакции, подготовленные запросы, курсоры).

Функция `locale()` возвращает или устанавливает значение кодировки для соединений по умолчанию. Если параметр `set_locale` задан, то будет произведена установка новой кодировки, в противном случае функция вернёт значение текущей кодировки. Это необходимо для преобразования передаваемых и получаемых строк, перед и после обмена с источником ODBC. По умолчанию установлена кодировка cp1251.

Если изначально соединение с БД установлено с кодировкой `UTF8`, то можно установить utf8, согласно названиям iconv. Если с кодировкой `NONE`, то лучше переводить в свою языковую кодировку с помощью функций `convert_[var]char()`.

Функция `error_message()` возвращает текст последней ошибки.

### 15.3.2. Пакет NANO$CONN

Заголовок этого пакета выглядит следующим образом:

```
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
```

Пакет `NANO$CONN` содержит функции для установки и источником данных ODBC, а также получении некоторой информации о соединении.

Функция `connection()` устанавливает соединение с источником данных ODBC. Если не один параметр не задан, то функция вернёт указатель на объект "соединение". Непосредственно само соединение с источником данных ODBC можно выполнить позднее с помощью функции `connect_()`.

Параметры функции:

- `attr` задаёт строку подключения или так называемый DSN;
- `user_` задаёт имя пользователя;
- `pass` задаёт пароль;

- `timeout` задаёт тайм-аут простоя.

Функция `valid()` возвращает является ли указатель на объект соединения корректным.

Функция `release_()` освобождает указатель на соединение и все связанные с ним ресурсы (транзакции, подготовленные запросы, курсоры).

Функция `expunge()` освобождает все связанные с соединением ресурсы (транзакции, подготовленные запросы, курсоры).

Функция `allocate()` позволяет по требованию выделять дескрипторы для настройки среды и атрибутов ODBC до установления соединения с базой данных. Обычно пользователю не нужно делать этот вызов явно.

Функция `deallocate()` освобождает дескрипторы подключения.

Функция `txn_read_uncommitted()` возвращает числовую константу, которая требуется для установки уровня изолированности транзакции `READ UNCOMMITTED`.

Функция `txn_read_committed()` возвращает числовую константу, которая требуется для установки уровня изолированности транзакции `READ COMMITTED`.

Функция `txn_repeatable_read()` возвращает числовую константу, которая требуется для установки уровня изолированности транзакции `REPEATABLE READ`.

Функция `txn_serializable()` возвращает числовую константу, которая требуется для установки уровня изолированности транзакции `SERIALIZABLE`.

Функция `isolation_level()` устанавливает уровень изолированности для новых транзакций. Параметры:

- `conn` — указатель на объект соединения;
- `level_` — уровень изолированности транзакции, должно быть одним из чисел возвращаемых функциями `tnx_*`.

Функция `connect_()` устанавливает соединение с источником данных ODBC и привязывает его к переданному указателю на объект соединения.

Параметры функции:

- `conn` — указатель на объект соединения;
- `attr` задаёт строку подключения или так называемый DSN;
- `user_` задаёт имя пользователя;
- `pass` задаёт пароль;
- `timeout` задаёт тайм-аут простоя.

Функция `connected()` возвращает установлено ли соединение с базой данных для заданного указателя на объект соединения.

Функция `disconnect_()` отключается от базы данных. В качестве параметра передаётся указатель на объект соединения.

Функция `transactions()` возвращает количество активных транзакций для заданного соединения.

Функция `get_info()` возвращает различную информацию о драйвере или источнике данных. Это низкоуровневая функция является аналогом ODBC функции `SQLGetInfo`. Не рекомендуется использовать её напрямую. Параметры:

- `conn` — указатель на объект соединения;
- `info_type` — тип возвращаемой информации. Числовые константы с типами возвращаемой информации можно найти в https://github.com/microsoft/ODBC-Specification/blob/master/Windows/inc/sql.h

Функция `dbms_name()` возвращает имя СУБД к которой произведено подключение.

Функция `dbms_version()` возвращает версию СУБД к которой произведено подключение.

Функция `driver_name()` возвращает имя драйвера.

Функция `database_name()` возвращает имя базы данных к которой произведено подключение.

Функция `catalog_name()` возвращает имя каталога базы данных к которой произведено подключение.

### 15.3.3. Пакет NANO$TNX

Заголовок этого пакета выглядит следующим образом:

```
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
```

Пакет `NANO$TNX` содержит функции для явного управления транзакциями.

Функция `transaction_()` отключает отключает автоматическое подтверждение транзакции и стартует новую транзакцию с уровнем изолированности указанным в функции `NANO$CONN.isolation_level()`. Функция возвращает указатель на новую транзакцию.

Функция `valid()` возвращает является ли указатель на объект транзакции корректным.

Функция `release_()` освобождает указатель на объект транзакции. При освобождении указателя не подтверждённая транзакция откатывается и драйвер возвращает в режим автоматического подтверждения транзакций.

Функция `connection()` возвращает указатель на соединение для которого была запущена транзакция.

Функция `commit_()` производит подтверждение транзакции.

Функция `rollback_()` производит откат транзакции.

### 15.3.4. Пакет NANO$STMT

Заголовок этого пакета выглядит следующим образом:

```
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
```

Пакет NANO$STMT содержит функции для работы с SQL запросами.

Функция statement_() создаёт и возвращает указатель на объект SQL запрос.

Параметры:

- conn — указатель на объект соединения;
- query — текст SQL запроса;
- scrollable — является ли курсор прокручиваемым (если конечно оператор возвращает курсор), если не задан (значение NULL), то используется поведения ODBC драйвера по умолчанию;
- timeout — тайм-аут SQL оператора.

Если не указан ни один параметр, то возвращает указатель на вновь созданный объект SQL запроса, без привязки к соединению. Позже этот указатель можно связать с соединением и задать другие свойства запроса.

Функция valid() возвращает является ли указатель на объект SQL запроса корректным.

Функция release_() освобождает указатель на объект SQL запроса.

Функция connected() возвращает привязан ли запрос к подключению.

Функция connection() возвращает указатель на привязанное подключение.

Функция open_() открывает соединение и привязывает его к запросу.

Параметры:

- stmt — указатель на SQL запрос;
- conn — указатель на подключение.

Функция close_() закрывает открытый ранее запрос и очищает все выделенные запросом ресурсы.

Функция cancel() отменяет выполнение запроса.

Функция closed() возвращает является ли запрос закрытым.

Функция prepare_direct() подготавливает SQL запрос и привязывает его к указанному соединению.

Параметры:

- stmt — указатель на запрос;
- conn — указатель на соединение;
- query — текст SQL запроса;
- scrollable — является ли курсор прокручиваемым (если конечно оператор возвращает курсор), если не задан (значение NULL), то используется поведения ODBC драйвера по

умолчанию;

- `timeout` — тайм-аут SQL оператора.

Функция `prepare_()` подготавливает SQL запрос.

Параметры:

- `stmt` — указатель на запрос;
- `query` — текст SQL запроса;
- `scrollable` — является ли курсор прокручиваемым (если конечно оператор возвращает курсор), если не задан (значение `NULL`), то используется поведения ODBC драйвера по умолчанию;
- `timeout` — тайм-аут SQL оператора.

Функция `scrollable_()` возвращает или устанавливает будет ли курсор прокручиваемым.

Параметры:

- `stmt` — указатель на запрос;
- `usage_` — является ли курсор прокручиваемым (если конечно оператор возвращает курсор), если не задан (значение `NULL`), то возвращает текущее значение этого флага.

Функция timeout() устанавливает тайм-аут SQL запроса.

Функция `execute_direct()` подготавливает и выполняет SQL запрос. Функция возвращает указатель на набор данных (курсор), который можно обработать с помощью функций пакета `NANO$RSLT`.

Параметры:

- `stmt` — указатель на запрос;
- `conn` — указатель на соединение;
- `query` — текст SQL запроса;
- `scrollable` — является ли курсор прокручиваемым (если конечно оператор возвращает курсор), если не задан (значение `NULL`), то используется поведения ODBC драйвера по умолчанию;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1;
- `timeout` — тайм-аут SQL оператора.

Функция `just_execute_direct()` подготавливает и выполняет SQL запрос. Функция предназначена для выполнения SQL операторов не возвращающих данные (не открывающих курсор).

Параметры:

- `stmt` — указатель на запрос;

- `conn` — указатель на соединение;
- `query` — текст SQL запроса;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1;
- `timeout` — тайм-аут SQL оператора.

Функция `execute_()` выполняет подготовленный SQL запрос. Функция возвращает указатель на набор данных (курсор), который можно обработать с помощью функций пакета `NANO$RSLT`.

Параметры:

- `stmt` — указатель на подготовленный запрос;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1;
- `timeout` — тайм-аут SQL оператора.

Функция `just_execute()` выполняет подготовленный SQL запрос. Функция предназначена для выполнения SQL операторов не возвращающих данные (не открывающих курсор).

Параметры:

- `stmt` — указатель на подготовленный запрос;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1;
- `timeout` — тайм-аут SQL оператора.

Функция `procedure_columns()` возвращает описание выходного поля хранимой процедуры в виде набора данных `nano$rslt`.

Параметры:

- `stmt` — указатель на запрос;
- `catalog_` — имя каталога которому принадлежит ХП;
- `schema_` — имя схемы в которой находится ХП;
- `procedure_` — имя хранимой процедуры;
- `column_` — имя выходного столбца ХП.

Функция `affected_rows()` возвращает количество строк затронутых SQL оператором. Эту функцию можно вызывать после выполнения оператора.

Функция `columns()` возвращает количество столбцов возвращаемых SQL запросом.

Функция `parameters()` возвращает количество параметров SQL запроса. Эту функцию можно вызывать только после подготовки SQL запроса.

Функция `parameter_size()` возвращает размер параметра в байтах.

Параметры:

- `stmt` — указатель на подготовленный запрос;

- parameter_index — индекс параметра.

Функции семейства bind_<type>⋯ связывают значение с параметром, если СУБД поддерживает пакетные операции см. execute() параметр batch_operations, то количество передаваемых значений не ограничивается, в разумных пределах. В противном случае передается только первый введенный пакет значений. Само связывание происходит уже при вызове execute().

Функция bind_smallint() привязывает значение типа SMALLINT к SQL параметру. Параметры:

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- value_ — значение параметра.

Функция bind_integer() привязывает значение типа INTEGER к SQL параметру. Параметры:

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- value_ — значение параметра.

Функция bind_bigint() привязывает значение типа BIGINT к SQL параметру. Параметры:

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- value_ — значение параметра.

Функция bind_float() привязывает значение типа FLOAT к SQL параметру. Параметры:

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- value_ — значение параметра.

Функция bind_double() привязывает значение типа DOUBLE PRECISION к SQL параметру. Параметры:

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- value_ — значение параметра.

Функция bind_varchar() привязывает значение типа VARCHAR к SQL параметру. Используется для однобайтных кодировок. Параметры:

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- value_ — значение параметра;
- param_size — размер параметра (строки).

Функция `bind_char()` привязывает значение типа `CHAR` к SQL параметру. Используется для однобайтных кодировок. Параметры:

- `stmt` — указатель на подготовленный запрос;
- `parameter_index` — индекс параметра;
- `value_` — значение параметра;
- `param_size` — размер параметра (строки).

Функция `bind_u8_varchar()` привязывает значение типа `VARCHAR` к SQL параметру. Используется для строк в кодировке UTF8. Параметры:

- `stmt` — указатель на подготовленный запрос;
- `parameter_index` — индекс параметра;
- `value_` — значение параметра;
- `param_size` — размер параметра (строки).

Функция `bind_u8_char()` привязывает значение типа `CHAR` к SQL параметру. Используется для строк в кодировке UTF8. Параметры:

- `stmt` — указатель на подготовленный запрос;
- `parameter_index` — индекс параметра;
- `value_` — значение параметра;
- `param_size` — размер параметра (строки).

Функция bind_blob() привязывает значение типа `BLOB` к SQL параметру. Параметры:

- `stmt` — указатель на подготовленный запрос;
- `parameter_index` — индекс параметра;
- `value_` — значение параметра.

Функция `bind_boolean()` привязывает значение типа `BOOLEAN` к SQL параметру. Параметры:

- `stmt` — указатель на подготовленный запрос;
- `parameter_index` — индекс параметра;
- `value_` — значение параметра.

Функция `bind_date()` привязывает значение типа `DATE` к SQL параметру. Параметры:

- `stmt` — указатель на подготовленный запрос;
- `parameter_index` — индекс параметра;
- `value_` — значение параметра.

Функция `bind_time()` привязывает значение типа TIME к SQL параметру. Параметры:

- `stmt` — указатель на подготовленный запрос;

- parameter_index — индекс параметра;
- value_ — значение параметра.

При использовании bind_time() теряются миллисекунды в отличие от bind_timestamp().

Функция bind_timestamp() привязывает значение типа TIMESTAMP к SQL параметру. Параметры:

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- value_ — значение параметра.

Функция bind_null() привязывает значение типа NULL к SQL параметру. Нет принципиальной необходимости назначать значение NULL непосредственно для одного значения, если это не вытекает из логики обработки. Привязку NULL можно сделать и при вызове соответствующей функции bind_⋯ если ей передано значение NULL.

- stmt — указатель на подготовленный запрос;
- parameter_index — индекс параметра;
- batch_size — размер пакета (по умолчанию 1). Позволяет установить значение NULL для параметра с заданным индексом, сразу в нескольких элементах пакета.

Функция convert_varchar() преобразует значение типа VARCHAR в другую кодировку. Параметры:

- value_ — строковое значение;
- from_ — кодировка из которой надо перекодировать строку;
- to_ — кодировка в которую надо перекодировать строку;
- convert_size — задаёт размер входного буфера для конвертирования (для скорости), для UTF8 например должен быть количество символов * 4. Размер выходного буфера всегда равен размеру объявления returns (можно своих наделать функций), изменение размера зависит от того откуда и куда конвертируется строковое значение: однобайтовая кодировка в многобайтовую - возможно увеличение относительно convert_size и наоборот — уменьшение, если многобайтовая кодировка преобразуется в однобайтовую. Усечение результата всегда происходит по размеру получаемого параметра.

Это вспомогательная функция, предназначенная для конвертирования строк в желаемую кодировку, поскольку не всегда другая сторона ODBC может ответить в правильной кодировке.

Функция convert_char() преобразует значение типа CHAR в другую кодировку. Параметры:

- value_ — строковое значение;
- from_ — кодировка из которой надо перекодировать строку;
- to_ — кодировка в которую надо перекодировать строку;

- `convert_size` — задаёт размер входного буфера для конвертирования (для скорости), для `UTF8` например должен быть количество символов * 4. Размер выходного буфера всегда равен размеру объявления returns (можно своих наделать функций), изменение размера зависит от того откуда и куда конвертируется строковое значение: однобайтовая кодировка в многобайтовую — возможно увеличение относительно `convert_size` и наоборот — уменьшение, если многобайтовая кодировка преобразуется в однобайтовую. Усечение результата всегда происходит по размеру получаемого параметра.

Это вспомогательная функция, предназначенная для конвертирования строк в желаемую кодировку, поскольку не всегда другая сторона ODBC может ответить в правильной кодировке.

Функция `clear_bindings()` очищает текущий пакет значений для параметров. Вызов данной функции необходим при повторном использовании подготовленного оператора с новыми значениями.

Функция `describe_parameter()` заполняет буфер для описания параметра, то есть позволяет задать тип, размер и масштаб параметра. Параметры:

- `stmt` — указатель на подготовленный запрос;
- `idx` — индекс параметра;
- `type_` — тип параметра;
- `size_` — размер (для строк);
- `scale_` — масштаб.

Функция `describe_parameters()` отправляет этот буфер описания параметров в ODBC, фактически описывает параметры.

Функция `reset_parameters()` сбрасывает информацию о параметрах подготовленного запроса.

### 15.3.5. Пакет NANO$RSLT

Заголовок этого пакета выглядит следующим образом:

```
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
```

Пакет NANO$RSLT содержит функции для работы с набором данных возвращаемым SQL запросом.

Функция `valid()` возвращает является ли указатель на набор данных корректным.

Функция `release_()` освобождает указатель на набор данных.

Функция `connection()` возвращает указатель на соединение с базой данных.

Функция `rowset_size()` возвращает размер набора данных (сколько активных курсоров в наборе данных).

Функция `affected_rows()` возвращает количество строк затронутых оператором (выбрано в курсоре).

Функция `has_affected_rows()` возвращает есть ли хотя бы одна строка затронутая запросом.

Функция `rows_()` возвращает количество записей в открытом курсоре.

Функция `columns()` возвращает количество столбцов в текущем курсоре.

Функция `first_()` перемещает указатель текущего курсора на первую запись. Работает только для двунаправленных (прокручиваемых курсоров). Возвращает `true` если операция успешна.

Функция `last_()` перемещает указатель текущего курсора на последнюю запись. Работает только для двунаправленных (прокручиваемых курсоров). Возвращает `true` если операция успешна.

Функция `next_()` перемещает указатель текущего курсора на следующую запись. Возвращает `true` если операция успешна.

Функция `prior_()` перемещает указатель текущего курсора на предыдущую запись. Работает только для двунаправленных (прокручиваемых курсоров). Возвращает `true` если операция успешна.

Функция `move()` перемещает указатель текущего курсора на указанную запись. Работает только для двунаправленных (прокручиваемых курсоров). Возвращает `true` если операция успешна.

- `rslt` — указатель на подготовленный набор данных;
- `row_` — номер записи.

Функция `skip_()` перемещает указатель текущего курсора на указанное количество записей. Работает только для двунаправленных (прокручиваемых курсоров). Возвращает `true` если операция успешна.

- `rslt` — указатель на подготовленный набор данных;
- `row_` — сколько записей пропустить.

Функция `position_()` возвращает текущую позицию курсора.

Функция `at_end()` возвращает достиг ли указатель курсора последней записи.

Функция get_smallint() возвращает значение столбца типа SMALLINT.

- rslt — указатель на подготовленный набор данных;
- column_ — имя столбца или его номер 0..n-1.

Функция get_integer() возвращает значение столбца типа INTEGER.

- rslt — указатель на подготовленный набор данных;
- column_ — имя столбца или его номер 0..n-1.

Функция get_bigint() возвращает значение столбца типа BIGINT.

- rslt — указатель на подготовленный набор данных;
- column_ — имя столбца или его номер 0..n-1.

Функция get_float() возвращает значение столбца типа FLOAT.

- rslt — указатель на подготовленный набор данных;
- column_ — имя столбца или его номер 0..n-1.

Функция get_double() возвращает значение столбца типа DOUBLE PRECISION.

- rslt — указатель на подготовленный набор данных;
- column_ — имя столбца или его номер 0..n-1.

Функция get_varchar() возвращает значение столбца типа VARCHAR(256) CHARACTER SET NONE. Функция предназначена для однобайтовых кодировок.

- rslt — указатель на подготовленный набор данных;
- column_ — имя столбца или его номер 0..n-1.

Существует целое семейство этих функций с суффиксами. В зависимости от суффикса изменяется максимальный размер возвращаемой строки:

- _s - VARCHAR (64) CHARACTER SET NONE;
- _l - VARCHAR (1024) CHARACTER SET NONE;
- _xl - VARCHAR (8192) CHARACTER SET NONE;
- _xxl - VARCHAR (32765) CHARACTER SET NONE.

Скорость получения данных зависит от максимального размера строки. Так заполнение буфера для строки VARCHAR(32765) происходит в разы медленней, чем для строки VARCHAR(256), поэтому надо подбирать размер меньшего значения, если не нужно большего.

Функция get_char() возвращает значение столбца типа CHAR(256) CHARACTER SET NONE. Функция предназначена для однобайтовых кодировок.

- rslt — указатель на подготовленный набор данных;
- column_ — имя столбца или его номер 0..n-1.

