MWA Software

IBX for Lazarus User Guide

Issue 1.1,
24 February 2017
McCallum Whyman Associates Ltd
EMail: info@mccallumwhyman.com, http://www.mccallumwhyman.com

Registered in England Registration No. 2624328

От переводчика

Во-первых, должен предупредить, что это мой первый опыт перевода такой, сравнительно большой документации. Во-вторых я никоим образом не являюсь профессиональным переводчиком, зато довольно давно пользуюсь компонентами IBX, ни разу не встретив подобного руководства. Поэтому решил помочь тем русским программистам для которых читать оригинал затруднительно.

Замечания по поводу перевода терминов.

- Набор данных = таблица
- Запись = строка
- Поле = столбец
- Английские update и refresh по русски близки к одному слову — обновить, но смысл их в применении к базе данных, сильно различается. Update – это изменить текущую запись и я стараюсь переводить его словом «редактировать/изменять», а Refresh – это привести данные в соответствии с текущим состоянием в базе (с учетом изменений, сделанных другими) и я стараюсь переводить его словом «актуализировать»
- Похожая проблема с post/commit . Post относится к набору данных и текущей записи и я его стараюсь переводить как «зафиксировать», а commit – к транзакции и я его перевожу «подтвердить»
- Нормального русского перевода Data aware controls я не нашел, хотя обращался даже на форум freepascal.ru . Мой вариант — виджеты управления данными (или просто виджеты данных). Кому не нравится — извиняйте.

Разумеется не все темы мне хорошо знакомы, так что в случае сомнений — читайте оригинал. Особенно это относится к работе со встроенным сервером (раздел 11) и вопросам изменения схемы базы данных (ну не пользовался никогда).

В работе активно пользовался переводчиком Yandex

Работа по переводу посвящается Глаше Кулик.

DedFriend@Mail.ru
Форум на FreePascal.ru

Оглавление

1 Введение

Документ *IBX for Lazarus* Guide является руководством по использованию клона IBX для Lazarus разработанного кампанией MWA Software. *IBX для Lazarus* является производным от издания IBX с открытым исходным кодом, опубликованного Borland/Inprise в 2000 году под Общественной лицензией InterBase

В 2011, версия IBX с исходным кодом была обновлена компанией MWA Software (http://www.mwasoftware.co.uk) , после чего компания сосредоточилась на поддержке Firebird API для баз данных на платформах Linux и Windows (32 и 64-bit), чем продолжает заниматься до сих пор. На этот продукт распространяется действие лицензии InterBase на открытый код и совместимой с ней государственной лицензии на новое программное обеспечение. Систему Управления Реляционными Базами Данных (СУБД) Firebird можно скачать с http://www.firebirdsql.org.

Несмотря на то, что ядром системы остается оригинальный код IBX software, эта версия включает ряд совершенно новых редакторов свойств, поддерживающих генерацию SQL и их тестирование с использованием Firebird непосредственно из IDE. Значительно улучшены редакторы свойств Delphi. Также существенно переработан IBSQLMonitor с тем, чтобы разделить платформо-зависимые детали, что позволило использовать на Linux платформах SV5 IPC. Для платформе Windows сохранен оригинальный Windows IPC. Чтобы обеспечить совместимость с Firebird Events был также обновлен модуль IBEvents,.

Кроме того была добавлена поддержка генераторов ключей, совместимая с той, что была добавлена в IBX уже после опубликования кодов. Генераторы могут использоваться в событиях "On New Record" или "On Post". Добавлено также несколько виджетов управления данными, распространяемых как часть пакета, а также процессор скриптов. Кроме того был обновлен компонент TIBExtract.

Начиная с IBX версии 2 пакет *fbintf* применяется как для новой, так и для старой версии Firebird API. Пакет fbintf частично является производным от IBX и , если доступно API Firebird 3, то автоматически загружает его, а если нет — старое API . Пакет *fbintf* распространяется в комплекте с IBX. Руководство Firebird Pascal API, поставляемое вместе с пакетом fbintf, содержит важную информацию об установке сервера Firebird для системы разработки и рекомендации по развертыванию.

Смотрите также документацию Firebird Pascal API Guide для получения информации о:

- Использование интерфейсов API с помощью IBX
- Наборы данных и их связь с кодовыми страницами AnsiString
- Развертывание приложений с использованием клиентской библиотеки Firebird.

Настоящее руководство предполагает, что пользователь имеет представление о среде разработки Lazarus (Lazarus Integrated Development Environment — IDE). Желательно также иметь представление о базах данных и СУБД Firebird. Для многих изучаемых тем приведены примеры приложений (см. 4.6 Примеры).