Существует целое семейство этих функций с суффиксами. В зависимости от суффикса изменяется максимальный размер возвращаемой строки:

- `_s` - `CHAR (64) CHARACTER SET NONE`;
- `_l` - `CHAR (1024) CHARACTER SET NONE`;
- `_xl` - `CHAR (8192) CHARACTER SET NONE`;
- `_xxl` - `CHAR (32767) CHARACTER SET NONE`.

Скорость получения данных зависит от максимального размера строки. Так заполнение буфера для строки `CHAR(32765)` происходит в разы медленней, чем для строки `CHAR(256)`, поэтому надо подбирать размер меньшего значения, если не нужно большего.

Функция `get_u8_varchar()` возвращает значение столбца типа `VARCHAR(64) CHARACTER SET UTF8`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Существует целое семейство этих функций с суффиксами. В зависимости от суффикса изменяется максимальный размер возвращаемой строки:

- `_l` - `VARCHAR (256) CHARACTER SET UTF8`;
- `_xl` - `VARCHAR (2048) CHARACTER SET UTF8`;
- `_xxl` - `VARCHAR (8191) CHARACTER SET UTF8`.

Функция `get_u8_char()` возвращает значение столбца типа `CHAR(64) CHARACTER SET UTF8`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Существует целое семейство этих функций с суффиксами. В зависимости от суффикса изменяется максимальный размер возвращаемой строки:

- `_l` - `CHAR (256) CHARACTER SET UTF8`;
- `_xl` - `CHAR (2048) CHARACTER SET UTF8`;
- `_xxl` - `CHAR (8191) CHARACTER SET UTF8`.

Функция `get_blob()` возвращает значение столбца типа `BLOB`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `get_boolean()` возвращает значение столбца типа `BOOLEAN`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `get_date()` возвращает значение столбца типа `DATE`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `get_time()` возвращает значение столбца типа `TIME`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `get_timestamp()` возвращает значение столбца типа `TIMESTAMP`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `convert_varchar()` преобразует значение типа VARCHAR в другую кодировку.

Параметры:

- `value_` — строковое значение;
- `from_` — кодировка из которой надо перекодировать строку;
- `to_` — кодировка в которую надо перекодировать строку;
- `convert_size` — задаёт размер входного буфера для конвертирования. См. `nano$stmt.convert_[var]char`.

Существует целое семейство этих функций с суффиксами. В зависимости от суффикса изменяется максимальный размер возвращаемой строки:

- `_s` - `VARCHAR (64) CHARACTER SET NONE`;
- `_l` - `VARCHAR (1024) CHARACTER SET NONE`;
- `_xl` - `VARCHAR (8192) CHARACTER SET NONE`;
- `_xxl` - `VARCHAR (32765) CHARACTER SET NONE`.

Функция `convert_char()` преобразует значение типа CHAR в другую кодировку. Параметры:

- `value_` — строковое значение;
- `from_` — кодировка из которой надо перекодировать строку;
- `to_` — кодировка в которую надо перекодировать строку;
- `convert_size` — задаёт размер входного буфера для конвертирования. См. `nano$stmt.convert_[var]char`.

Существует целое семейство этих функций с суффиксами. В зависимости от суффикса изменяется максимальный размер возвращаемой строки:

- `_s` - `CHAR (64) CHARACTER SET NONE`;
- `_l` - `CHAR (1024) CHARACTER SET NONE`;
- `_xl` - `CHAR (8192) CHARACTER SET NONE`;

- `_xxl - CHAR (32765) CHARACTER SET NONE`.

Функция `unbind()` отвязывает буфер от заданного столбца. Особенность передачи больших типов данных в некоторых реализациях ODBC.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `is_null()` возвращает является ли значение столбца значением `NULL`.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `is_bound()` проверяет привязан ли буфер значений для заданного столбца.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `column_()` возвращает номер столбца по его имени.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `column_name()` возвращает имя столбца по его индексу.

- `rslt` — указатель на подготовленный набор данных;
- `index_` — номер столбца `0..n-1`.

Функция `column_size()` возвращает размер столбца. Для строковых полей количество символов.

Функция `column_decimal_digits()` возвращает точность для числовых типов.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `column_datatype()` возвращает тип столбца.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция column_datatype_name() возвращает имя типа столбца.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `column_c_datatype()` возвращает тип столбца как он закодирован в константах ODBC.

- `rslt` — указатель на подготовленный набор данных;
- `column_` — имя столбца или его номер `0..n-1`.

Функция `next_result()` переключает на следующий набор данных.

- `rslt` — указатель на подготовленный набор данных.

Функция `has_data()` возвращает есть ли данные в наборе данных.

- `rslt` — указатель на подготовленный набор данных.

### 15.3.6. Пакет NANO$FUNC

Заголовок этого пакета выглядит следующим образом:

```
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
```

Пакет `NANO$FUNC` содержит функции для работы с SQL запросами. Этот пакет является облегчённой версией пакета `NANO$STMT`. Особенность состоит в том, что выполняемые функции унаследовали поведение NANODBC без изменений и собственных доработок UDR в части обмена параметрами и значениями. Возможное направление использования: выполнение настроек ODBC соединения через выполнение SQL-команд (`just_execute⋯`), если поддерживается, логирование событий и т.п. простые операции.

Функция `execute_conn()` подготавливает и выполняет SQL запрос. Функция возвращает указатель на набор данных (курсор), который можно обработать с помощью функций пакета `NANO$RSLT`.

Параметры:

- `conn` — указатель на соединение;
- `query` — текст SQL запроса;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1;
- `timeout` — тайм-аут SQL оператора.

Функция `just_execute_conn()` подготавливает и выполняет SQL запрос. Функция предназначена для выполнения SQL операторов не возвращающих данные (не открывающих курсор). Указатель на объект SQL запрос не создается. Параметры:

- `conn` — указатель на соединение;
- `query` — текст SQL запроса;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1;
- `timeout` — тайм-аут SQL оператора.

Функция `execute_stmt()` выполняет подготовленный SQL запрос. Функция возвращает указатель на набор данных (курсор), который можно обработать с помощью функций пакета `NANO$RSLT`.

Параметры:

- `stmt` — указатель на подготовленный запрос;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1.

Функция `transact_stmt()` - выполняет ранее подготовленный SQL запрос, стартуя и завершая собственную (автономную) транзакцию. Функция возвращает указатель на набор данных (курсор), который можно обработать с помощью функций пакета `NANO$RSLT`. Параметры:

Параметры:

- `stmt` — указатель на подготовленный запрос;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1.

Функция just_transact_stmt() - выполняет ранее подготовленный SQL запрос, стартуя и завершая собственную (автономную) транзакцию. Функция предназначена для выполнения SQL операторов не возвращающих данные (не открывающих курсор).

Параметры:

- `stmt` — указатель на подготовленный запрос;
- `batch_operations` — количество пакетных операций. По умолчанию равно 1.

Функция `prepare_stmt()` подготавливает SQL запрос. Параметры:

- `stmt` — указатель на запрос;
- `query` — текст SQL запроса;
- `timeout` — тайм-аут SQL оператора.

# 15.4. Примеры

## 15.4.1. Выборка данных из таблицы Postgresql

В этом примере производится выборка из базы данных Postgresql. Текст блока снабжён комментариями для понимания происходящего.

```
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';
```

```
-- инициализация nanodbc
-- эту функцию можно вызывать в ON CONNECT триггере
nano$udr.initialize();

BEGIN
  -- соединение с источником данных ODBC
  conn = nano$conn.connection(conn_str);
  WHEN EXCEPTION nano$nanodbc_error DO
  BEGIN
    -- если соединение было неудачным
    -- вызываем функцию для завершения работы nanodbc
    -- вместо явного вызова в скрипте эту функцию можно вызывать
    -- в ON DISCONNECT триггере
    nano$udr.finalize();
    -- после чего можно пробросить исключение далее
    EXCEPTION;
  END
END

BEGIN
  -- выделяем указатель на SQL оператор
  stmt = nano$stmt.statement_(conn);
  -- подготавливаем запрос
  nano$stmt.prepare_(stmt, sql_txt);
  -- выполняем запрос
  -- функция возвращает указатель на набор данных
  rs = nano$stmt.execute_(stmt);
  -- пока в курсоре есть записи перемещаемся по нему вперёд
  while (nano$rslt.next_(rs)) do
  begin
    -- для каждого столбца необходимо в зависимости от его типа вызывать
    -- соответствующую функцию или функцию с типом в который возможно
    -- преобразование исходного столбца
    id = nano$rslt.get_integer(rs, 'id');
    -- обратите внимание, поскольку мы работает с UTF8 вызывается функция с u8
    name = nano$rslt.get_u8_char_l(rs, 'name');
    suspend;
  end

  -- освобождаем ранее выделенные ресурсы
  /*
  rs = nano$rslt.release_(rs);
  stmt = nano$stmt.release_(stmt);
  */
  -- вышеперечисленные функции можно опустить, поскольку
  -- вызов nano$conn.release_ автоматически освободит все
  -- привязанные к соединению ресурсы
  conn = nano$conn.release_(conn);
  -- вызываем функцию для завершения работы nanodbc
  -- вместо явного вызова в скрипте эту функцию можно вызывать в
  -- ON DISCONNECT триггере
```

```
    nano$udr.finalize();

    WHEN EXCEPTION nano$invalid_resource,
         EXCEPTION nano$nanodbc_error,
         EXCEPTION nano$binding_error
    DO
    BEGIN
      -- если произошла ошибка
      -- освобождаем ранее выделенные ресурсы
      /*
      rs = nano$rslt.release_(rs);
      stmt = nano$stmt.release_(stmt);
      */
    -- вышеперечисленные функции можно опустить, поскольку
    -- вызов nano$conn.release_ автоматически освободит все
    -- привязанные к соединению ресурсы
      conn = nano$conn.release_(conn);
      -- вызываем функцию для завершения работы nanodbc
      -- вместо явного вызова в скрипте эту функцию можно вызывать в ON DISCONNECT
триггере
      nano$udr.finalize();
      -- после чего можно пробросить исключение далее
      EXCEPTION;
    END
  END
END
```

### 15.4.2. Вставка данных в таблицу Postgresql

В этом примере производится вставка новой строки в таблицу. Текст блока снабжён комментариями для понимания происходящего.

```
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 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(?, ?)';

  -- инициализация nanodbc
  -- эту функцию можно вызывать в ON CONNECT триггере
```

```
nano$udr.initialize();

BEGIN
  -- соединение с источником данных ODBC
  conn = nano$conn.connection(conn_str);
  WHEN EXCEPTION nano$nanodbc_error DO
  BEGIN
    -- если соединение было неудачным
    -- вызываем функцию для завершения работы nanodbc
    -- вместо явного вызова в скрипте эту функцию можно вызывать
    -- в ON DISCONNECT триггере
    nano$udr.finalize();
    EXCEPTION;
  END
END

BEGIN
  -- выделяем указатель на SQL оператор
  stmt = nano$stmt.statement_(conn);
  -- подготавливаем запрос
  nano$stmt.prepare_(stmt, sql_txt);
  -- устанавливаем параметры запроса
  -- индекс начинается с 0!
  nano$stmt.bind_integer(stmt, 0, 4);
  nano$stmt.bind_u8_varchar(stmt, 1, 'Row 4', 4 * 20);
  -- выполняем оператор INSERT
  nano$stmt.just_execute(stmt);
  -- получаем количество затронутых строк
  aff_rows = nano$stmt.affected_rows(stmt);
  -- освобождаем ранее выделенные ресурсы
  conn = nano$conn.release_(conn);
  -- вызываем функцию для завершения работы nanodbc
  -- вместо явного вызова в скрипте эту функцию можно вызывать в
  -- ON DISCONNECT триггере
  nano$udr.finalize();

  WHEN EXCEPTION nano$invalid_resource,
       EXCEPTION nano$nanodbc_error,
       EXCEPTION nano$binding_error
  DO
  BEGIN
    -- освобождаем ранее выделенные ресурсы
    conn = nano$conn.release_(conn);
    -- вызываем функцию для завершения работы nanodbc
    -- вместо явного вызова в скрипте эту функцию можно вызывать в
    -- ON DISCONNECT триггере
    nano$udr.finalize();
    EXCEPTION;
  END
END
```

```
  suspend;
END
```

### 15.4.3. Пакетная вставка данных в таблицу Postgresql

Если СУБД и ODBC драйвер поддерживают пакетное выполнение запросов, то можно использовать batch операции.

```
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(?, ?)';

  -- инициализация nanodbc
  -- эту функцию можно вызывать в ON CONNECT триггере
  nano$udr.initialize();

  BEGIN
    -- соединение с источником данных ODBC
    conn = nano$conn.connection(conn_str);
    WHEN EXCEPTION nano$nanodbc_error DO
    BEGIN
      -- если соединение было неудачным
      -- вызываем функцию для завершения работы nanodbc
      -- вместо явного вызова в скрипте эту функцию можно вызывать
      -- в ON DISCONNECT триггере
      nano$udr.finalize();
      EXCEPTION;
    END
  END

  BEGIN
    -- выделяем указатель на SQL оператор
    stmt = nano$stmt.statement_(conn);
    -- подготавливаем запрос
    nano$stmt.prepare_(stmt, sql_txt);
    -- устанавливаем параметры запроса
    -- индекс начинается с 0!
    -- первая запись
    nano$stmt.bind_integer(stmt, 0, 5);
    nano$stmt.bind_u8_varchar(stmt, 1, 'Row 5', 4 * 20);
    -- вторая запись
```

```
    nano$stmt.bind_integer(stmt, 0, 6);
    nano$stmt.bind_u8_varchar(stmt, 1, 'Row 6', 4 * 20);
    -- выполняем оператор INSERT, с размером пакета 2
    nano$stmt.just_execute(stmt, 2);
    -- освобождаем ранее выделенные ресурсы
    conn = nano$conn.release_(conn);
    -- вызываем функцию для завершения работы nanodbc
    -- вместо явного вызова в скрипте эту функцию можно вызывать в
    -- ON DISCONNECT триггере
    nano$udr.finalize();

    WHEN EXCEPTION nano$invalid_resource,
         EXCEPTION nano$nanodbc_error,
         EXCEPTION nano$binding_error
    DO
    BEGIN
      -- освобождаем ранее выделенные ресурсы
      conn = nano$conn.release_(conn);
      -- вызываем функцию для завершения работы nanodbc
      -- вместо явного вызова в скрипте эту функцию можно вызывать в
      -- ON DISCONNECT триггере
      nano$udr.finalize();
      EXCEPTION;
    END
  END
END
```

### 15.4.4. Использование транзакций

```
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$pointer;
  DECLARE tnx ty$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(?, ?)';
  sql_txt2 = 'insert into t2(id, name) values(?, ?)';

  -- инициализация nanodbc
  -- эту функцию можно вызывать в ON CONNECT триггере
  nano$udr.initialize();

  BEGIN
```

```
  -- соединение с источником данных ODBC
  conn = nano$conn.connection(conn_str);
  WHEN EXCEPTION nano$nanodbc_error DO
  BEGIN
    -- если соединение было неудачным
    -- вызываем функцию для завершения работы nanodbc
    -- вместо явного вызова в скрипте эту функцию можно вызывать
    -- в ON DISCONNECT триггере
    nano$udr.finalize();
    EXCEPTION;
  END
END

BEGIN
  -- подготавливаем первый SQL запрос
  stmt = nano$stmt.statement_(conn);
  nano$stmt.prepare_(stmt, sql_txt);
  -- подготавливаем второй SQL запрос
  stmt2 = nano$stmt.statement_(conn);
  nano$stmt.prepare_(stmt2, sql_txt2);
  -- стартуем транзакцию
  tnx = nano$tnx.transaction_(conn);
  --выполняем первый запрос в рамках транзакции
  nano$stmt.bind_integer(stmt, 0, 8);
  nano$stmt.bind_u8_varchar(stmt, 1, 'Row 8', 4 * 20);
  nano$stmt.just_execute(stmt);
  --выполняем второй запрос в рамках транзакции
  nano$stmt.bind_integer(stmt2, 0, 1);
  nano$stmt.bind_u8_varchar(stmt2, 1, 'Row 1', 4 * 20);
  nano$stmt.just_execute(stmt2);
  -- подтверждаем транзакцию
  nano$tnx.commit_(tnx);

  -- освобождаем ранее выделенные ресурсы
  conn = nano$conn.release_(conn);
  -- вызываем функцию для завершения работы nanodbc
  -- вместо явного вызова в скрипте эту функцию можно вызывать в
  -- ON DISCONNECT триггере
  nano$udr.finalize();

  WHEN EXCEPTION nano$invalid_resource,
       EXCEPTION nano$nanodbc_error,
       EXCEPTION nano$binding_error
  DO
  BEGIN
    -- освобождаем ранее выделенные ресурсы
    -- в случаем ошибки неподтверждённая транзакция откатится автоматически
    conn = nano$conn.release_(conn);
    -- вызываем функцию для завершения работы nanodbc
    -- вместо явного вызова в скрипте эту функцию можно вызывать в
    -- ON DISCONNECT триггере
```

```
      nano$udr.finalize();
      EXCEPTION;
    END
  END
END
```

# Глава 16. HTTP Client UDR

Библиотека IBSurgeon HTTP Client UDR предназначена для работы с HTTP сервисами, например через REST-API. Для реализации HTTP клиента используется библиотека с открытыми исходными кодами libcurl, которая обеспечивает взаимодействие с Web-сервисами по протоколу HTTP методами 'GET', 'HEAD', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS', 'TRACE'. Кроме того, предоставляются дополнительные процедуры и функции для разбора HTTP заголовков, а также разбора на составные части URL адресов и их построения.

HTTP Client UDR является 100% бесплатной и с открытым исходным кодом, с лицензией IDPL.

Доступны версии для Windows и Linux: для Windows у нас есть готовые к использованию двоичные файлы, а для Linux необходимо собрать UDR из исходных кодов в зависимости от конкретного дистрибутива (у нас есть простая инструкция по сборке).

Библиотека разработана за счет гранта IBSurgeon www.ib-aid.com.

## 16.1. Установка HTTP Client UDR

Установочный пакет HQBird для Windows устанавливает клиент UDR Http, если выбран соответствующий параметр. В Linux вам рекомендуется собрать библиотеку самостоятельно из исходного кода.

Чтобы ваша база данных могла использовать библиотеку UDR Http Client, вам необходимо запустить в ней сценарий регистрации SQL, который находится в `{$fbroot}/plugins/udr/sql/http_client_install.sql`.

## 16.2. Сборка под Linux

Перед сборкой необходимо установить

В Ubuntu

```
sudo apt-get install libcurl4-openssl-dev
```

В CentOS

```
sudo yum install libcurl-devel
```

Теперь можно производить саму сборку.

```
git clone https://github.com/IBSurgeon/http_client_udr.git
cd http_client_udr
mkdir build; cd build
cmake ..
```

```
make
sudo make install
```

## 16.3. Пакет HTTP_UTILS

Все процедуры и функции для работы с библиотекой HTTP Client расположены в PSQL пакете HTTP_UTILS.

### 16.3.1. Процедура HTTP_UTILS.HTTP_REQUEST

Процедура HTTP_UTILS.HTTP_REQUEST предназначена для отправки HTTP запроса и получения HTTP ответа.

```
PROCEDURE HTTP_REQUEST (
  METHOD               D_HTTP_METHOD NOT NULL,
  URL                  VARCHAR(8191) NOT NULL,
  REQUEST_BODY         BLOB DEFAULT NULL,
  REQUEST_TYPE         VARCHAR(256) DEFAULT NULL,
  HEADERS              VARCHAR(8191) DEFAULT NULL,
  OPTIONS              VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE          SMALLINT,
  STATUS_TEXT          VARCHAR(256),
  RESPONSE_TYPE        VARCHAR(256),
  RESPONSE_BODY        BLOB,
  RESPONSE_HEADERS     BLOB SUB_TYPE TEXT
);
```

Входные параметры:

- METHOD - HTTP метод. Обязательный параметр. Возможны следующие значения 'GET', 'HEAD', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS', 'TRACE'.
- URL - URL адрес. Обязательный параметр.
- REQUEST_BODY - тело HTTP запроса.
- REQUEST_TYPE - тип содержимого тела запроса. Значение этого параметра передаётся в качестве заголовка Content-Type.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.

- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

Процедура HTTP_UTILS.HTTP_REQUEST является основной процедурой с помощью которой происходит общение с web-сервисами. Процедуры HTTP_UTILS.HTTP_GET, HTTP_UTILS.HTTP_HEAD, HTTP_UTILS.HTTP_POST, HTTP_UTILS.HTTP_PUT, HTTP_UTILS.HTTP_PATCH, HTTP_UTILS.HTTP_DELETE, HTTP_UTILS.HTTP_OPTIONS, HTTP_UTILS.HTTP_TRACE являются производными от HTTP_UTILS.HTTP_REQUEST. Внутри они вызывают HTTP_UTILS.HTTP_REQUEST с заполненным параметром METHOD, а также убираются лишние входные и выходные параметры, что упрощает обращение к web-ресурсу определённым HTTP методом.

Первые два параметра процедуры HTTP_UTILS.HTTP_REQUEST являются обязательными.

Тело запроса REQUEST_BODY позволяется не для всех HTTP методов. Если оно, есть то желательно также указывать параметр REQUEST_TYPE, который соответствует заголовку Content-Type.

В параметре HEADERS вы можете передать дополнительные заголовки в виде строки. Каждый заголовок должен быть разделён переводом строки.

В параметре OPTIONS вы можете передать дополнительные параметры для библиотеки CURL в виде CURLOPT_*=<value>. Каждый новый параметр должен быть отделён переводом строки.

Тело ответа всегда возвращается в двоичном виде, но вы можете преобразовать его в текст с нужной кодировкой с помощью CAST(RESPONSE_BODY AS BLOB SUB_TYPE TEXT ⋯).

Примеры использования:

```
SELECT
  R.STATUS_CODE,
  R.STATUS_TEXT,
  R.RESPONSE_TYPE,
  R.RESPONSE_HEADERS,
  CAST(R.RESPONSE_BODY AS BLOB SUB_TYPE TEXT CHARACTER SET UTF8) AS RESPONSE_BODY
FROM HTTP_UTILS.HTTP_REQUEST (
  'GET',
  'https://www.cbr-xml-daily.ru/latest.js'
) R;

SELECT
  R.STATUS_CODE,
  R.STATUS_TEXT,
  R.RESPONSE_TYPE,
  R.RESPONSE_HEADERS,
  CAST(R.RESPONSE_BODY AS BLOB SUB_TYPE TEXT CHARACTER SET UTF8) AS RESPONSE_BODY
FROM HTTP_UTILS.HTTP_REQUEST (
  -- method
  'POST',
```

```
  -- URL
  'https://suggestions.dadata.ru/suggestions/api/4_1/rs/suggest/party',
  -- query body
  trim('
{
    "query": "810702819220",
    "type": "INDIVIDUAL"
}
  '),
  -- content-type
  'application/json',
  -- headers
  q'{
Authorization: Token b81a595753ff53056468a939c034c96b49177db3
  }'
) R;
```

Пример задания параметров CURL:

```
SELECT
  R.STATUS_CODE,
  R.STATUS_TEXT,
  R.RESPONSE_TYPE,
  R.RESPONSE_HEADERS,
  CAST(R.RESPONSE_BODY AS BLOB SUB_TYPE TEXT CHARACTER SET UTF8) AS RESPONSE_BODY
FROM HTTP_UTILS.HTTP_REQUEST (
  'GET',
  'https://yandex.ru',
  NULL,
  NULL,
  NULL,
  q'{
CURLOPT_FOLLOWLOCATION=0
CURLOPT_USERAGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/112.0.0.0 Safari/537.36 OPR/98.0.0.0
  }'
) R;
```

### Поддерживаемые CURL опции

- CURLOPT_DNS_SERVERS
- CURLOPT_PORT
- CURLOPT_PROXY
- CURLOPT_PRE_PROXY
- CURLOPT_PROXYPORT
- CURLOPT_PROXYUSERPWD

- CURLOPT_PROXYUSERNAME
- CURLOPT_PROXYPASSWORD
- CURLOPT_PROXY_TLSAUTH_USERNAME
- CURLOPT_PROXY_TLSAUTH_PASSWORD
- CURLOPT_PROXY_TLSAUTH_TYPE
- CURLOPT_TLSAUTH_USERNAME
- CURLOPT_TLSAUTH_PASSWORD
- CURLOPT_TLSAUTH_TYPE
- CURLOPT_SSL_VERIFYHOST
- CURLOPT_SSL_VERIFYPEER
- CURLOPT_SSLCERT
- CURLOPT_SSLKEY
- CURLOPT_SSLCERTTYPE
- CURLOPT_CAINFO
- CURLOPT_TIMEOUT
- CURLOPT_TIMEOUT_MS
- CURLOPT_TCP_KEEPALIVE
- CURLOPT_TCP_KEEPIDLE
- CURLOPT_TCP_KEEPINTVL
- CURLOPT_CONNECTTIMEOUT
- CURLOPT_USERAGENT
- CURLOPT_FOLLOWLOCATION (значение по умолчанию 1)
- CURLOPT_MAXREDIRS (значение по умолчанию 50)

Список поддерживаемых опций зависит от того с какой версий `libcurl` происходила сборка библиотеки.

### 16.3.2. Процедура `HTTP_UTILS.HTTP_GET`

Процедура `HTTP_UTILS.HTTP_GET` предназначена для отправки HTTP запроса методом GET.

```
PROCEDURE HTTP_GET (
  URL                 VARCHAR(8191) NOT NULL,
  HEADERS             VARCHAR(8191) DEFAULT NULL,
  OPTIONS             VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE         SMALLINT,
  STATUS_TEXT         VARCHAR(256),
  RESPONSE_TYPE       VARCHAR(256),
```

```
  RESPONSE_BODY         BLOB,
  RESPONSE_HEADERS      BLOB SUB_TYPE TEXT
);
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.
- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

Пример использования:

```
SELECT
  R.STATUS_CODE,
  R.STATUS_TEXT,
  R.RESPONSE_TYPE,
  R.RESPONSE_HEADERS,
  CAST(R.RESPONSE_BODY AS BLOB SUB_TYPE TEXT CHARACTER SET UTF8) AS RESPONSE_BODY
FROM HTTP_UTILS.HTTP_GET('https://www.cbr-xml-daily.ru/latest.js') R;
```

### 16.3.3. Процедура HTTP_UTILS.HTTP_HEAD

Процедура HTTP_UTILS.HTTP_HEAD предназначена для отправки HTTP запроса методом HEAD.

```
  PROCEDURE HTTP_HEAD (
    URL                 VARCHAR(8191) NOT NULL,
    HEADERS             VARCHAR(8191) DEFAULT NULL,
    OPTIONS             VARCHAR(8191) DEFAULT NULL
  )
  RETURNS (
    STATUS_CODE         SMALLINT,
    STATUS_TEXT         VARCHAR(256),
    RESPONSE_TYPE       VARCHAR(256),
    RESPONSE_HEADERS    BLOB SUB_TYPE TEXT
  );
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.
- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_HEADERS - заголовки ответа.

### 16.3.4. Процедура HTTP_UTILS.HTTP_POST

Процедура HTTP_UTILS.HTTP_POST предназначена для отправки HTTP запроса методом POST.

```
PROCEDURE HTTP_POST (
  URL                 VARCHAR(8191) NOT NULL,
  REQUEST_BODY        BLOB DEFAULT NULL,
  REQUEST_TYPE        VARCHAR(256) DEFAULT NULL,
  HEADERS             VARCHAR(8191) DEFAULT NULL,
  OPTIONS             VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE         SMALLINT,
  STATUS_TEXT         VARCHAR(256),
  RESPONSE_TYPE       VARCHAR(256),
  RESPONSE_BODY       BLOB,
  RESPONSE_HEADERS    BLOB SUB_TYPE TEXT
);
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- REQUEST_BODY - тело HTTP запроса.
- REQUEST_TYPE - тип содержимого тела запроса. Значение этого параметра передаётся в качестве заголовка Content-Type.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.

- STATUS_TEXT - текст статуса ответа.
- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

### 16.3.5. Процедура HTTP_UTILS.HTTP_PUT

Процедура HTTP_UTILS.HTTP_PUT предназначена для отправки HTTP запроса методом PUT.

```
PROCEDURE HTTP_PUT (
  URL                 VARCHAR(8191) NOT NULL,
  REQUEST_BODY        BLOB DEFAULT NULL,
  REQUEST_TYPE        VARCHAR(256) DEFAULT NULL,
  HEADERS             VARCHAR(8191) DEFAULT NULL,
  OPTIONS             VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE         SMALLINT,
  STATUS_TEXT         VARCHAR(256),
  RESPONSE_TYPE       VARCHAR(256),
  RESPONSE_BODY       BLOB,
  RESPONSE_HEADERS    BLOB SUB_TYPE TEXT
);
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- REQUEST_BODY - тело HTTP запроса.
- REQUEST_TYPE - тип содержимого тела запроса. Значение этого параметра передаётся в качестве заголовка Content-Type.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.
- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

### 16.3.6. Процедура HTTP_UTILS.HTTP_PATCH

Процедура HTTP_UTILS.HTTP_PATCH предназначена для отправки HTTP запроса методом PATCH.

```
PROCEDURE HTTP_PATCH (
  URL                 VARCHAR(8191) NOT NULL,
  REQUEST_BODY        BLOB DEFAULT NULL,
  REQUEST_TYPE        VARCHAR(256) DEFAULT NULL,
  HEADERS             VARCHAR(8191) DEFAULT NULL,
  OPTIONS             VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE         SMALLINT,
  STATUS_TEXT         VARCHAR(256),
  RESPONSE_TYPE       VARCHAR(256),
  RESPONSE_BODY       BLOB,
  RESPONSE_HEADERS    BLOB SUB_TYPE TEXT
);
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- REQUEST_BODY - тело HTTP запроса.
- REQUEST_TYPE - тип содержимого тела запроса. Значение этого параметра передаётся в качестве заголовка Content-Type.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.
- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

### 16.3.7. Процедура HTTP_UTILS.HTTP_DELETE

Процедура HTTP_UTILS.HTTP_DELETE предназначена для отправки HTTP запроса методом DELETE.

```
PROCEDURE HTTP_DELETE (
  URL                 VARCHAR(8191) NOT NULL,
  REQUEST_BODY        BLOB DEFAULT NULL,
```

```
  REQUEST_TYPE          VARCHAR(256) DEFAULT NULL,
  HEADERS               VARCHAR(8191) DEFAULT NULL,
  OPTIONS               VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE           SMALLINT,
  STATUS_TEXT           VARCHAR(256),
  RESPONSE_TYPE         VARCHAR(256),
  RESPONSE_BODY         BLOB,
  RESPONSE_HEADERS      BLOB SUB_TYPE TEXT
);
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- REQUEST_BODY - тело HTTP запроса.
- REQUEST_TYPE - тип содержимого тела запроса. Значение этого параметра передаётся в качестве заголовка Content-Type.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.
- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

### 16.3.8. Процедура HTTP_UTILS.HTTP_OPTIONS

Процедура HTTP_UTILS.HTTP_OPTIONS предназначена для отправки HTTP запроса методом OPTIONS.

```
PROCEDURE HTTP_OPTIONS (
  URL                   VARCHAR(8191) NOT NULL,
  HEADERS               VARCHAR(8191) DEFAULT NULL,
  OPTIONS               VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE           SMALLINT,
  STATUS_TEXT           VARCHAR(256),
  RESPONSE_TYPE         VARCHAR(256),
  RESPONSE_BODY         BLOB,
  RESPONSE_HEADERS      BLOB SUB_TYPE TEXT
```

```
);
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.
- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

### 16.3.9. Процедура HTTP_UTILS.HTTP_TRACE

Процедура HTTP_UTILS.HTTP_TRACE предназначена для отправки HTTP запроса методом TRACE.

```
PROCEDURE HTTP_TRACE (
  URL                 VARCHAR(8191) NOT NULL,
  HEADERS             VARCHAR(8191) DEFAULT NULL,
  OPTIONS             VARCHAR(8191) DEFAULT NULL
)
RETURNS (
  STATUS_CODE         SMALLINT,
  STATUS_TEXT         VARCHAR(256),
  RESPONSE_TYPE       VARCHAR(256),
  RESPONSE_BODY       BLOB,
  RESPONSE_HEADERS    BLOB SUB_TYPE TEXT
);
```

Входные параметры:

- URL - URL адрес. Обязательный параметр.
- HEADERS - другие заголовки HTTP запроса. Каждый заголовок должен быть на новой строке, то есть заголовки разделяются символом перевода строки.
- OPTIONS - опции библиотеки CURL.

Выходные параметры:

- STATUS_CODE - код статуса ответа.
- STATUS_TEXT - текст статуса ответа.

- RESPONSE_TYPE - тип содержимого ответа. Содержит значения заголовка Content-Type.
- RESPONSE_BODY - тело ответа.
- RESPONSE_HEADERS - заголовки ответа.

### 16.3.10. Функция HTTP_UTILS.URL_ENCODE

Функция HTTP_UTILS.URL_ENCODE предназначена для URL кодирования строки.

```
  FUNCTION URL_ENCODE (
    STR VARCHAR(8191)
  )
  RETURNS VARCHAR(8191);
```

Пример использования:

```
SELECT
  HTTP_UTILS.URL_ENCODE('N&N') as encoded
FROM RDB$DATABASE;
```

### 16.3.11. Функция HTTP_UTILS.URL_DECODE

Функция HTTP_UTILS.URL_DECODE предназначена для URL декодирования строки.

```
  FUNCTION URL_DECODE (
    STR VARCHAR(8191)
  )
  RETURNS VARCHAR(8191);
```

Пример использования:

```
SELECT
  HTTP_UTILS.URL_DECODE('N%26N') as decoded
FROM RDB$DATABASE;
```

### 16.3.12. Процедура HTTP_UTILS.PARSE_URL

Процедура HTTP_UTILS.PARSE_URL предназначена для разбора URL на составные части, согласно спецификации RFC 3986.

Требование: минимальная версия libcurl 7.62.0.

```
  PROCEDURE PARSE_URL (
    URL                 VARCHAR(8191)
  )
```

```
  RETURNS (
    URL_SCHEME          VARCHAR(64),
    URL_USER            VARCHAR(64),
    URL_PASSWORD        VARCHAR(64),
    URL_HOST            VARCHAR(256),
    URL_PORT            INTEGER,
    URL_PATH            VARCHAR(8191),
    URL_QUERY           VARCHAR(8191),
    URL_FRAGMENT        VARCHAR(8191)
  );
```

Входные параметры:

- URL - URL адрес, в формате

```
<URL> ::=
<scheme>:[//[<user>:<password>@]<host>[:<port>]][/]<path>[?<query>][#<fragment>]
```

Выходные параметры:

- URL_SCHEME - схема, определяющая протокол.
- URL_USER - имя пользователя.
- URL_PASSWORD - пароль.
- URL_HOST - хост.
- URL_PORT - номер порта (1-65535) указанный в URL, если порт не указан, то возвращает NULL.
- URL_PATH - URL путь. Часть пути будет равна '/', даже если в URL-адресе не указан путь. URL-путь всегда начинается с косой черты.
- URL_QUERY - запрос (параметры).
- URL_FRAGMENT - фрагмент (якорь).

Пример использования:

```
SELECT
    URL_SCHEME,
    URL_USER,
    URL_PASSWORD,
    URL_HOST,
    URL_PORT,
    URL_PATH,
    URL_QUERY,
    URL_FRAGMENT
FROM HTTP_UTILS.PARSE_URL
('https://user:password@server:8080/part/put?a=1&b=2#fragment');
```

### 16.3.13. Функция HTTP_UTILS.BUILD_URL

Функция HTTP_UTILS.BUILD_URL собирает URL из составных частей, согласно спецификации RFC 3986.

Требование: минимальная версия libcurl 7.62.0.

```
FUNCTION BUILD_URL (
  URL_SCHEME           VARCHAR(64) NOT NULL,
  URL_USER             VARCHAR(64),
  URL_PASSWORD         VARCHAR(64),
  URL_HOST             VARCHAR(256) NOT NULL,
  URL_PORT             INTEGER DEFAULT NULL,
  URL_PATH             VARCHAR(8191) DEFAULT NULL,
  URL_QUERY            VARCHAR(8191) DEFAULT NULL,
  URL_FRAGMENT         VARCHAR(8191) DEFAULT NULL
)
RETURNS VARCHAR(8191);
```

Входные параметры:

- URL_SCHEME - схема, определяющая протокол.
- URL_USER - имя пользователя.
- URL_PASSWORD - пароль.
- URL_HOST - хост.
- URL_PORT - номер порта (1-65535) указанный в URL, если порт не указан, то возвращает NULL.
- URL_PATH - URL путь. Часть пути будет равна '/', даже если в URL-адресе не указан путь. URL-путь всегда начинается с косой черты.
- URL_QUERY - запрос (параметры).
- URL_FRAGMENT - фрагмент (якорь).

Результат: URL строка согласно спецификации RFC 3986, т.е. в формате

```
<URL> ::=
<scheme>:[//[<user>:<password>@]<host>[:<port>]][/]<path>[?<query>][#<fragment>]
```

Пример использования:

```
SELECT
  HTTP_UTILS.BUILD_URL(
    'https',
    NULL,
    NULL,
    'localhost',
```

```
    8080,
    '/',
    'query=database',
    'DB'
  ) AS URL
FROM RDB$DATABASE;
```

### 16.3.14. Функция HTTP_UTILS.URL_APPEND_QUERY

Функция HTTP_UTILS.URL_APPEND_QUERY предназначена для добавление параметров к URL адресу, при этом ранее существующая QUERY часть URL адреса сохраняется.

Требование: минимальная версия libcurl 7.62.0.

```
FUNCTION URL_APPEND_QUERY (
  URL                 VARCHAR(8191) NOT NULL,
  URL_QUERY           VARCHAR(8191),
  URL_ENCODE          BOOLEAN NOT NULL DEFAULT FALSE
)
RETURNS VARCHAR(8191);
```

Входные параметры:

- URL - URL адрес, в формате <URL> ::= <scheme>:[//[<user>:<password>@]<host>[:<port>]][/]<path>[?<query>][#<fragment>].
- URL_QUERY - добавляемые параметры или параметр.
- URL_ENCODE - если TRUE, то производиться URL кодирования добавляемого параметра URL_QUERY. Часть строки до первого знака = не кодируется.

Результат: URL адрес с добавленными параметрами.

Пример использования:

```
EXECUTE BLOCK
RETURNS (
  URL VARCHAR(8191)
)
AS
BEGIN
  URL = 'https://example.com/?shoes=2';
  URL = HTTP_UTILS.URL_APPEND_QUERY(URL, 'hat=1');
  URL = HTTP_UTILS.URL_APPEND_QUERY(URL, 'candy=N&N', TRUE);
  SUSPEND;
END
```

Результатом будет URL https://example.com/?shoes=2&hat=1&candy=N%26N.

### 16.3.15. Функция HTTP_UTILS.APPEND_QUERY

Функция HTTP_UTILS.APPEND_QUERY сборки значений параметров в единую строку. Далее эта строка может быть добавлена в URL адрес как параметры или передана в тело запроса, если запрос отправляется методом POST с Content-Type: application/x-www-form-urlencoded.

Требование: минимальная версия libcurl 7.62.0.