1.1 Ссылки

1. Руководство по InterBase 6 API (http://www.ibphoenix.com/files/60ApiGuide.zip)
2. Обзор языка Firebird 2.5 (http://firebirdsql.org/file/documentation/reference_manuals/fblangref25-en/html/fblangref25.html)
3. Руководство по определению данных InterBase 6 (http://www.ibphoenix.com/files/60DataDef.zip)
4. Замечания по реализации Firebird 3.0.1 (http://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html
5. IBX для Lazarus (MWA Software □ http://www.mwasoftware.co.uk/ibx)
6. Руководство по Firebird Pascal API □ MWA Software, 2016

1.2 История изменений

1.2.1 Версия 1.1

Эта версия была обновлена, чтобы включить :

- Расширенные возможности TIBExtract по выводу данных, включая простой формат XML для вывода двоичных данных BLOB и массивов полей с предоставлением прав доступа (grants) триггерам и хранимым процедурам.
- Расширенные возможности TIBXScript с тем, чтобы позволить обработку данных в формате XML, экспортируемых с помощью TIBExtract или для использования в операторе INSERT.
- Документацию по компонентам форматирования вывода (см. 7.2 Компоненты форматирования вывода).
- Исправить незначительные опечатки и неточности.

2 Установка и подготовка к использованию

IBX для Lazarus распространяется в виде единого архива (в формате zip или tar.gz) и включает пакет *fbintf*. Вы можете скачать последнюю версию с http://www.mwasoftware.co.uk/ibx. Архив следует распаковать в подходящий каталог вашей системы, недалеко от Lazarus IDE. Мы рекомендуем создать каталог "otherComponents" в каталоге, где установлен Lazarus и распаковать архив IBX в его подкаталог. В этом случае IBX будет расположен в :

"<Lazarus instal ation directory>/otherComponents/ibx".

2.1 Минимальные требования

Для IBX 2.0.0 требуется версият Lazarus не ниже 1.6.0 и версия Free Pascal Compiler не ниже 3.0.0. Поддерживаются все версии СУБД Firebird, включая 3.

Обязательно должна быть установлена клиентская библиотека Firebird. Если эта клиентская библиотека поддерживает новое API Firebird 3 , то используется оно, иначе используется более старое API Firebird 2.

2.2 Установка в среде Lazarus

Как минимум клиентская часть СУБД Firebird должна быть установлена на системе, где вы собираетесь использовать IBXЮ Инструкция по ее установке находится в руководстве Firebird Pascal API Guide.

Установка в среде Lazarus IDE происходит одинаково на плятформах Linux и Windows. Распакуйте архив с исходными кодами в подходящее место, как описано выше, и откройте файл пакета "dclibx.lpk" , используя пункты меню Lazarus IDE, "Package->Open Package File". После того как откроется Package Editor нажмите на "Use->Instal ". В результате Lazarus перекомпилирует самого себя (на что будет запрошено подтверждение) и перезапустится. ТРИ новых закладки должны появиться на Палитре компонентов: "Firebird", "Firebird Admin", и "Firebird Data Controls". На них, соотвественно, находятся — компоненты доступа к данным IBX и компоненты служб API. На третьей закладке появяться виджеты доступа к данным Firebird (Firebird Data Controls). Если компоненты IBX не появились, то наиболее вероятной причиной, является то, что клиентская библиотека Firebird не установлена или не может быть найдена. Смотрите руководство по Firebird Pascal API сведения о том, как пакет *fbintf* и, следовательно, IBX ищет клиентскую библиотеку Firebird.

2.3 Консольный режим IBX

IBX может использоваться как в качестве визуальных компонент в среде Lazarus так и в консольных приложениях. Для консольных приложений прилагается специальный пакет, в котором удалены зависимости от библиотеки LCL (например, диалог соединения с базой). Пакет называется "ibexpressconsolemode".

Для того, чтобы использовать пакет в консольном приложении надо в IDE выбрать "Packages->Open Package File" и в открывшемся диалоге выбрать файл ibexpressconsolemode.lpk, который находтся в корневом каталоге IBX. После выбора файла диалог следует закрыть. Ни устаналивать ни компилировать его не надо.

Достаточно открыть пакет, чтобы Lazarus его запомнил. Пример использования в консольном приложении находится в каталоге ibx/examples/fbsql.

2.4 Установка Firebird

Для того, чтобы использовать пакет *fbintf* , вы должны установить как минимум клиентскую часть СУБД Firebird. Это необходимо как для разработки так и для использования созданных приложений. Руководство по использования приводится в главе 13 Firebird Pascal API Guide.

Для систем разработки рекомендуется загрузить установочный пакет с сайта http://www.firebirdsql.org и установить полную систему, включая примеры использования. При этом будет установлена демонстрационная база данных "employee" и локальная СУБД Firebird для тестирования. Установить Firebird можно для операционных систем Linux, Windows и OSX. При использовании Linux также можно установить пакеты с примерами, однако они могут быть не самой последней версии. При использовании Debian/Ubuntu база данных с примерами прилагается в отдельном пакете и вам необходимо установить пакет, а также распаковать файл базы данных и установить доступ к ней, чтобы тестовые примеры правильно работали.

Парадоксально, но если вы не очень хорошо знакомы с Firebird и Linux, часто проще установить пакет firebirdsql, чем пакет из вашего дистрибутива .