```
FUNCTION APPEND_QUERY (
  URL_QUERY             VARCHAR(8191),
  NEW_QUERY             VARCHAR(8191),
  URL_ENCODE            BOOLEAN NOT NULL DEFAULT FALSE
)
RETURNS VARCHAR(8191);
```

Входные параметры:

- URL_QUERY - существующие параметры к которым необходимо добавить новые. Если параметр URL_QUERY равен NULL, то результатом будет строка содержащая только добавляемые параметры.
- NEW_QUERY - добавляемые параметры или параметр.
- URL_ENCODE - если TRUE, то производиться URL кодирования добавляемого параметра NEW_QUERY. Часть строки до первого знака = не кодируется.

Результат: строка с добавленными параметрами.

Пример использования:

```
EXECUTE BLOCK
RETURNS (
  QUERY VARCHAR(8191)
)
AS
BEGIN
  QUERY = HTTP_UTILS.APPEND_QUERY(NULL, 'shoes=2');
  QUERY = HTTP_UTILS.APPEND_QUERY(QUERY, 'hat=1');
  QUERY = HTTP_UTILS.APPEND_QUERY(QUERY, 'candy=N&N', TRUE);
  SUSPEND;
END
```

Результатом будет строка shoes=2&hat=1&candy=N%26N.

### 16.3.16. Процедура HTTP_UTILS.PARSE_HEADERS

Процедура HTTP_UTILS.PARSE_HEADERS предназначена для анализа заголовков возвращаемых в HTTP ответе. Каждый заголовок процедура возвращает отдельной записью в параметре HEADER_LINE. Если заголовок имеет вид <header name>: <header value>, то наименование

заголовка возвращается в параметре HEADER_NAME, а значение - HEADER_VALUE.

```
  PROCEDURE PARSE_HEADERS (
    HEADERS               BLOB SUB_TYPE TEXT
  )
  RETURNS (
    HEADER_LINE           VARCHAR(8191),
    HEADER_NAME           VARCHAR(256),
    HEADER_VALUE          VARCHAR(8191)
  );
```

Входные параметры:

- HEADERS - HTTP заголовки.

Выходные параметры:

- HEADER_LINE - HTTP заголовок.
- HEADER_NAME - имя HTTP заголовка.
- HEADER_VALUE - значение HTTP заголовка.

Пример использования:

```
WITH
  T AS (
    SELECT
      RESPONSE_HEADERS
    FROM HTTP_UTILS.HTTP_GET (
      'https://www.cbr-xml-daily.ru/latest.js'
    )
  )
SELECT
  H.HEADER_LINE,
  H.HEADER_NAME,
  H.HEADER_VALUE
FROM
  T
  LEFT JOIN HTTP_UTILS.PARSE_HEADERS(T.RESPONSE_HEADERS) H ON TRUE;
```

### 16.3.17. Функция HTTP_UTILS.GET_HEADER_VALUE

Функция HTTP_UTILS.GET_HEADER_VALUE возвращает значение первого найденного заголовка с заданным именем. Если заголовок не найден, то возвращается NULL.

```
  FUNCTION GET_HEADER_VALUE (
    HEADERS               BLOB SUB_TYPE TEXT,
    HEADER_NAME           VARCHAR(256)
```

```
  )
  RETURNS VARCHAR(8191);
```

Входные параметры:

- HEADERS - HTTP заголовки.
- HEADER_NAME - имя HTTP заголовка.

Результат: значение первого найденного заголовка с заданным именем или NULL, если заголовок не найден.

Пример использования:

```
WITH
  T AS (
    SELECT
      RESPONSE_HEADERS
    FROM HTTP_UTILS.HTTP_GET (
      'https://www.cbr-xml-daily.ru/latest.js'
    )
  )
SELECT
  HTTP_UTILS.GET_HEADER_VALUE(T.RESPONSE_HEADERS, 'age') AS HEADER_VALUE
FROM T;
```

# 16.4. Примеры

## 16.4.1. Получение курсов валют

```
SELECT
  STATUS_CODE,
  STATUS_TEXT,
  RESPONSE_TYPE,
  RESPONSE_HEADERS,
  RESPONSE_BODY
FROM HTTP_UTILS.HTTP_REQUEST (
  'GET',
  'https://www.cbr-xml-daily.ru/latest.js'
);
```

## 16.4.2. Получение сведений о компании по ИНН

```
SELECT
  STATUS_CODE,
  STATUS_TEXT,
  RESPONSE_TYPE,
```

```
  RESPONSE_HEADERS,
  RESPONSE_BODY
FROM HTTP_UTILS.HTTP_REQUEST (
  'POST',
  'https://suggestions.dadata.ru/suggestions/api/4_1/rs/suggest/party',
  trim('
{
    "query": "810712829220",
    "type": "INDIVIDUAL"
}
  '),
  'application/json',
  q'{
Authorization: Token b81a595753ff53056469a939c064c96b49177db3
  }'
)
```

Токен намеренно изменён на нерабочий. Его необходимо получить при регистрации на сервисе dadata.ru.

# Глава 17. Firebird Streaming

Firebird streaming — это технология асинхронной публикации событий возникающих в процессе анализа журнала репликации.

Для обработки событий используется служба (демон) `fb_streaming`. Служба отслеживает новые файлы журнала репликации, анализирует их, и генерирует события, которые обрабатываются одним из плагинов.

Доступные плагины:

- `kafka_cdc_plugin` — публикация CDC (Change Data Capture) событий в Kafka;
- `rabbitmq_plugin` — публикация событий репликации в RabbitMQ;
- `mongodb_events_plugin` — сохранение событий репликации в MongoDB;
- `fts_lucene_plugin` — автоматическое обновление полнотекстовых индексов без триггеров (IBSurgeon Full Text Search UDR);
- `simple_json_plugin` — сохранение журналов репликации в формате JSON.

Технология Firebird Streaming разработана для работы с журналами репликации с форматом Firebird 4.0 и выше. С сегментами репликации Firebird 2.5 и 3.0 она работать не будет.

Служба `fb_streaming` и её плагины не входит в штатный дистрибутив HQBird 2024. Они могут быть предоставлена нашим клиентам по отдельному запросу.

## 17.1. Принцип работы fb_streaming

Служба (демон) `fb_streaming` проверяет содержимое директории, указанной для задачи в конфигурационном файле `fb_streaming.conf`, и если в этой директории присутствует ещё не обработанные файлы журнала репликации, то она анализирует их, и генерирует соответствующие события, которые обрабатываются указанным плагином. Последний номер обработанного сегмента репликации записывается в контрольный файл с именем `<database guid>`. Расположение этого файла можно указать в параметре конфигурации `lockDir`. Если этот параметр не указан, то по умолчанию контрольный файл `<database guid>` будет создан в директории, указанной в качестве директории для журналов архивов для задачи.

Файлы журналов репликации должны быть в архивном состоянии.

Контрольный файл необходим для восстановления работы после внезапного завершения работы службы (например выключили свет или перезагрузили сервер). Он содержит следующую информацию:

- контрольную информацию (GUID базы данных, размер и т.д.);
- номер последнего обработанного сегмента и позицию в нём;
- номер сегмента с самой старой активной транзакцией (транзакция может начаться в

одном сегменте, а завершиться в другом);

- список всех активных транзакций в виде пар `{tnxNumber, segmentNumber}`, где `segmentNumber` номер сегмента в котором транзакция началась.

Если произошла внештатная ситуация и служба `fb_streaming` была остановлена, то при следующем старте она читает контрольный файл, и перечитывает все сегменты репликации, начиная с номера сегмента с самой старой активной транзакцией, и заканчивая номером последнего обработанного сегмента. В процессе чтения этих сегментов `fb_streaming` повторяет все события из списка активных транзакций, после чего переходит в штатный режим работы.

Файл сегмента репликации удаляется, если его номер меньше чем номер сегмента с самой старой активной транзакцией.

В процессе анализа журналов репликации могут возникать следующие события:

- старт транзакции (START);
- выполнение первой фазы подтверждения двухфазной транзакции (PREPARE);
- подтверждение транзакции (COMMIT);
- откат транзакции (ROLLBACK);
- установка точки сохранения (SAVE);
- откат точки сохранения (UNDO);
- освобождение точки сохранения (RELEASE);
- установка значения генератора (SET SEQUENCE);
- выполнение SQL оператора (EXECUTE SQL). Такие события происходят только для DDL операторов;
- сохранение BLOB (BLOB);
- вставка новой записи в таблицу (INSERT);
- обновление записи в таблице (UPDATE);
- удаление записи из таблицы (DELETE).

Какие из них будут обработаны, а какие просто проигнорированы зависит от выбранного плагина.

## 17.2. Установка и запуск службы в Windows

Перейдите в каталог установки `fb_streaming`.

Выполните настойку в файле конфигурации `fb_streaming.conf`.

После настройки файла конфигурации, вы можете установить и запустить службу.

Для получения справки по использованию и установке службы наберите

```
fb_streaming help
```

Вам будет выведена информация по установке, удаления, старту и остановки службы:

```
================================================================================
 Firebird streaming service

 Copyright (c) 2023 IBSurgeon ltd.
================================================================================

usage: fb_streaming <command> [service_name]
Possible commands are:
    install            install service
    remove             remove service
    start              start service
    stop               stop service
```

После настройки файла конфигурации, вы можете установить и запустить службу, выполнив следующие команды:

```
c:\streaming>fb_streaming install
Success install service!

c:\streaming>fb_streaming start
Service start pending...
Service started successfully.
```

После установки вы можете управлять службой через графическую утилиту - службы (`services.msc`).

Для удаления службы выполните следующие команды:

```
c:\streaming>fb_streaming stop
Service is already stopped.

c:\streaming>fb_streaming remove
Service deleted successfully
```

Если у вас будут функционировать несколько служб `fb_streaming`, то при установке, запуске, остановке и удалении им надо дать уникальные имена.

## 17.3. Установка и запуск демона в Linux

Перейдите в каталог установки `fb_streaming`. В Linux это обычно директория `/opt/fb_streaming`.

Выполните настойку в файле конфигурации `fb_streaming.conf`.

После настройки файла конфигурации, вы можете установить и запустить службу, выполнив следующие команды:

```
sudo systemctl enable fb_streaming

sudo systemctl start fb_streaming
```

Для удаления службы выполните следующие команды:

```
sudo systemctl stop fb_streaming

sudo systemctl disable fb_streaming
```

## 17.4. Настройка конфигурации служба (демон) fb_streaming

Для конфигурации службы используется файл `fb_streaming.conf`, который должен быть расположен в корневой директории `fb_streaming`.

Синтаксис файла конфигурации точно такой же как для файлов конфигурации Firebird.

Символ # обозначает комментарий, то есть всё что следует за этим символом до конца строки игнорируется.

Структура файла конфигурации `fb_streaming.conf` выглядит так:

```
# logLevel = info

# pluginDir = $(root)/stream_plugins

task = <sourceDir_1>
{
# другие параметры задачи 1
}

task = <sourceDir_2>
{
# другие параметры задачи 2
}
```

Общие параметры службы (демона) `fb_streaming`:

- `logLevel` — уровень журналирования (по умолчанию info). Допускаются уровни журналирования: trace, debug, info, warning, error, critical, off.

- `pluginDir` — директория, где размещаются плагины `fb_streaming`. По умолчанию `$(root)/stream_plugins`. Макроподстановка `$(root)` обозначает корневую директорию службы (демона).

Далее следуют конфигурации задач. Один экземпляр службы может обслуживать сразу несколько задач. Каждая задача работает в своём потоке. На каждый каталог с файлами журнала репликации создаётся своя задача.

Здесь `sourceDir_N` — директория с файлами журнала репликации. Эти директории должны быть уникальны. Одну и ту же директорию нельзя обрабатывать более чем одной задачей.

Основные параметры задачи:

- `controlFileDir` — директория в которой будет создан контрольный файл (по умолчанию та же директория, что и `sourceDir`);
- `errorTimeout` — тайм-аут после ошибки, в секундах. По истечению этого таймаута будет произведено повторное сканирование сегментов и перезапуск задачи. По умолчанию равен 60 секунд;
- `database` — строка подключения к базе данных (обязательный);
- `username` — имя пользователя для подключения к базе данных;
- `password` — пароль для подключения к базе данных;
- `plugin` — плагин, который обрабатывает события, возникающие в процессе анализа журнала репликации (обязательный);
- `deleteProcessedFile` — удалять ли файл журнала после обработки (по умолчанию `true`).

Задача может содержать также другие параметры, специфичные для используемого плагина.

Плагины для обработки событий из журнала репликации расположены в динамических библиотеках. Имя файла динамической библиотеки зависит от операционной системы и строится следующим образом:

- для Windows `<имя плагина>.dll`
- для Linux `lib<имя плагина>.so`

Файл динамической библиотеки должен быть расположен в директории `stream_plugins`, или в той, что указана в параметре `pluginDir`.

## 17.5. Трюки при конфигурации Firebird

Обратите внимание: папка с архивами журналов репликации должна обрабатываться только одной задачей службы `fb_streaming`. Если вы хотите чтобы, одновременно работала логическая репликация или с архивом журналов работало несколько служб, то необходимо дублирование архивных журналов в разные директории.

Это можно сделать в файле `replication.conf` следующим образом

```
...
journal_archive_directory = <archiveDir>
journal_archive_command = "copy $(pathname) $(archivepathname) && copy $(pathname)
<archiveDirTask>
...
```

Здесь `archiveDir` — директория архивов для работы асинхронной репликации, `archiveDirTask` — директория архивов для работы задачи службы `fb_streaming`.

## 17.6. Плагин kafka_cdc_plugin

Плагин `kafka_cdc_plugin` предназначен для регистрации событий изменения данных в таблицах базы данных Firebird и их сохранения их в Kafka. Этот процесс также известен как Change Data Capture, далее CDC.

Каждое событие представляет собой документ в формате JSON, который максимально приближен к формату событий debezium. В следующей версии планируется добавить возможность сериализовать события в формате AVRO.

### 17.6.1. Как это работает

Служба (демон) `fb_streaming` проверяет содержимое директории, указанной для задачи в конфигурационном файле `fb_streaming.conf`, и если в этой директории присутствует ещё не обработанные файлы журнала репликации, то она анализирует их, и генерирует соответствующие события, которые обрабатываются указанным плагином. Последний номер обработанного сегмента репликации записывается в контрольный файл с именем `<database guid>`. Расположение этого файла можно указать в параметре конфигурации `lockDir`. Если этот параметр не указан, то по умолчанию контрольный файл `<database guid>` будет создан в директории, указанной в качестве директории для журналов архивов для задачи.

**Важно**: Файлы журналов репликации должны быть в архивном состоянии.

Контрольный файл необходим для восстановления работы после внезапного завершения работы службы (например выключили свет или перезагрузили сервер). Он содержит следующую информацию:

- контрольную информацию (GUID базы данных, размер и т.д.);
- номер последнего обработанного сегмента и позицию в нём;
- номер сегмента с самой старой активной транзакцией (транзакция может начаться в одном сегменте, а завершиться в другом);
- список всех активных транзакций в виде пар `{tnxNumber, segmentNumber}`, где `segmentNumber` номер сегмента в котором транзакция началась.

Если произошла внештатная ситуация и служба `fb_streaming` была остановлена, то при следующем старте она читает контрольный файл, и перечитывает все сегменты

репликации, начиная с номера сегмента с самой старой активной транзакцией, и заканчивая номером последнего обработанного сегмента. В процессе чтения этих сегментов `fb_streaming` повторяет все события из списка активных транзакций, после чего переходит в штатный режим работы.

Файл сегмента репликации удаляется, если его номер меньше чем номер сегмента с самой старой активной транзакцией.

Плагин `kafka_cdc_plugin` генерирует только события изменения данных для заданных таблиц для каждой из операции `INSERT`, `UPDATE` и `DELETE`. Для каждой активной транзакции существует свой буфер, в котором накапливаются эти события, и они отправляются в Kafka только по завершению транзакции, после чего буфер очищается.

События могут попадать в Kafka в двух форматах:

- JSON без схемы (параметры `key_converter_schemas_enable` и `value_converter_schemas_enable` равный `false`);
- JSON со схемой (параметры `key_converter_schemas_enable` и `value_converter_schemas_enable` равный `true`).

В будущих версиях планируется также добавить поддержку сериализации событий в формате AVRO, а также поддержку реестра схем.

### 17.6.2. Наименование топиков

По умолчанию `kafka_cdc_plugin` записывает события изменения для всех операций `INSERT`, `UPDATE` и `DELETE`, в одну тему Apache Kafka. `kafka_cdc_plugin` использует для имён топиков события изменения данных схему описанную в параметре `kafka_topic_cdc_event`, равную `$[topicPrefix].cdc`.

Здесь `$[topicPrefix]` — это подстановка, которая заменятся префиксом для топиков. Этот префикс задаётся параметром `kafka_topic_prefix`, по умолчанию он равен `fb_streaming`. Таким образом, по умолчанию все события изменения данных записываются в топик `fb_streaming.cdc` (после применения подстановки).

Существуют и другие подстановки. Подстановка `$[tableName]` обозначает имя таблицы, для которой генерируется событие. Таким образом, указав в конфигурации

```
kafka_topic_cdc_event = $[topicPrefix].cdc.$[tableName]
```

мы получаем для каждой таблицы свой топик событий изменения данных.

Примеры полученных топиков с событиями изменения данных:

```
fb_streaming.cdc.products
fb_streaming.cdc.products_on_hand
fb_streaming.cdc.customers
fb_streaming.cdc.orders
```

Имя топика для события изменения метаданных указывается в параметре `kafka_topic_transaction`. В этом параметре доступна только подстановка `$[topicPrefix]`.

### 17.6.3. Метаданные транзакции

Плагин `kafka_cdc_plugin` может генерировать события, которые представляют границы транзакций и дополняют сообщения о событиях изменения данных. По умолчанию генерация событий границ транзакций отключено. Её можно включить установив параметр `transaction_metadata_enable` в значение `true`.

Плагин `kafka_cdc_plugin` генерирует события границ транзакций `BEGIN` и `END` для каждой транзакции. События границ транзакций содержат следующие поля:

**`status`**

`BEGIN` или `END`;

**`id`**

идентификатор транзакции;

**`ts_ms`**

время события границы транзакции (событие `BEGIN` или `END`). Поскольку в журналах репликации данного времени нет, то записывается время когда событие обрабатывается плагином `kafka_cdc_plugin`;

**`event_count` (для события `END`)**

общее количество событий, созданных транзакцией;

**`data_collections` (для события `END`)**

массив пар элементов `data_collection` и `event_count`, указывающий количество событий, которые `kafka_cdc_plugin` генерирует для изменений, происходящих для коллекции данных.

*Пример 5. Пример*

```
{
    "status": "BEGIN",
    "id": 901,
    "ts_ms": 1713099401529,
    "event_count": null,
    "data_collections": null
}

{
    "status": "END",
    "id": 901,
    "ts_ms": 1713099404605,
    "event_count": 2,
    "data_collections": [
        {
```

```
            "data_collection": "fbstreaming.cdc.CUSTOMERS",
            "event_count": 2
        }
    ]
}
```

Если это не переопределено с помощью параметра `kafka_topic_transaction`, то `kafka_cdc_plugin` отправляет события транзакции в тему `$[topicPrefix].transaction`.

### Расширение события изменения данных

Когда метаданные транзакции включены (`transaction_metadata_enable = true`), Envelope сообщения с данными пополняется новым полем `transaction`. Это поле предоставляет информацию о каждом событии в виде совокупности полей:

| | |
|---|---|
| `id` | идентификатор транзакции; |
| `total_order` | абсолютная позиция события среди всех событий, созданных транзакцией; |
| `data_collection_order` | позиция события в коллекции, среди всех событий созданных транзакцией. |

*Пример 6. Пример*

```
{
    "before": null,
    "after": {
        "ID": 20,
        "FIRST_NAME": "Anne",
        "LAST_NAME": "Kretchmar",
        "EMAIL": "annek@noanswer.org"
    },
    "source": {
       ...
    },
    "op": "c",
    "ts_ms": 1713099401533,
    "transaction": {
        "id": 901,
        "total_order": 1,
        "data_collection_order": 1
    }
}
```

### 17.6.4. Data Change Events (События изменения данных)

Плагин `kafka_cdc_plugin` генерирует событие изменения данных для каждой операции `INSERT`, `UPDATE` и `DELETE` на уровне записи. Каждое событие содержит ключ и значение. Структура ключа и значения зависят от измененной таблицы.

Служба `fb_streaming` и плагин `kafka_cdc_plugin` созданы для непрерывных потоков сообщений о событиях. Однако структура этих событий может со временем меняться, и потребителям может быть сложно это обработать. Чтобы решить эту проблему, каждое событие содержит схему для своего содержимого или, если вы используете реестр схем, идентификатор схемы, который потребитель может использовать для получения схемы из реестра. Это делает каждое событие самоописываемым.

Следующий скелет JSON показывает четыре основные части события изменения. Поле схемы находится в событии изменения только тогда, когда вы настраиваете плагин `kafka_cdc_plugin` для её создания. Аналогично, ключ события и полезные данные события находятся в событии изменения, только если вы настроили плагин `kafka_cdc_plugin` для их создания. Если вы используете формат JSON и настроили `kafka_cdc_plugin` для создания всех четырех основных частей событий изменений, события изменений имеют следующую структуру:

```
{
  "schema": { ①
    ...
   },
  "payload": { ②
    ...
  }
}
{
  "schema": { ③
    ...
  },
  "payload": { ④
    ...
  }
}
```

*Таблица 3. Описание основных частей события изменения*

| Номер | Поле | Описание |
|---|---|---|
| 1 | schema | Первое поле `schema` является частью ключа события (event key). Оно определяет схему, которая описывает, что находится в части полезной нагрузки ключа события. Другими словами, первое поле `schema` описывает структуру первичного ключа или уникального ключа, если таблица не имеет первичного ключа, для измененной таблицы. Эта часть события будет публиковаться только если параметр `key_converter_schemas_enable = true`. |
| 2 | payload | Первое поле `payload` является частью ключа события. Он имеет структуру, описанную предыдущим полем `schema`, и содержит ключ для измененной записи. |
| 3 | schema | Второе поле `schema` является частью значения события (event value). Оно определяет схему, которая описывает, что находится в "полезной нагрузке" значения события. Другими словами, вторая `schema` описывает структуру измененной записи. Обычно эта схема содержит вложенные схемы. Эта часть события будет публиковаться только если параметр `value_converter_schemas_enable = true`. |
| 4 | payload | Второе поле `payload` является частью значения события. Он имеет структуру, описанную предыдущим полем `schema`, и содержит фактические данные для измененной записи. |

### Ключи события изменения

Для заданной таблицы ключ события изменения имеет структуру, содержащую поле для каждого столбца первичного ключа таблицы на момент создания события.

Рассмотрим таблицу `CUSTOMERS`, и пример ключа события изменения для этой таблицы.

```
CREATE TABLE CUSTOMERS (
  ID BIGINT GENERATED BY DEFAULT AS IDENTITY,
  FIRST_NAME VARCHAR(255) NOT NULL,
  LAST_NAME VARCHAR(255) NOT NULL,
  EMAIL VARCHAR(255) NOT NULL,
  CONSTRAINT PK_CUSTOMERS PRIMARY KEY(ID)
);
```

```
{
    "schema": { ①
        "type": "struct",
        "name": "fbstreaming.CUSTOMERS.Key", ②
```

```
        "optional": "false", ③
        "fields": [ ④
            {
                "type": "int64",
                "optional": false,
                "field": "ID"
            }
        ]
    },
    "payload": { ⑤
        "ID": 1
    }
}
```

*Таблица 4. Описание основных частей ключа события изменения*

| Элемент | Поле | Описание |
|---|---|---|
| 1 | `schema` | Схема, которая описывает, что находится в части полезных данных ключа. Схема будет включена в событие только если параметр конфигурации установлен как `key_converter_schemas_enable = true`. |
| 2 | `name` | Имя схемы, определяющей структуру `payload` части. Эта схема описывает структуру первичного ключа изменяемой таблицы. Имена схем имеют следующий формат `fbstreaming.<table_name>.Key`. |
| 3 | `optional` | Указывает, должен ли ключ события содержать значение в поле `payload`. В этом примере требуется значение в полезных данных ключа. Значение в поле полезных данных ключа является необязательным, если таблица не имеет первичного ключа. |
| 4 | `fields` | Описывает поля, которые ожидаются в полезных данных (`payload`), включая имя, тип и необязательность. |
| 5 | `payload` | Содержит ключ записи, для которой было создано это событие изменения. В этом примере ключ содержит одно поле `ID`, значение которого равно 1. |

### Значения событий изменения

Значение в событии изменения немного сложнее, чем ключ. Как и ключ, значение имеет раздел `schema` и раздел `payload`. Раздел `schema` содержит схему, описывающую структуру `Envelope` раздела `payload`, включая её вложенные поля. События изменения для операций, которые создают, обновляют или удаляют данные, имеют значение `payload` со структурой `Envelope`.

Рассмотрим ту же таблицу, которая использовалась для демонстрации примера ключа события изменения:

```
CREATE TABLE CUSTOMERS (
  ID BIGINT GENERATED BY DEFAULT AS IDENTITY,
  FIRST_NAME VARCHAR(255) NOT NULL,
  LAST_NAME VARCHAR(255) NOT NULL,
  EMAIL VARCHAR(255) NOT NULL,
  CONSTRAINT PK_CUSTOMERS PRIMARY KEY(ID)
);
```

Часть значения события изменения для изменения в этой таблице описывается для следующих операций:

- create events
- update events
- delete events

Для демонстрации этих событий запустим следующий набор запросов:

```
insert into CUSTOMERS (FIRST_NAME, LAST_NAME, EMAIL)
values ('Anne', 'Kretchmar', 'annek@noanswer.org');

commit;

update CUSTOMERS
set FIRST_NAME = 'Anne Marie';

commit;

delete from CUSTOMERS;

commit;
```

### Create events

В следующем примере показана часть значения события изменения, которое генерирует `fb_streaming` для операции, создающей данные в таблице `CUSTOMERS`:

```
{
    "schema": { ①
        "type": "struct",
        "fields": [
            {
                "type": "struct",
                "fields": [
                    {
                        "type": "int64",
                        "optional": false,
                        "field": "ID"
```

```
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "FIRST_NAME"
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "LAST_NAME"
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "EMAIL"
                }
            ],
            "optional": true,
            "name": "fbstreaming.CUSTOMERS.Value", ②
            "field": "before"
        },
        {
            "type": "struct",
            "fields": [
                {
                    "type": "int64",
                    "optional": false,
                    "field": "ID"
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "FIRST_NAME"
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "LAST_NAME"
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "EMAIL"
                }
            ],
            "optional": true,
            "name": "fbstreaming.CUSTOMERS.Value",
            "field": "after"
        },
        {
            "type": "struct",
```

```
            "fields": [
                {
                    "type": "string",
                    "optional": false,
                    "field": "dbguid"
                },
                {
                    "type": "int64",
                    "optional": false,
                    "field": "sequence"
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "filename"
                },
                {
                    "type": "string",
                    "optional": false,
                    "field": "table"
                },
                {
                    "type": "int64",
                    "optional": false,
                    "field": "tnx"
                },
                {
                    "type": "int64",
                    "optional": false,
                    "field": "ts_ms"
                }
            ],
            "optional": false,
            "name": "fbstreaming.Source", ③
            "field": "source"
        },
        {
            "type": "string",
            "optional": false,
            "field": "op"
        },
        {
            "type": "int64",
            "optional": true,
            "field": "ts_ms"
        }
    ],
    "optional": false,
    "name": "fbstreaming.CUSTOMERS.Envelope" ④
},
"payload": { ⑤
```

```
        "before": null, ⑥
        "after": { ⑦
            "ID": 1,
            "FIRST_NAME": "Anne",
            "LAST_NAME": "Kretchmar",
            "EMAIL": "annek@noanswer.org"
        },
        "source": { ⑧
            "dbguid": "{9D66A972-A8B9-42E0-8542-82D1DA5F1692}",
            "sequence": 1,
            "filename": "TEST.FDB.journal-000000001",
            "table": "CUSTOMERS",
            "tnx": 200,
            "ts_ms": 1711288254908
        },
        "op": "c", ⑨
        "ts_ms": 1711288255056  ⑩
    }
}
```

*Таблица 5. Описание основных частей события create*

| Элемент | Поле | Описание |
|---|---|---|
| 1 | `schema` | Схема, которая описывает, что находится в части `payload`. Схема значений события изменения одинакова для каждого события изменения, создаваемого `fb_streaming` для конкретной таблицы. Схема будет включена в событие только если параметр конфигурации установлен как `value_converter_schemas_enable = true`. |
| 2 | `name` | В разделе `schema` каждое поле `name` определяет имя схемы для полей `payload` части.<br><br>`fbstreaming.CUSTOMERS.Value` - это схема для полей `before` и `after` полезной нагрузки. Эта схема специфична для таблицы `CUSTOMERS`.<br><br>Имена схем для полей `before` и `after` имеют вид `<logicalName>.<tableName>.Value`, что гарантирует уникальность имени схемы в базе данных. Это означает, что при использовании конвертера Avro результирующая схема Avro для каждой таблицы в каждом логическом источнике имеет свою собственную эволюцию и историю. |

| Элемент | Поле | Описание |
|---|---|---|
| 3 | `name` | `fbstreaming.Source` — это схема поля `source` полезной нагрузки. Эта схема специфична для службы `fbstreaming` и плагина `kafka_cdc_plugin`. `fbstreaming` использует её для всех событий, которые он генерирует. |
| 4 | `name` | `fbstreaming.CUSTOMERS.Envelope` — это схема общей структуры полезных данных, где `fbstreaming` — имя службы, а `CUSTOMERS` — таблица. |
| 5 | `payload` | Фактические данные значения. Это информация, которую предоставляет событие изменения. |
| 6 | `before` | Необязательное поле, указывающее состояние записи до того, как произошло событие. Если поле `op` имеет значение `c` для события create, как в этом примере, то поле `before` имеет значение `null`, поскольку предыдущего состояния записи не существовало. |
| 7 | `after` | Необязательное поле, указывающее состояние строки после возникновения события. В данном примере поле `after` содержит значения столбцов `ID`, `FIRST_NAME`, `LAST_NAME` и `EMAIL` новой записи. |
| 8 | `source` | Обязательное поле, описывающее метаданные источника события. Это поле содержит информацию, которую вы можете использовать для сравнения этого события с другими событиями, с учетом происхождения событий, порядка их возникновения и того, были ли события частью одной и той же транзакции. Метаданные источника включают в себя:<br>• GUID базы данных<br>• Номер сегмента журнала репликации<br>• Имя файла сегмента журнала репликации<br>• Имя таблицы<br>• Номер транзакции, в которой произошло событие<br>• Время последней модификации файла сегмента журнала репликации |

| Элемент | Поле | Описание |
|---|---|---|
| 9 | `op` | Обязательное поле, описывающее тип операции события. В этом примере `c` указывает, что операция создала новую запись. Допустимые значения:<br>• `c` - create<br>• `u` - update<br>• `d` - delete |
| 10 | `ts_ms` | Отображает время, в которое `fbstreaming` записал событие в Kafka.<br><br>В объекте `source` значение `ts_ms` указывает время последней модификации файла сегмента журнала репликации (с некоторым приближением можно считать это время временем возникновения события в базе данных). Сравнивая значение `payload.source.ts_ms` со значением `payload.ts_ms`, вы можете определить задержку между обновлением исходной базы данных и `fbstreaming`. |

### Update events

Значение события изменения для операции *update* в примере таблицы `CUSTOMERS` имеет ту же схему, что и событие *create* для этой таблицы. Аналогично, полезная нагрузка значения события имеет ту же структуру. Однако полезная нагрузка значения события содержит разные значения в событии *update*. Вот пример значения события изменения в событии, которое `fb_streaming` генерирует для обновления в таблице `CUSTOMERS`:

```
{
    "schema": { ... },
    "payload": {
        "before": { ①
            "ID": 1,
            "FIRST_NAME": "Anne",
            "LAST_NAME": "Kretchmar",
            "EMAIL": "annek@noanswer.org"
        },
        "after": { ②
            "ID": 1,
            "FIRST_NAME": "Anne Marie",
            "LAST_NAME": "Kretchmar",
            "EMAIL": "annek@noanswer.org"
        },
        "source": { ③
            "dbguid": "{9D66A972-A8B9-42E0-8542-82D1DA5F1692}",
            "sequence": 2,
            "filename": "TEST.FDB.journal-000000002",
```

```
            "table": "CUSTOMERS",
            "tnx": 219,
            "ts_ms": 1711288254908
        },
        "op": "u", ④
        "ts_ms": 1711288256121  ⑤
    }
}
```

*Таблица 6. Описание основных частей события update*

| Элемент | Поле | Описание |
|---|---|---|
| 1 | `before` | Необязательное поле, указывающее состояние записи таблицы до того, как произошло событие. В значении события *update* поле `before` содержит имя поля для каждого столбца таблицы и значение, которое было в этом столбце до события *update*. В этом примере значение столбца `FIRST_NAME` — `Anne`. |
| 2 | `after` | Необязательное поле, указывающее состояние записи таблицы после возникновения события. Вы можете сравнить структуры `before` и `after`, чтобы определить, что именно было изменено. В этом примере значение столбца `FIRST_NAME` теперь равно `Anne Marie`. |
| 3 | `source` | Обязательное поле, описывающее метаданные источника события. Это поле содержит информацию, которую вы можете использовать для сравнения этого события с другими событиями, с учетом происхождения событий, порядка их возникновения и того, были ли события частью одной и той же транзакции. Метаданные источника включают в себя:<br>• GUID базы данных<br>• Номер сегмента журнала репликации<br>• Имя файла сегмента журнала репликации<br>• Имя таблицы<br>• Номер транзакции, в которой произошло событие |
| 4 | `op` | Обязательное поле, описывающее тип операции события. В этом примере `u` указывает, что операция обновила существующую запись таблицы. |

| Элемент | Поле | Описание |
|---|---|---|
| 5 | `ts_ms` | Отображает время, в которое `fbstreaming` записал событие в Kafka.<br><br>В объекте `source` значение `ts_ms` указывает время последней модификации файла сегмента журнала репликации (с некоторым приближением можно считать это время временем возникновения события в базе данных). Сравнивая значение `payload.source.ts_ms` со значением `payload.ts_ms`, вы можете определить задержку между обновлением исходной базы данных и `fbstreaming`. |

### Delete events

Значение в событии изменения *delete* имеет ту же часть схемы, что и события *create* и *update* для той же таблицы. Часть `payload` в событии *delete* для примера таблицы `CUSTOMERS` выглядит следующим образом:

```
{
    "schema": { ... },
    "payload": { ①
        "before": {
            "ID": 1,
            "FIRST_NAME": "Anne Marie",
            "LAST_NAME": "Kretchmar",
            "EMAIL": "annek@noanswer.org"
        },
        "after": null, ②
        "source": { ③
            "dbguid": "{9D66A972-A8B9-42E0-8542-82D1DA5F1692}",
            "sequence": 3,
            "filename": "TEST.FDB.journal-000000003",
            "table": "CUSTOMERS",
            "tnx": 258,
            "ts_ms": 1711288254908
        },
        "op": "d", ④
        "ts_ms": 1711288256152 ⑤
    }
}
```

*Таблица 7. Описание основных частей события delete*

| Элемент | Поле | Описание |
|---|---|---|
| 1 | `before` | Необязательное поле, указывающее состояние записи таблицы до того, как произошло событие. В события *delete* поле `before` содержит значения, которые были в записи до того, как она была удалена. |
| 2 | `after` | Необязательное поле, указывающее состояние записи таблицы после возникновения события. В значении события *delete* поле `after` имеет значение `null`, что означает, что записи больше не существует. |
| 3 | `source` | Обязательное поле, описывающее метаданные источника события. Это поле содержит информацию, которую вы можете использовать для сравнения этого события с другими событиями, с учетом происхождения событий, порядка их возникновения и того, были ли события частью одной и той же транзакции. Метаданные источника включают в себя:<br>• GUID базы данных<br>• Номер сегмента журнала репликации<br>• Имя файла сегмента журнала репликации<br>• Имя таблицы<br>• Номер транзакции, в которой произошло событие |
| 4 | `op` | Обязательное поле, описывающее тип операции события. В этом примере `d` указывает, что операция удалила запись таблицы. |
| 5 | `ts_ms` | Отображает время, в которое `fbstreaming` записал событие в Kafka.<br><br>В объекте `source` значение `ts_ms` указывает время последней модификации файла сегмента журнала репликации (с некоторым приближением можно считать это время временем возникновения события в базе данных). Сравнивая значение `payload.source.ts_ms` со значением `payload.ts_ms`, вы можете определить задержку между обновлением исходной базы данных и `fbstreaming`. |

### 17.6.5. Отображение типов данных

| Тип данных Firebird | Литерал типа | Примечание |
|---|---|---|
| BOOLEAN | boolean | |

| Тип данных Firebird | Литерал типа | Примечание |
|---|---|---|
| SMALLINT | int16 | |
| INTEGER | int32 | |
| BIGINT | int64 | |
| INT128 | string | |
| FLOAT | float32 | |
| DOUBLE PRECISION | float64 | |
| NUMERIC(N,M) | string | |
| DECIMAL(N,M) | string | |
| DECFLOAT(16) | string | |
| DECFLOAT(34) | string | |
| CHAR(N) | string | |
| VARCHAR(N) | string | |
| BINARY(N) | string | Каждый байт закодирован 16-ричной парой `XX`. |
| VARBINARY(N) | string | Каждый байт закодирован 16-ричной парой `XX`. |
| TIME | string | Строковое представление времени в формате `HH24:MI:SS.F`, где `F` - десятитысячные доли секунды. |
| TIME WITH TIME ZONE | string | Строковое представление времени в формате `HH24:MI:SS.F TZ`, где `F` - десятитысячные доли секунды, `TZ` - имя часового пояса. |
| DATE | string | Строковое представление даты в формате `Y-M-D`. |
| TIMESTAMP | string | Строковое представление даты и времени в формате `Y-M-D HH24:MI:SS.F`, где `F` - десятитысячные доли секунды. |
| TIMESTAMP WITH TIMEZONE | string | Строковое представление даты и времени в формате `Y-M-D HH24:MI:SS.F TZ`, где `F` - десятитысячные доли секунды, `TZ` - имя часового пояса. |
| BLOB SUB_TYPE TEXT | string | Для `before` значений всегда `null`, поскольку старое значения BLOB полей не хранятся в сегментах репликации. |
| BLOB SUB_TYPE 0 | string | Для `before` значений всегда `null`, поскольку старое значения BLOB полей не хранятся в сегментах репликации. Каждый байт закодирован 16-ричной парой `XX`. |

### 17.6.6. Запуск Change Data Capture

Подробно опишем шаги необходимые для запуска сбора изменений (Change Data Capture) на вашей базе данных:

1. Настройка Kafka
2. Настройка Firebird и подготовка базы данных
3. Настройка службы `fb_streaming` и плагина `kafka_cdc_plugin`
4. Запуск Kafka
5. Установка и старт службы `fb_streaming`
6. Старт публикации в базе данных

#### Настройка Kafka

Для тестирования плагина `kafka_cdc_plugin`, используется настроенная установка Kafka в docker. Для этого используется `docker-compose.yml` со следующим содержимым:

```
version: "2"

services:

  zookeeper:
    image: confluentinc/cp-zookeeper:7.2.1
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-server:7.2.1
    hostname: kafka
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
      - "9997:9997"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

```
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_CONFLUENT_LICENSE_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_CONFLUENT_BALANCER_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_JMX_PORT: 9997
      KAFKA_JMX_HOSTNAME: kafka

  kafka-ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    ports:
      - 8080:8080
    environment:
      DYNAMIC_CONFIG_ENABLED: 'true'
    volumes:
      - "d:\\docker\\kafka\\config.yml:/etc/kafkaui/dynamic_config.yaml"
```

Подключаемый файл `config.yml` содержит:

```
auth:
  type: DISABLED
kafka:
  clusters:
  - bootstrapServers: kafka:29092
    name: Kafka CDC Cluster
    properties: {}
    readOnly: false
rbac:
  roles: []
webclient: {}
```

### Настройка Firebird и подготовка базы данных

Теперь необходимо настроить асинхронную репликацию для вашей базы данных, для этого в файле `replication.conf` необходимо добавить следующие строчки:

```
database = c:\fbdata\5.0\test.fdb
{
   journal_directory = d:\fbdata\5.0\replication\test\journal
   journal_archive_directory = d:\fbdata\5.0\replication\test\archive
   journal_archive_command = "copy $(pathname) $(archivepathname) && copy $(pathname)
d:\fbdata\5.0\replication\test\kafka_source"
   journal_archive_timeout = 10
}
```

Обратите внимание: здесь происходит дублирование файлов архивов журналов, чтобы одновременно работала логическая репликация и задача по отправки событий в Kafka. Это

необходимо, поскольку файлы с архивами журналов удаляются после обработки и не могут быть использованы другой задаче.

Если журналы репликации не используются для самой репликации, а только необходимы для Change Data Capture, то конфигурацию можно упростить:

```
database = c:\fbdata\5.0\test.fdb
{
   journal_directory = d:\fbdata\5.0\replication\test\journal
   journal_archive_directory = d:\fbdata\5.0\replication\test\kafka_source
   journal_archive_timeout = 10
}
```

Теперь надо включить необходимые таблицы в публикацию. Для примера выше достаточно добавить в публикацию таблицу `CUSTOMERS`. Это делается следующим запросом:

```
ALTER DATABASE INCLUDE CUSTOMERS TO PUBLICATION;
```

или можно включить в публикацию сразу все таблицы базы данных:

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

#### Настройка службы `fb_streaming` и плагина `kafka_cdc_plugin`

Далее настроим конфигурацию `fb_streaming.conf` для того, чтобы `fb_streaming` автоматически отправлял события изменения данных в Kafka.

```
task = d:\fbdata\5.0\replication\test\kafka_source
{
    database = inet://localhost:3055/test
    username = SYSDBA
    password = masterkey
    deleteProcessedFile = true
    plugin = kafka_cdc_plugin
    dumpBlobs = true
    kafka_brokers = localhost:9092
    kafka_topic_prefix = fb_streaming
    kafka_topic_cdc_event = $[topicPrefix].cdc
    kafka_topic_transaction = $[topicPrefix].transaction
    key_cdc_events_enable = true
    key_converter_schemas_enable = true
    value_converter_schemas_enable = true
    transaction_metadata_enable = true
}
```

В Linux эта конфигурация будет выглядеть так:

```
task = /mnt/d/fbdata/5.0/replication/test/kafka_source
{
    database = inet://localhost:3055/test
    username = SYSDBA
    password = masterkey
    deleteProcessedFile = true
    plugin = kafka_cdc_plugin
    dumpBlobs = true
    kafka_brokers = localhost:9092
    kafka_topic_prefix = fb_streaming
    kafka_topic_cdc_event = $[topicPrefix].cdc
    kafka_topic_transaction = $[topicPrefix].transaction
    key_cdc_events_enable = true
    key_converter_schemas_enable = true
    value_converter_schemas_enable = true
    transaction_metadata_enable = true
}
```

Параметр `task` описывает задачу для выполнения службой `fb_streaming`. Он указывает папку, в которой расположены файлы сегментов репликации для их обработки плагином. Таких задач может быть несколько. Этот параметр является сложным и сам описывает конфигурацию конкретной задачи. Опишем параметры доступные для задачи выполняемой плагином `kafka_cdc_plugin`:

- `controlFileDir` — директория в которой будет создан контрольный файл (по умолчанию та же директория, что и `sourceDir`);
- `database` — строка подключения к базе данных (обязательный);
- `username` — имя пользователя для подключения к базе данных;
- `password` — пароль для подключения к базе данных;
- `plugin` — плагин, который обрабатывает события, возникающие в процессе анализа журнала репликации (обязательный);
- `deleteProcessedFile` — удалять ли файл журнала после обработки (по умолчанию `true`). Этот параметр полезно ставить в значение `false` для отладки, когда одни и те же журналы надо обработать многократно не удаляя их;
- `warn_tnx_not_found` — генерировать предупреждение вместо ошибки, если транзакция не найдена в сегментах репликации. Если этот параметр установлен в `true`, то в журнал `fb_streaming` будет записано соответствующее предупреждение, содержимое потерянной транзакции будет проигнорировано, и плагин продолжит свою работу. По умолчанию установлен в `false`;
- `errorTimeout` — тайм-аут после ошибки, в секундах. По истечению этого таймаута будет произведено повторное сканирование сегментов и перезапуск задачи. По умолчанию равен 60 секунд;
- `include_tables` — регулярное выражение, определяющие имена таблиц для которых необходимо отслеживать события;

- `exclude_tables` — регулярное выражение, определяющие имена таблиц для которых не надо отслеживать события;
- `dumpBlobs` — публиковать ли новые значения BLOB полей (по умолчанию `false`);
- `kafka_brokers` — адреса брокеров Kafka. Можно указать несколько адресов. Адреса разделяются запятой;
- `kafka_topic_prefix` — префикс топиков Kafka. Он задаёт макроподстановку `$[topicPrefix]`, которая может использоваться для имён топиков;
- `kafka_topic_cdc_event` — имя топика(ов) Kafka, в которой сохраняются события изменения данных. Могут использоваться макроподстановки `$[topicPrefix]` и `$[tableName]`;
- `kafka_topic_transaction` — имя топика Kafka, в которой сохраняются метаданные транзакций. Может использоваться макроподстановка `$[topicPrefix]`;
- `key_cdc_events_enable` - если эта опция установлена в `true`, то каждое событие изменения данных содержит информацию о ключе, в противном случае ключ события будет равен `null`. Это может быть полезно, поскольку Kafka использует ключ события для секционирования;
- `key_converter_schemas_enable` — включать ли в ключ события обновления данных схему;
- `value_converter_schemas_enable` — включать ли в значение события обновления данных схему;
- `transaction_metadata_enable` — отправлять ли события старта и завершения транзакции.

### Запуск Kafka

Теперь можно запустить docker с контейнером Kafka:

```
docker-compose up -d
```

Для остановки docker выполните:

```
docker-compose down
```

### Установка и старт службы `fb_streaming`

Следующим шагом необходимо установить и запустить службу `fb_streaming`.

В Windows это делается следующими командами (необходимы права Администратора):

```
fb_streaming install
fb_streaming start
```

В Linux:

```
sudo systemctl enable fb_streaming
```

```
sudo systemctl start fb_streaming
```

Для тестирования работы `fb_streaming` без установки службы просто наберите команду `fb_streaming` без аргументов. `fb_streaming` будет запущен как приложение и завершён после нажатия клавиши Enter.

### Старт публикации в базе данных

После того, как вы всё настроили и запустили, необходимо разрешить публикацию в вашей базе данных. Это делается следующим SQL запросом:

```
ALTER DATABASE ENABLE PUBLICATION;
```

С этого момента служба `fb_streaming` будет отслеживать изменения в указанных таблицах и публиковать в топике `fb_streaming.cdc, на серверах указных в `kafka_brokers`.

## 17.7. Плагин rabbitmq_plugin

Плагин `rabbitmq_plugin` предназначен для регистрации событий в базе данных Firebird и их отправки на сервер очередей RabbitMQ.

Каждое событие представляет собой документ в формате JSON.

В событии содержаться следующие сведения:

- тип события (`INSERT`, `UPDATE`, `DELETE`, `SET SEQUENCE`, `EXECUTE SQL`);
- для событий `INSERT`, `UPDATE`, `DELETE`:
  - имя таблицы, для которого произошло это событие;
  - номер транзакции в которой произошло это событие;
  - старые и новые значение полей записи таблицы;
  - массив изменившихся полей для события `UPDATE`;
  - новые значения BLOB полей;
- для события `SET SEQUENCE`:
  - имя последовательности (генератора);
  - значение последовательности;
- для события `EXECUTE SQL`:
  - номер транзакции в которой произошло это событие;
  - текст SQL запроса.

В событии `INSERT` присутствуют только новые значения полей.

В событии `UPDATE` присутствуют как старые, так и новые значения полей.

В событии DELETE присутствуют только старые значения полей.

Событие EXECUTE SQL возникает только для DDL запросов.

Для BLOB полей всегда сохраняется BlobId. Если параметр конфигурации dumpBlobs, то в дополнение к этому для событий INSERT и UPDATE будут публиковаться новые значения BLOB полей. Для старых значений полей это не возможно, поскольку старые значения BLOB полей отсутствуют в файлах журнала репликации.

### 17.7.1. Алгоритм работы плагина rabbitmq_plugin

Каждое событие упаковывается в формате JSON и отправляется в обменник сервера очередей RabbitMQ, указанный в параметре exchange_name. В зависимости от типа обменника, событие либо отправляется во все привязанные к обменнику очереди, либо в очередь, привязанную по ключу маршрутизации routing_key. Если указан параметр queue_name, то очередь с заданным именем декларируется и привязывается по ключу routing_key к обменнику exchange_name.

- служба fb_streaming анализирует директорию с журналами репликации, установленную для задачи, на наличие необработанных файлов;
- для ещё необработанных файлов репликации анализируются следующие события:
  - старт транзакции. Выделяется структура в памяти для хранения точек сохранения;
  - создание точки сохранения. Выделяется структура в памяти для хранения документов;
  - освобождение точки сохранения. Документы сохранённые в этой точки сохранения копируются в точку сохранения более высокого уровня;
  - откат точки сохранения (точка сохранения уничтожается вместе со всеми документами);
  - фиксация транзакции. Документы созданные в событиях INSERT, UPDATE, DELETE и EXECUTE SQL отправляются в обменник RabbitMQ;
  - откат транзакции. Уничтожаются все точки сохранения и сохранённые в них документы;
  - выполнение DDL запроса (EXECUTE SQL). Создаётся новый документ и сохраняется в список документов для точки сохранения. В документе сохраняются текст запроса, тип события и номер транзакции;
  - установка нового значения последовательности (SET SEQUENCE). Создаётся новый документ и немедленно отправляются в обменник RabbitMQ. В документе сохраняются имя последовательности, тип события и новое значение последовательности;
  - события INSERT, UPDATE и DELETE. Проверяется имя таблицы в фильтрах include_tables и exclude_tables, и если имя таблицы удовлетворяет критериям фильтрации, то создаётся новый документ и сохраняется в список документов для точки сохранения. В документе сохраняются имя таблицы, тип события, номер транзакции, старые и

новые значения полей.

- после обработки файла, его номер фиксируется в контрольном файле.

Остальные события репликации игнорируются.

### 17.7.2. Настройка плагина rabbitmq_plugin

В первую очередь необходимо настроить асинхронную репликацию для вашей базы данных, для этого в файле `replication.conf` необходимо добавить следующие строчки:

```
database = d:\fbdata\4.0\horses.fdb
{
    journal_directory = d:\fbdata\4.0\replication\horses\journal
    journal_archive_directory = d:\fbdata\4.0\replication\horses\archive
    journal_archive_command = "copy $(pathname) $(archivepathname) && copy $(pathname)
d:\fbdata\4.0\replication\horses\archive_rabbit
}
```

Обратите внимание: здесь происходит дублирование файлов архивов журналов, чтобы одновременно работала логическая репликация и задача по отправки событий в RabbitMQ. Это необходимо, поскольку файлы с архивами журналов удаляются после обработки и не могут быть использованы другой задаче.

Теперь необходимо включить репликацию в базе данных:

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

Далее настроим конфигурацию `fb_streaming.conf` для того, чтобы `fb_streaming` автоматически отправлял события в обменник RabbitMQ.

Перед настройкой конфигурации откройте в браузере инструмент "RabbitMQ Management". Если вы устанавливали его на тот же компьютер, то по умолчанию необходимо набрать в браузере `http://localhost:15672`, логин "guest" пароль "guest". Этот инструмент позволяет управлять обменниками, очередями и привязками, а также просматривать очереди.

```
task = d:\fbdata\4.0\replication\horses\archive_rabbit
{
    database = inet://localhost:3054/d:\fbdata\4.0\horses.fdb
    username = SYSDBA
    password = masterkey
    deleteProcessedFile = true
    plugin = rabbitmq_plugin
    dumpBlobs = true
    register_ddl_events = true
    register_sequence_events = true
    # include_tables =
```

```
    # exclude_tables =
    rabbit_uri = amqp://guest:guest@localhost
    exchange_name = horses
    exchange_type = direct
    queue_name = horses_events
    routing_key = 123
}
```

В Linux эта конфигурация будет выглядеть так:

```
task = /mnt/d/fbdata/4.0/replication/horses/archive
{
    database = inet://192.168.1.48:3054/horses
    username = SYSDBA
    password = masterkey
    deleteProcessedFile = true
    plugin = rabbitmq_plugin
    dumpBlobs = true
    register_ddl_events = true
    register_sequence_events = true
    # include_tables =
    # exclude_tables =
    rabbit_uri = amqp://test:test@192.168.1.48
    exchange_name = horses
    exchange_type = direct
    queue_name = horses_events
    routing_key = 123
}
```

Для этого плагина появились ряд дополнительных настроек:

- `rabbit_uri` — URI для подключения к серверу RabbitMQ;
- `exchange_name` — имя обменника. Если обменника ещё не существует, то он будет создан;
- `exchange_type` — тип обменника. Допустимые значения: `fanout`, `topic`, `direct` (по умолчанию `fanout`);
- `queue_name` — имя очереди. Если указана, то очередь декларируется и привязывается к обменнику по ключу `routing_key`;
- `routing_key` — ключ маршрутизации;
- `dumpBlobs` — публиковать ли новые значения BLOB полей (по умолчанию `false`);
- `register_ddl_events` — регистрировать ли DDL события (по умолчанию `true`);
- `register_sequence_events` — регистрировать ли события установки значения последовательности (по умолчанию `true`);
- `include_tables` — регулярное выражение, определяющие имена таблиц для которых необходимо отслеживать события;
- `exclude_tables` — регулярное выражение, определяющие имена таблиц для которых не

надо отслеживать события.

Теперь можно установить и запустит службу:

```
c:\streaming>fb_streaming install
Success install service!

c:\streaming>fb_streaming start
Service start pending...
Service started successfully.
```

В Linux:

```
sudo systemctl enable fb_streaming

sudo systemctl start fb_streaming
```

Пока будет работать служба, в очередь `horses_events` будут приходить сообщения.

Содержание сообщений будет примерно таким:

```
{
  "event": "EXECUTE SQL",
  "sql": "CREATE SEQUENCE SEQ1",
  "tnx": 6590
}
{
  "event": "EXECUTE SQL",
  "sql": "CREATE TABLE TABLE1 (\r\n  ID INT NOT NULL,\r\n  S VARCHAR(10),\r\n  PRIMARY
KEY(ID)\r\n)",
  "tnx": 6591
}
{
  "event": "EXECUTE SQL",
  "sql": "ALTER TABLE TABLE1\r\nENABLE PUBLICATION",
  "tnx": 6594
}
{
  "event": "SET SEQUENCE",
  "sequence": "SEQ1",
  "value": 1
}
{
  "event": "INSERT",
  "table": "TABLE1",
  "tnx": 6597,
  "record": {
    "ID": 1,
```

```
    "S": "Hello"
  }
}
{
  "event": "UPDATE",
  "table": "COLOR",
  "tnx": 11771,
  "changedFields": [ "NAME_DE" ],
  "oldRecord": {
     "NAME_EN": "dun",
     "NAME": "мышастая",
     "CODE_COLOR": 14,
     "CODE_SENDER": 1,
     "NAME_DE": "",
     "SHORTNAME_EN": "dun",
     "SHORTNAME": "мыш."
  },
  "record": {
     "NAME_EN": "dun",
     "NAME": "мышастая",
     "CODE_COLOR": 14,
     "CODE_SENDER": 1,
     "NAME_DE": "g",
     "SHORTNAME_EN": "dun",
     "SHORTNAME": "мыш."
  }
}
{
  "event": "INSERT",
  "table": "CLIP",
  "tnx": 11821,
  "record": {
     "AVALUE": 44,
     "CODE_CLIP": 1,
     "CODE_CLIPTYPE": 1,
     "CODE_RECORD": 345,
     "REMARK": null
  }
}
{
  "event": "DELETE",
  "table": "CLIP",
  "tnx": 11849,
  "record": {
     "AVALUE": 44,
     "CODE_CLIP": 1,
     "CODE_CLIPTYPE": 1,
     "CODE_RECORD": 345,
     "REMARK": null
  }
}
```

```
{
  "event": "UPDATE",
  "table": "BREED",
  "tnx": 11891,
  "changedFields": [ "MARK" ],
  "oldRecord": {
     "NAME": "орловская рысистая",
     "CODE_DEPARTURE": 15,
     "CODE_BREED": 55,
     "CODE_SENDER": 1,
     "NAME_EN": "Orlov trotter",
     "SHORTNAME_EN": "orl. trot.",
     "SHORTNAME": "орл.рыс.",
     "MARK": ""
  },
  "record": {
     "NAME": "орловская рысистая",
     "CODE_DEPARTURE": 15,
     "CODE_BREED": 55,
     "CODE_SENDER": 1,
     "NAME_EN": "Orlov trotter",
     "SHORTNAME_EN": "orl. trot.",
     "SHORTNAME": "орл.рыс.",
     "MARK": "5"
  }
}
```


Описание столбцов:

- `event` — тип события;
- `table` — имя таблицы для которой произошло событие;
- `tnx` — номер транзакции в которой произошло событие;
- `record` — новая запись в событиях `INSERT` и `UPDATE`, старая — в событии `DELETE`;
- `oldRecord` — старая запись в событии `UPDATE`;
- `changedFields` — список имён столбцов, которые были изменены в событии `UPDATE`;
- `newBlobs` — новые значения BLOB полей;
- `sql` — текст SQL запроса для DDL операторов;
- `sequence` — наименование последовательности;
- `value` — новое значение последовательности.

## 17.8. Плагин mongodb_events_plugin

Плагин `mongodb_events_plugin` предназначен для регистрации событий в базе данных Firebird и их запись в NoSQL СУБД MongoDB.

Каждое событие представляет собой документ в коллекции `events`.

В событии содержаться следующие сведения:

- тип события (`INSERT`, `UPDATE`, `DELETE`, `SET SEQUENCE`, `EXECUTE SQL`);
- для событий `INSERT`, `UPDATE`, `DELETE`:
  - имя таблицы, для которого произошло это событие;
  - номер транзакции в которой произошло это событие;
  - старые и новые значение полей записи таблицы;
  - массив изменившихся полей для события `UPDATE`;
  - новые значения BLOB полей;
- для события `SET SEQUENCE`:
  - имя последовательности (генератора);
  - значение последовательности;
- для события `EXECUTE SQL`:
  - номер транзакции в которой произошло это событие;
  - текст SQL запроса.

В событии `INSERT` присутствуют только новые значения полей.

В событии `UPDATE` присутствуют как старые, так и новые значения полей.

В событии `DELETE` присутствуют только старые значения полей.

Событие `EXECUTE SQL` возникает только для DDL запросов.

Для BLOB полей всегда сохраняется `BlobId`. Если параметр конфигурации `dumpBlobs`, то в дополнение к этому для событий `INSERT` и `UPDATE` будут публиковаться новые значения BLOB полей. Для старых значений полей это не возможно, поскольку старые значения BLOB полей отсутствуют в файлах журнала репликации.

### 17.8.1. Алгоритм работы плагина mongodb_events_plugin

Каждое событие в базе данных MongoDB сохраняется как документ в коллекцию `events`.

- служба `fb_streaming` анализирует директорию с журналами репликации, установленную для задачи, на наличие необработанных файлов;
- для ещё необработанных файлов репликации анализируются следующие события:
  - старт транзакции. Выделяется структура в памяти для хранения точек сохранения;
  - создание точки сохранения. Выделяется структура в памяти для хранения документов;
  - освобождение точки сохранения. Документы сохранённые в этой точки сохранения копируются в точку сохранения более высокого уровня;

- откат точки сохранения. Точка сохранения уничтожается вместе со всеми документами;
- фиксация транзакции. Документы созданные в событиях `INSERT`, `UPDATE`, `DELETE` и `EXECUTE SQL` записываются в базу данных MongoDB;
- откат транзакции. Уничтожаются все точки сохранения и сохранённые в них документы;
- выполнение DDL запроса (`EXECUTE SQL`). Создаётся новый документ и сохраняется в список документов для точки сохранения. В документе сохраняются текст запроса, тип события и номер транзакции;
- установка нового значения последовательности (`SET SEQUENCE`). Создаётся новый документ и немедленно записываются в базу данных MongoDB. В документе сохраняются имя последовательности, тип события и новое значение последовательности;
- события `INSERT`, `UPDATE` и `DELETE`. Проверяется имя таблицы в фильтрах `include_tables` и `exclude_tables`, и если имя таблицы удовлетворяет критериям фильтрации, то создаётся новый документ и сохраняется в список документов для точки сохранения. В документе сохраняются имя таблицы, тип события, номер транзакции, старые и новые значения полей.

- после обработки файла, его номер фиксируется в контрольном файле.

Остальные события репликации игнорируются.

### 17.8.2. Настройка плагина mongodb_events_plugin

В первую очередь необходимо настроить асинхронную репликацию для вашей базы данных, для этого в файле `replication.conf` необходимо добавить следующие строчки:

```
database = d:\fbdata\4.0\horses.fdb
{
    journal_directory = d:\fbdata\4.0\replication\horses\journal
    journal_archive_directory = d:\fbdata\4.0\replication\horses\archive
    journal_archive_command = "copy $(pathname) $(archivepathname) && copy $(pathname)
d:\fbdata\4.0\replication\horses\archive_mongo
}
```

Обратите внимание: здесь происходит дублирование файлов архивов журналов, чтобы одновременно работала логическая репликация и задача по записи событий в MongoDB. Это необходимо, поскольку файлы с архивами журналов удаляются после обработки и не могут быть использованы другой задаче.

Теперь необходимо включить репликацию в базе данных:

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

Далее настроим конфигурацию `fb_streaming.conf` для того, чтобы `fb_streaming` автоматически записывал события в MongoDB.

```
task = d:\fbdata\4.0\replication\horses\archive_mongo
{
   database = inet://localhost:3054/d:\fbdata\4.0\horses.fdb
   username = SYSDBA
   password = masterkey
   plugin = mongodb_events_plugin
   mongo_uri = mongodb://localhost:27017
   mongo_database = horses
   dumpBlobs = true
   register_ddl_events = true
   register_sequence_events = true
   # include_tables =
   # exclude_tables =
}
```

Для этого плагина появились ряд дополнительных настроек:

- `mongo_uri` — URI для подключения к серверу MongoDB;
- `mongo_database` — имя базы данных в MongoDB в которую сохраняются события;
- `dumpBlobs` — публиковать ли новые значения BLOB полей (по умолчанию `false`);
- `register_ddl_events` — регистрировать ли DDL события (по умолчанию `true`);
- `register_sequence_events` — регистрировать ли события установки значения последовательности (по умолчанию `true`);
- `include_tables` — регулярное выражение, определяющие имена таблиц для которых необходимо отслеживать события;
- `exclude_tables` — регулярное выражение, определяющие имена таблиц для которых не надо отслеживать события.

Если базы данных, указанной в параметре `mongo_database` не существует, то она будет создана при первой записи в неё.

Теперь можно установить и запустит службу:

```
c:\streaming>fb_streaming install
Success install service!

c:\streaming>fb_streaming start
Service start pending...
Service started successfully.
```

В Linux:

```
sudo systemctl enable fb_streaming

sudo systemctl start fb_streaming
```

### 17.8.3. Пример содержимого лога событий в БД MongoDB

Для получения всех событий, набираем в `mongosh` следующие команды

```
use horses;
'switched to db horses'
db.events.find();
```

Здесь первой командой мы переключились на базу данных `horses` в которую велась запись событий.

Вторая команда является запросом на выборку данных из коллекции `events`. Именно в эту коллекцию плагин `mongodb_events_plugin` записывает свои события.

Содержимое коллекции выглядит следующим образом:

```
{ _id: ObjectId("638f37f5022b0000ad005775"),
  event: 'EXECUTE SQL',
  sql: 'CREATE SEQUENCE SEQ1',
  tnx: 6590 }
{ _id: ObjectId("638f37f9022b0000ad005776"),
  event: 'EXECUTE SQL',
  sql: 'CREATE TABLE TABLE1 (\r\n  ID INT NOT NULL,\r\n  S VARCHAR(10),\r\n  PRIMARY
KEY(ID)\r\n)',
  tnx: 6591 }
{ _id: ObjectId("638f37f9022b0000ad005777"),
  event: 'EXECUTE SQL',
  sql: 'ALTER TABLE TABLE1\r\nENABLE PUBLICATION',
  tnx: 6594 }
{ _id: ObjectId("638f37f9022b0000ad005778"),
  event: 'SET SEQUENCE',
  sequence: 'SEQ1',
  value: 1 }
{ _id: ObjectId("638f37f9022b0000ad005779"),
  event: 'INSERT',
  table: 'TABLE1',
  tnx: 6597,
  record: { ID: 1, S: 'Hello' } }
{ _id: ObjectId("638f3823022b0000ad00577b"),
  event: 'EXECUTE SQL',
  sql: 'DROP TABLE TABLE1',
  tnx: 6608 }
{ _id: ObjectId("638f3827022b0000ad00577c"),
  event: 'EXECUTE SQL',
```

```
  sql: 'DROP SEQUENCE SEQ1',
  tnx: 6609 }
{ _id: ObjectId("633d8c86873d0000d8004172"),
  event: 'UPDATE',
  table: 'COLOR',
  tnx: 11771,
  changedFields: [ 'NAME_DE' ],
  oldRecord:
   { NAME_EN: 'dun',
     NAME: 'мышастая',
     CODE_COLOR: 14,
     CODE_SENDER: 1,
     NAME_DE: '',
     SHORTNAME_EN: 'dun',
     SHORTNAME: 'мыш.' },
  record:
   { NAME_EN: 'dun',
     NAME: 'мышастая',
     CODE_COLOR: 14,
     CODE_SENDER: 1,
     NAME_DE: 'g',
     SHORTNAME_EN: 'dun',
     SHORTNAME: 'мыш.' } }
{ _id: ObjectId("633d8c8a873d0000d8004173"),
  event: 'UPDATE',
  table: 'COLOR',
  tnx: 11790,
  changedFields: [ 'NAME_DE' ],
  oldRecord:
   { NAME_EN: 'dun',
     NAME: 'мышастая',
     CODE_COLOR: 14,
     CODE_SENDER: 1,
     NAME_DE: 'g',
     SHORTNAME_EN: 'dun',
     SHORTNAME: 'мыш.' },
  record:
   { NAME_EN: 'dun',
     NAME: 'мышастая',
     CODE_COLOR: 14,
     CODE_SENDER: 1,
     NAME_DE: '',
     SHORTNAME_EN: 'dun',
     SHORTNAME: 'мыш.' } }
{ _id: ObjectId("633d8c8a873d0000d8004174"),
  event: 'INSERT',
  table: 'CLIP',
  tnx: 11821,
  record:
   { AVALUE: 44,
     CODE_CLIP: 1,
```

```
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 345,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d8004175"),
  event: 'DELETE',
  table: 'CLIP',
  tnx: 11849,
  record:
   { AVALUE: 44,
     CODE_CLIP: 1,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 345,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d8004176"),
  event: 'UPDATE',
  table: 'BREED',
  tnx: 11891,
  changedFields: [ 'MARK' ],
  oldRecord:
   { NAME: 'орловская рысистая',
     CODE_DEPARTURE: 15,
     CODE_BREED: 55,
     CODE_SENDER: 1,
     NAME_EN: 'Orlov trotter',
     SHORTNAME_EN: 'orl. trot.',
     SHORTNAME: 'орл.рыс.',
     MARK: '' },
  record:
   { NAME: 'орловская рысистая',
     CODE_DEPARTURE: 15,
     CODE_BREED: 55,
     CODE_SENDER: 1,
     NAME_EN: 'Orlov trotter',
     SHORTNAME_EN: 'orl. trot.',
     SHORTNAME: 'орл.рыс.',
     MARK: '5' } }
{ _id: ObjectId("633d8c8a873d0000d8004177"),
  event: 'INSERT',
  table: 'CLIP',
  tnx: 11913,
  record:
   { AVALUE: 1,
     CODE_CLIP: 2,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 1,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d8004178"),
  event: 'DELETE',
  table: 'CLIP',
  tnx: 11942,
  record:
```

```
   { AVALUE: 1,
     CODE_CLIP: 2,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 1,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d8004179"),
  event: 'INSERT',
  table: 'CLIP',
  tnx: 12001,
  record:
   { AVALUE: 3,
     CODE_CLIP: 5,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 1,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d800417a"),
  event: 'DELETE',
  table: 'CLIP',
  tnx: 12039,
  record:
   { AVALUE: 3,
     CODE_CLIP: 5,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 1,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d800417b"),
  event: 'UPDATE',
  table: 'COLOR',
  tnx: 11799,
  changedFields: [ 'NAME_DE' ],
  oldRecord:
   { NAME_EN: 'clay with mixed hairs',
     NAME: 'бурая в седине',
     CODE_COLOR: 118,
     CODE_SENDER: 1,
     NAME_DE: '',
     SHORTNAME_EN: 'c.m.h.',
     SHORTNAME: 'бур. в сед.' },
  record:
   { NAME_EN: 'clay with mixed hairs',
     NAME: 'бурая в седине',
     CODE_COLOR: 118,
     CODE_SENDER: 1,
     NAME_DE: '3',
     SHORTNAME_EN: 'c.m.h.',
     SHORTNAME: 'бур. в сед.' } }
{ _id: ObjectId("633d8c8a873d0000d800417c"),
  event: 'INSERT',
  table: 'CLIP',
  tnx: 12087,
  record:
```

```
   { AVALUE: 1,
     CODE_CLIP: 6,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 2,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d800417d"),
  event: 'INSERT',
  table: 'CLIP',
  tnx: 12087,
  record:
   { AVALUE: 3,
     CODE_CLIP: 7,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 3,
     REMARK: 'Странный' } }
{ _id: ObjectId("633d8c8a873d0000d800417e"),
  event: 'UPDATE',
  table: 'BREED',
  tnx: 12197,
  changedFields: [ 'MARK' ],
  oldRecord:
   { NAME: 'русская рысистая',
     CODE_DEPARTURE: 17,
     CODE_BREED: 58,
     CODE_SENDER: 1,
     NAME_EN: 'Trotter',
     SHORTNAME_EN: 'rus.rys.',
     SHORTNAME: 'рус.рыс.',
     MARK: '' },
  record:
   { NAME: 'русская рысистая',
     CODE_DEPARTURE: 17,
     CODE_BREED: 58,
     CODE_SENDER: 1,
     NAME_EN: 'Trotter',
     SHORTNAME_EN: 'rus.rys.',
     SHORTNAME: 'рус.рыс.',
     MARK: '3' } }
{ _id: ObjectId("633d8c8a873d0000d800417f"),
  event: 'UPDATE',
  table: 'COLOR',
  tnx: 12218,
  changedFields: [ 'NAME_DE', 'SHORTNAME_EN' ],
  oldRecord:
   { NAME_EN: 'red grey',
     NAME: 'красно-серая',
     CODE_COLOR: 3,
     CODE_SENDER: 1,
     NAME_DE: '',
     SHORTNAME_EN: '2',
     SHORTNAME: 'кр.-сер.' },
```

```
  record:
   { NAME_EN: 'red grey',
     NAME: 'красно-серая',
     CODE_COLOR: 3,
     CODE_SENDER: 1,
     NAME_DE: '5',
     SHORTNAME_EN: '',
     SHORTNAME: 'кр.-сер.' } }
{ _id: ObjectId("633d8c8a873d0000d8004180"),
  event: 'INSERT',
  table: 'CLIP',
  tnx: 12287,
  record:
   { AVALUE: 0,
     CODE_CLIP: 8,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 5,
     REMARK: null } }
{ _id: ObjectId("633d8c8a873d0000d8004181"),
  event: 'UPDATE',
  table: 'CLIP',
  tnx: 12287,
  changedFields: [ 'REMARK' ],
  oldRecord:
   { AVALUE: 3,
     CODE_CLIP: 7,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 3,
     REMARK: 'Странный' },
  record:
   { AVALUE: 3,
     CODE_CLIP: 7,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 3,
     REMARK: 'Странный 2' } }
{ _id: ObjectId("633d8c8a873d0000d8004182"),
  event: 'DELETE',
  table: 'CLIP',
  tnx: 12287,
  record:
   { AVALUE: 1,
     CODE_CLIP: 6,
     CODE_CLIPTYPE: 1,
     CODE_RECORD: 2,
     REMARK: null } }
```

Описание полей:

- `_id` — внутренний первичный ключ для коллекции MongoDB;
- `event` — тип события;

- `table` — имя таблицы для которой произошло событие;
- `tnx` — номер транзакции в которой произошло событие;
- `record` — новая запись в событиях `INSERT` и `UPDATE`, старая — в событии `DELETE`;
- `oldRecord` — старая запись в событии `UPDATE`;
- `changedFields` — список имён столбцов, которые были изменены в событии `UPDATE`;
- `newBlobs` — новые значения BLOB полей;
- `sql` — текст SQL запроса для DDL операторов;
- `sequence` — наименование последовательности;
- `value` — новое значение последовательности.

## 17.9. Плагин fts_lucene_plugin

Плагин `fts_lucene_plugin` предназначен для поддержания полнотекстовых индексов, созданных с помощью библиотеки IBSurgeon Full Text Search UDR (см. https://github.com/IBSurgeon/lucene_udr), в актуальном состоянии.

Плагин `fts_lucene_plugin` не может поддерживать индексы в качестве ключа которого выбран псевдо-столбец `RDB$DB_KEY`, поскольку значения этого псевдо-столбца не попадают в лог репликации. Кроме того, не будут автоматически обновляться полнотекстовые индексы построенные для представлений.

### 17.9.1. Алгоритм работы плагина fts_lucene_plugin

- служба `fb_streaming` анализирует директорию с журналами репликации, установленную для задачи, на наличие необработанных файлов;
- на каждой итерации перечитываются метаданные полнотекстовых индексов;
  - выбираются только полнотекстовые индексы в активном состоянии;
  - создаётся список таблиц, для которых созданы эти индексы, в дальнейшем этот список используется для фильтрации событий (`INSERT`, `UPDATE`, `DELETE`);
- для ещё необработанных файлов репликации анализируются следующие события:
  - старт транзакции (выделяется структура в памяти для хранения точек сохранения);
  - создание точки сохранения (выделяется структура в памяти для хранения документов полнотекстовых индексов);
  - освобождение точки сохранения (документы сохранённые в этой точки сохранения копируются в точку сохранения более высокого уровня);
  - откат точки сохранения (точка сохранения уничтожается вместе со всеми документами);
  - фиксация транзакции (применяются изменения для всех затронутых полнотекстовых индексов: новые документы добавляются в соответствующий индекс, изменившиеся обновляются, а удалённые удаляются);

- откат транзакции (структура в памяти для транзакции уничтожается);
- вставка новой записи (только для таблиц которые участвуют в полнотекстовых индексах). Для каждого индекса создаётся новый документ и сохраняется в список добавленных документов для точки сохранения.
- обновление записи (только для таблиц которые участвуют в полнотекстовых индексах). Для каждого индекса создаётся новый документ. Новый документ и ключ для поиска сохраняются в список документов для обновления для точки сохранения.
- удаление записи (только для таблиц которые участвуют в полнотекстовых индексах). В текущую точку сохранение сохраняется значение ключа для поиска и удаления соответствующего документа.

Остальные события репликации игнорируются.

Особенности работы:

- если в созданном документе все индексируемые поля пусты, документ не будет добавлен в индекс;
- если не изменилось ни одного индексируемого поля в документе, то такой документ не будет обновлён в индексе;
- если в обновлённом документе все индексируемые поля пусты, то такой документ будет удалён.

### 17.9.2. Настройка плагина fts_lucene_plugin

Для примера будем использовать базу данных fts_demo_4.0

Прежде всего, необходимо установить для этой базы данных библиотеку IBSurgeon Full Text Search UDR и создать нужные нам полнотекстовые индексы. Этот процесс описан в https://github.com/IBSurgeon/lucene_udr

Теперь настроим асинхронную репликацию для этой базы данных, для этого в файле `replication.conf` необходимо добавить следующие строчки:

```
database = d:\fbdata\4.0\fts_demo.fdb
{
    journal_directory = d:\fbdata\4.0\replication\fts_demo\journal
    journal_archive_directory = d:\fbdata\4.0\replication\fts_demo\archive
    journal_archive_command = "copy $(pathname) $(archivepathname) && copy $(pathname)
d:\fbdata\4.0\replication\fts_demo\archive_fts
}
```

Обратите внимание: здесь происходит дублирование файлов архивов журналов, чтобы одновременно работала логическая репликация и задача по обновлению полнотекстовых индексов. Это необходимо, поскольку файлы с архивами журналов удаляются после обработки и не могут быть использованы другой задаче.

Теперь необходимо включить репликацию в базе данных:

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

Далее настроим конфигурацию `fb_streaming.conf` для того, чтобы `fb_streaming` автоматически обновлял полнотекстовые индексы.

```
task = d:\fbdata\4.0\replication\fts_demo\archive_fts
{
   database = inet://localhost:3054/d:\fbdata\4.0\fts_demo.fdb
   username = SYSDBA
   password = masterkey
   plugin = fts_lucene_plugin
}
```

Теперь можно установить и запустит службу:

```
c:\streaming>fb_streaming install
Success install service!

c:\streaming>fb_streaming start
Service start pending...
Service started successfully.
```

В Linux:

```
sudo systemctl enable fb_streaming

sudo systemctl start fb_streaming
```

Пока служба запущена все активные полнотекстовые индексы, за исключением индексов созданных на представлениях и индексов, где в качестве ключа выбран псевдо-столбец `RDB$DB_KEY`, будут автоматически обновляться при изменении данных в таблицах базы данных `fts_demo.fdb`.

## 17.10. Плагин simple_json_plugin

Плагин `simple_json_plugin` предназначен для автоматической трансляции бинарных файлов журнала репликации в эквивалентный JSON формат и сохранять журналы в заданную директорию с тем же именем, но с расширением `.json`.

Каждый json файл содержит в себе корневой объект, состоящий из двух полей: `header` и `events`.

Поле `header` представляет собой объект описывающий заголовок сегмента репликации. Он содержит следующие поля:

- `guid` — GUID базы данных;
- `sequence` — номер сегмента репликации;
- `state` — состояние в котором находится сегмент. Возможные значения: `free`, `used`, `full`, `archive`;
- `version` — версия протокола журнала репликации.

Поле `events` представляет собой массив объектов, каждый из которых представляет одно из событий репликации.

Объект события может содержать следующие поля:

- `event` — тип события. Доступны следующие варианты:
  - `SET SEQUENCE` — установка значения последовательности (генератора);
  - `START TRANSACTION` — старт транзакции;
  - `PREPARE TRANSACTION` — выполнение первой фазы подтверждения двухфазной транзакции;
  - `SAVEPOINT` — установка точки сохранения;
  - `RELEASE SAVEPOINT` — освобождение точки сохранения;
  - `ROLLBACK SAVEPOINT` — откат точки сохранения;
  - `COMMIT` — подтверждение транзакции;
  - `ROLLBACK` — откат транзакции;
  - `INSERT` — вставка новой записи в таблицу;
  - `UPDATE` — обновление записи в таблице;
  - `DELETE` — удаление записи из таблицы;
  - `EXECUTE SQL` — выполнение SQL оператора. Такие события происходят только для DDL операторов;
  - `STORE BLOB` — сохранение BLOB.
- `tnx` — номер транзакции. Доступно для всех событий кроме `SET SEQUENCE`, поскольку генераторы работают вне контекста транзакций;
- `sequence` — имя последовательности. Доступно только в событии `SET SEQUENCE`;
- `value` — значение последовательности. Доступно только в событии `SET SEQUENCE`;
- `sql` — текст SQL запроса. Доступно только в событии `EXECUTE SQL`;
- `blobId` — идентификатор BLOB. Доступно только в событии `STORE BLOB`;
- `data` — данные сегмента BLOB в шестнадцатеричном представлении. Доступно только в событии `STORE BLOB`;
- `table` — имя таблицы. Доступно в событиях `INSERT`, `UPDATE` и `DELETE`;
- `record` — значения полей записи. Для событий `INSERT` и `UPDATE` это новая запись, а для `DELETE` - старая;
- `oldRecord` — значения полей старой записи. Доступно только в событии `UPDATE`;

- `changedFields` — массив имён изменённых полей. Доступно только в событии `UPDATE`.

Обратите внимание, что для BLOB полей в качестве значения указан идентификатор BLOB.

Пример содержимого файла `.json`:

```
{
    "events": [
        {
            "event": "START TRANSACTION",
            "tnx": 6259
        },
        {
            "event": "SAVEPOINT",
            "tnx": 6259
        },
        {
            "event": "SAVEPOINT",
            "tnx": 6259
        },
        {
            "changedFields": [
                "SHORTNAME_EN"
            ],
            "event": "UPDATE",
            "oldRecord": {
                "CODE_COLOR": 3,
                "CODE_SENDER": 1,
                "NAME": "красно-серая",
                "NAME_DE": "",
                "NAME_EN": "red grey",
                "SHORTNAME": "кр.-сер.",
                "SHORTNAME_EN": ""
            },
            "record": {
                "CODE_COLOR": 3,
                "CODE_SENDER": 1,
                "NAME": "красно-серая",
                "NAME_DE": "",
                "NAME_EN": "red grey",
                "SHORTNAME": "кр.-сер.",
                "SHORTNAME_EN": "fff"
            },
            "table": "COLOR",
            "tnx": 6259
        },
        {
            "event": "RELEASE SAVEPOINT",
            "tnx": 6259
        },
        {
```

```
            "event": "RELEASE SAVEPOINT",
            "tnx": 6259
        },
        {
            "event": "COMMIT",
            "tnx": 6259
        }
    ],
    "header": {
        "guid": "{AA08CB53-C875-4CA3-B513-877D0668885D}",
        "sequence": 3,
        "state": "archive",
        "version": 1
    }
}
```

### 17.10.1. Настройка плагина

Пример настройки плагина:

```
task = d:\fbdata\4.0\replication\testdb\archive
{
    deleteProcessedFile = true
    database = inet://localhost:3054/test
    username = SYSDBA
    password = masterkey
    plugin = simple_json_plugin
    dumpBlobs = true
    register_ddl_events = true
    register_sequence_events = true
    outputDir = d:\fbdata\4.0\replication\testdb\json_archive
    # include_tables =
    # exclude_tables =
}
```

Описание параметров:

- `controlFileDir` — директория в которой будет создан контрольный файл (по умолчанию та же директория, что и `sourceDir`);
- `database` — строка подключения к базе данных (обязательный);
- `username` — имя пользователя для подключения к базе данных;
- `password` — пароль для подключения к базе данных;
- `plugin` — плагин, который обрабатывает события, возникающие в процессе анализа журнала репликации (обязательный);
- `deleteProcessedFile` — удалять ли файл журнала после обработки (по умолчанию `true`);
- `outputDir` — директория в которой будут находится готовые JSON файлы (обязательный);

- `dumpBlobs` — публиковать ли данные BLOB полей (по умолчанию `false`);
- `register_ddl_events` — регистрировать ли DDL события (по умолчанию `true`);
- `register_sequence_events` — регистрировать ли события установки значения последовательности (по умолчанию `true`);
- `include_tables` — регулярное выражение, определяющие имена таблиц для которых необходимо отслеживать события;
- `exclude_tables` — регулярное выражение, определяющие имена таблиц для которых не надо отслеживать события.

# Приложение A: Выражения CRON

Все задания в FBDataGuard имеют настройки времени в формате CRON. CRON — очень простой и мощный формат для планирования времени выполнения.

## Формат CRON

Выражение CRON представляет собой строку, состоящую из 6 или 7 полей, разделенных пробелами. Поля могут содержать любые разрешенные значения, а также различные комбинации разрешенных специальных символов для этого поля. Поля следующие:

| Имя поля | Обязательное | Допустимые значения | Допустимые специальные символы |
|---|---|---|---|
| Секунды | Да | 0-59 | , - * / |
| Минуты | Да | 0-59 | , - * / |
| Часы | Да | 0-23 | , - * / |
| День месяца | Да | 1-31 | , - * / L W |
| Месяц | Да | 1-12 или JAN-DEC | , - * / |
| День недели | Да | 1-7 или SUN-SAT | , - * / L # |
| Год | Нет | empty, 1970-2099 | , - * / |

Таким образом, выражения cron могут быть такими простыми: `\* * * * ? *` или более сложными, например: `0 0/5 14,18,3-39,52 ? JAN,MAR,SEP MON-FRI 2002-2010`

## Специальные символы

- * (“все значения”) — используется для выбора всех значений в поле. Например, “*” в поле минуты обозначает “каждую минуту”.
- ? (“нет конкретного значения”) — полезно, когда вам нужно указать что-то в одном из двух полей, в котором разрешен символ, но не в другом. Например, если я хочу, чтобы мой триггер сработал в определенный день месяца (скажем, 10-го числа), но меня не волнует, какой это будет день недели. Тогда необходимо поместить “10” в поле день месяца и “?” в поле день недели. См. пояснительные примеры ниже.
- - — используется для указания диапазонов. Например, “10-12” в поле часы обозначает “10, 11 и 12 час”.
- , — используется для указания дополнительных значений. Например, `MON,WED,FRI` в поле день недели обозначает “Понедельник, Среда и Пятница”.
- / — используется для указания приращений. Например, “0/15” в поле секунды обозначает “0, 15, 30, и 45 секунду”. А “5/15” в поле секунды обозначает “5, 20, 35, и 50 секунду”. Вы также можете указать “/” после символа “*” — в этом случае “*” эквивалентна наличию “0” до “/”. “1/3” в поле день месяца обозначает “срабатывать

каждые 3 дня начиная с первого дня месяца”.

- `L` (“last”) — имеет разное значение в каждом из двух полей, в которых оно разрешено. Например, значение “L” в поле день месяца обозначает “последний день месяца” — 31 января, 28 день Февраля в невисокосные годы. Если оно используется в поле день недели, то это просто обозначает “7” или “SAT”. Но если оно используется в поле дня недели после другого значения, то оно означает “последний xxx день месяца” — например “6L” обозначает “последняя Пятница месяца”. При использовании опции “L” важно не указывать списки или диапазоны значений, так как вы получите запутанные результаты.
- `W` (“weekday”) — используется для указания дня недели (Понедельник-Пятница) ближайшего к данному дню. (Дня рабочей недели). Например, если вы указали “15W” в качестве значения в поле день месяца, то это обозначает: “ближайший день недели к 15 числу месяца”. Таким образом, если 15-е число — суббота, то триггер сработает в пятницу 14-го. Если 15-е число выпадает на воскресенье, триггер сработает в понедельник 16-го числа. Если 15-е число — вторник, то сработает во вторник 15-го. Однако если вы укажете “1W” в качестве значения для дня месяца, а 1-е число — суббота, то триггер сработает в понедельник, 3-го числа, так как он не будет “перепрыгивать” через границу дня месяца. Символ “W” можно указать только в том случае, если день месяца представляет собой один день, а не диапазон или список.

Символы “L” и “W” также можно объединить в поле день месяца, чтобы получить “LW”, что переводится как “последний день недели месяца”.

- `#` — используется для указания “n-ого” XXX дня месяца. Например, значение “6#3” в поле дня недели означает “третью пятницу месяца” (день 6 = Пятница и “#3” = третья пятница месяца). Другие примеры: “2#1” = первый понедельник месяца, “4#5” = пятая среда месяца. Обратите внимание, что если вы укажете “#5” и в месяце нет пяти заданных дней недели, то в этом месяце триггер не сработает.

Специальные символы и названия дней и месяцев не чувствительны к регистру символов. “MON” — то же самое, что и “mon”.

## Примеры CRON

Вот несколько полных примеров:

| Выражение | Значение |
|---|---|
| `0 0 12 * * ?` | Запуск в 12:00 (полдень) каждый день. |
| `0 15 10 ? * *` | Запуск в 10:15 каждый день. |
| `0 15 10 * * ?` | Запуск в 10:15 каждый день. |
| `0 15 10 * * ? *` | Запуск в 10:15 каждый день. |
| `0 15 10 * * ? 2005` | Запуск в 10:15 каждый день в течении 2005 года. |

| Выражение | Значение |
|---|---|
| `0 * 14 * * ?` | Запуск каждую минуту, начиная с 14:00 и заканчивая 14:59, каждый день. |
| `0 0/5 14 * * ?` | Запуск каждые 5 минут, начиная с 14:00 и заканчивая 14:55, каждый день. |
| `0 0/5 14,18 * * ?` | Запуск каждые 5 минут, начиная с 14:00 и заканчивая 14:55, и запуск каждые 5 минут, начиная с 18:00 и заканчивая 18:55, каждый день. |
| `0 0-5 14 * * ?` | Запуск каждую минуту, начиная с 14:00 и заканчивая 14:05, каждый день. |
| `0 10,44 14 ? 3 WED` | Запуск в 14:10 и в 14:44 каждую среду в марте. |
| `0 15 10 ? * MON-FRI` | Запуск в 10:15 каждый понедельник, вторник, среду, четверг и пятницу. |
| `0 15 10 15 * ?` | Запуск в 10:15 15-го числа каждого месяца. |
| `0 15 10 L * ?` | Запуск в 10:15 в последний день каждого месяца. |
| `0 15 10 ? * 6L` | Запуск в 10:15 в последнюю пятницу каждого месяца. |
| `0 15 10 ? * 6L 2002-2005` | Запуск в 10:15 каждую последнюю пятницу каждого месяца в 2002, 2003, 2004 и 2005 годах. |
| `0 15 10 ? * 6#3` | Запуск в 10:15 в третью пятницу каждого месяца. |
| `0 0 12 1/5 * ?` | Запуск в 12:00 (полдень) каждые 5 дней каждого месяца, начиная с первого числа месяца. |
| `0 11 11 11 11 ?` | Запуск каждое 11 ноября в 11:11. |

Обратите внимание на эффекты '?' и '*' в полях дня недели и дня месяца!

## Замечания

Поддержка указания значения дня недели и дня месяца не является полной (в настоящее время необходимо использовать символ '?' в одном из этих полей).

Будьте осторожны, устанавливая время пожара между полуночью и 1:00 ночи — “переход на летнее время” может привести к пропуску или повторению в зависимости от того, перемещается ли время назад или вперед.

Больше информации читайте http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html

# Приложение B: HQbird web API

HQbird FBDataGuard позволяет автоматизировать множество задач администрирования, запускать их по расписанию или по требованию из Web консоли DataGuard.

В некоторых случаях приложениям требуется запускать административные задачи, которые уже реализованы в HQbird FBDataGuard. Для разработчиков таких приложений HQbird FBDataGuard предоставляет Web API для запуска задач, настроенных в FBDataGuard.

Запускаемые задачи могут быть уровня агента, сервера или базы данных. Для успешного запуска задачи она должна быть "разрешена" (enabled в конфигурации). Для задач уровня базы данных, необходимо знать регистрационный идентификатор базы.

Все задачи запускаются через один и тот же обработчик, который представляет собой Web API по HTTP протоколу методом GET. URL этого обработчика имеет следующий вид:

```
<schema>://<host>:<port>/agent/runnow?<parameters>

<parameters> ::= <parameter>[&<parameter> [&<parameter> ...]]

<parameter> ::= <param_name>=<param_value>
```

Здесь

- `<schema>` - схема (в нашем случае протокол http или https);
- `<host>` - имя хоста или IP адрес на котором запущен HQbird FBDataGuard;
- `<port>` - номер порта на котором работает служба HQbird FBDataGuard;
- `<param_name>` - имя параметра;
- `<param_value>` - значение параметра, которое должно быть закодировано с помощью URL кодирования описанного в RFC 3986.

Для конфигурации по умолчанию этот URL принимает следующий вид:

```
http://127.0.0.1:8082/agent/runnow?<parameters>
```

Для запуска задач требуются следующий параметры:

- `src` - источник (уровень задачи);
- `job` - имя задачи;
- дополнительные параметры, зависящие от задачи, например `lvl=1`;
- опциональный параметр, которым выступает некоторое, уникальное, целое число (метка времени) - которая предотвращает кэширования результата браузерами.

Источник `src` должен иметь полную форму вида пути к задаче, примеры:

```
/agent/jobs/check-agent-space
```

или

```
/agent/servers/hqbirdsrv/jobs/check-server-log
```

или

```
/agent/servers/hqbirdsrv/databases/a8005009-c850-42ec-8def-133bd8405253/jobs/lockprint
```

В зависимости от содержимого источника (параметр `src`) определяет уровень задачи:

- если источник имеет вид `/agent/servers/hqbirdsrv/databases/<database_guid>/jobs/⋯`, где `<database_guid>` - это идентификатор базы данных, то это задача уровня базы данных. В примере выше база данных имеет идентификатор `a8005009-c850-42ec-8def-133bd8405253`. Идентификатор базы данных можно узнать по соответствующему имени каталога в файловом дереве конфигурации DataGuard;
- если источник имеет вид `/agent/servers/hqbirdsrv/jobs/⋯`, то это задача уровня уровня сервера;
- если источник имеет вид `/agent/jobs/⋯`, то это задача уровня уровня агента.

Приведём несколько примеров запуска задач по требованию. В этих примерах предполагается, что FBDataGuard работает на узле `127.0.0.1:8082`.

*Пример 7. Вызов задачи проверки лога сервера*

URL задачи:

```
http://127.0.0.1:8082/agent/runnow?r=1727043834949&src=%2Fagent%2Fservers%2Fhqbird
srv%2Fjobs%2Fcheck-server-log&job=check-server-log
```

Здесь

- `src=/agent/servers/hqbirdsrv/jobs/check-server-log`
- `job=check-server-log`
- `r=1727043834949` - параметр для предотвращения кэширования браузером повторных запросов.

Сервер в ответ вернёт строку в виде json с указанием времени, тэга, пути к файлу конфигурации задачи и возможным сообщением об ошибке.

Пример ответа:

```
{
```

```
    "Queried": "1727044250754",
    "Tag": "check-server-log@[ hqbirdsrv / * ]",
    "JobConfigFile": "E:\\HQBirdData\\config\\agent\\servers\\hqbirdsrv\\jobs\\check-server-log\\job.properties",
    "ErrorMessage": ""
}
```


Пример ответа с сообщением об ошибке на попытку запуска отключённой задачи:


```
{
    "Queried":"1727044440296",
    "Tag":"check-replica-log@[ hqbirdsrv / * ]",
    "JobConfigFile":"E:\\HQBirdData\\config\\agent\\servers\\hqbirdsrv\\jobs\\check-replica-log\\job.properties",
    "ErrorMessage":"Disabled job cannot be started! Enable job first! \nJob id: \"check-replica-log@[ hqbirdsrv /
* ]\""
}
```


Далее будут приведены примеры запуска задачи для базы данных, зарегистрированной в Dataguard под идентификатором Dataguard ID: 2409230136_EMPLOYEE3_FDB.

*Пример 8. Вызов sweep для базы данных*

URL задачи:

```
http://127.0.0.1:8082/agent/runnow?r=1727044656798&src=%2Fagent%2Fservers%2Fhqbird
srv%2Fdatabases%2F2409230136_EMPLOYEE3_FDB%2Fjobs%2Fcronsweep&job=cronsweep
```

Здесь

- `src=/agent/servers/hqbirdsrv/databases/2409230136_EMPLOYEE3_FDB/jobs/cronsweep`
- `job=cronsweep`

*Пример 9. Запуск задачи nbackup (версии simple) уровня 2*

URL задачи:

```
http://127.0.0.1:8082/agent/runnow?r=1727045317158&src=%2Fagent%2Fservers%2Fhqbird
srv%2Fdatabases%2F2409230136_EMPLOYEE3_FDB%2Fjobs%2Fnbackup&job=nbackup&lvl=2
```

Здесь

- `src=/agent/servers/hqbirdsrv/databases/2409230136_EMPLOYEE3_FDB/jobs/nbackup`
- `job=nbackup`
- `lvl=2` - задаёт уровень бакапа (2).

*Пример 10. Запуск задачи nbackup (версии advanced) уровня 3*

URL задачи:

```
http://127.0.0.1:8082/agent/runnow?r=1727045751740&src=%2Fagent%2Fservers%2Fhqbird
srv%2Fdatabases%2F2409230136_EMPLOYEE3_FDB%2Fjobs%2Fnbackup3&job=nbackup3&lvl=3
```

Здесь

- `src=/agent/servers/hqbirdsrv/databases/2409230136_EMPLOYEE3_FDB/jobs/nbackup3`
- `job=nbackup3`
- `lvl=3` - задаёт уровень резервной копии (3). Является некоторым излишеством, передаётся браузером/вебконсолью, но не используется сервером и может быть пропущен, т.к. уровни advanced nbackup реализованы отдельными потоками с уникальными именами, то есть задачи nbackup0..nbackup4 - и для них уровни в lvl должны соответствовать уровню в пути/имени.

Запуски задач по требованию протоколируются в журнал FBDataGuard.

Полный список путей/имён задач можно реконструировать из каталога действующей в конкретной версии в конфигурации DataGuard — в разных версиях DataGuard они могут отличаться так как задачи менялись, удалялись или добавлялись новые.

## Отладка

Следующие URL помогут при отладке команд:

- Запуск команд

  ```
  http://localhost:8082/static/config.html#hqbirdsrv
  ```

- Просмотр API в необработанном виде

  ```
  http://localhost:8082/agent/
  ```

Сервисную страницу можно открыть (откроется в новой вкладке) непосредственно из веб-окна консоли FbDataGuard, если кликнуть левой кнопкой мыши с зажатой клавишей `Ctrl` по тексту версии DataGuard в левом нижнем углу страницы.

# Приложение C: Контакты поддержки

Мы ответим на все ваши вопросы, касающиеся HQbird FBDataGuard. Пожалуйста, направляйте все ваши вопросы на support@ib-aid.com

Обратите внимание, что клиенты с активной поддержкой Firebird имеют приоритет в технической поддержке https://ib-aid.com/en/firebird-support-service/