После установки вы должны убедится, что "employee" правильно прописано в файле aliases.conf в установочном каталоге Firebird. Например, для 32-битного Firebird под Windows, файл C:\Program Files (x86)\Firebird\Firebird_2_5\aliases.conf должен содержать строку:
employee = C:\Program Files (x86)\Firebird\Firebird_2_5\examples\empbuild\employee.fdb

2.5 Обновление с более старой версии

Между файлами IBX2 и более ранними версиями имеется много отличий. Поэтому вы должны сначала удалить или переименовать каталог, содержащий более ранние версии IBX, а затем установить новую версию так, как описано в предыдущем разделе. Приложения, использующие IBX необходимо пересобрать, а не только перекомпилировать (используйте Run->Clean up and Build в меню Lazarus).

IBX2 содержит множество изменений в коде IBX нижнего уровня. Низкоуровневая «прослойка» связывающая Firebird API написанное на языке C и родной Паскаль выделены в новый пакет "fbintf" и теперь взаимодействие между IBX и этой «прослойкой» происходит посредством четко прописанного интерфейса на языке Паскаль. Предоставляются две реализации такого интерфейса: одна — для традиционного Firebird API , а другая — для новогоFirebird 3 API. По умолчанию, если возможно, IBX использует Firebird 3 API, в противном случае используется обычное Firebird API.

Ядро IBX было изменено так, чтобы использовать новый интерфейс. Также введена полная поддержка полей-массивов Firebird. Однако главный акцент был сделан на возможно более полной обратной совместимости, даже несмотря на значительные изменения в базовом коде. При переносе существующих IBX 1.4.x приложений на IBX 2.0.0 большинству пользователей для агрейда потребуется только перекомпиляция. Тем не менее, продвинутым пользователям может потребоваться внести изменения:

1. Модули IBIntf, IBCodePage и IBXConst следует удалить из Uses. Секция Uses, в которой используются IBIntf или IBXConst должны быть заменены на модуль "IB". Он теперь входит в состав пакета *fbintf* , который предоставляет Firebird Pascal API, в том числе определения констант и типов. IBCodePage был внутренним модулем, предоставляющим преобразование между наборами символов и кодовыми страницами. Аналогичная функциональность теперь предоставляется посредством Firebird Pascal API.
2. Модуля IBHeader в секции Uses следует заменить на модуль IB. Если после такой замены появятся ошибки компиляции, то причины, скорее всего, связаны с вызовами устаревшего Firebird API. Модуль IBHeader присутствует в новой версии, но он

содержит устаревшее API и любые его вызовы ведут к возможным проблемам в случае использования нового Firebird 3 API. Все такие зависимости следует выявить и заменить на соответствующую функциональность из Firebird Pascal API определенную в модуле IB.

3. В компоненте TIBSQL свойство SQLType переименовано в SQLStatementType. Это сделано для удаления потенциально неоднозначного имени свойства . Это свойство также используется входными и выходными метаданными для определения типов данных SQL.
4. В IBX2 автоматический старт или подтверждение транзакций больше не является действием по умолчанию. Это может повлиять на простые программы с единственным набором данных на форме, когда нет явного управления транзакциями. Если ваше приложение ранее использовало автоматический старт, то при открытии набора данных может возникать ошибка "transaction not active". Для повышения совместимости добавлено новое свойство AllowAutoActivateTransaction (see 5.2.4) к потомкам TIBCustomDataset. По умолчанию оно имеет значение false. Если его установить в true то восстанавливается прежнее поведение. Автоматический старт транзакций полноценно работает только для приложений с одним набором данных. В случае работы с несколькими наборами данных становится существенным порядок, в котором наборы данных закрываются (предполагается порядок, обратный порядку их открытия). В случае нескольких наборов данных транзакция может оставаться открытой, несмотря на закрытие наборов данных, позволяя правильно завершить транзакцию.
   Необходимость использовать явный старт транзакции может создавать неудобства, поскольку прежде чем осуществлять доступ к данным, программист должен быть уверен, что транзакция запущена, а иначе результат может оказаться ошибочным
   С другой стороны, важной особенностью является возможность открыть набор данных на этапе проектирования и видеть данные прямо в IDE.
   Если ваше приложение основано на автоматическом старте/завершении транзакций, то простейшим способом восстановить прежнее поведение можно, установив значение свойства AllowAutoActivateTransaction в true. Если ваше приложение использует несколько наборов данных на одной форме, то достаточно установить его только у того набора, который открывается первым. Такой набор данных должен также закрываться последним (устанавливать свойство active в false).
5. Свойство UniqueParamNames в настоящее время игнорируется и оставлено только для обратной совместимости. Уникальность имен параметров теперь определяется динамически.

2.6 Новые особенности IBX2

- Поддержка Firebird 3 API
- Доступ к Firebird Pascal API для выполнения встроенных SQL.
- Новое свойство IBDatabase: CreateIfNotExists. Если установлено в true, а файл базы данных отсутствует, то после попытки соединения (только на этапе выполнения) делается попытка создать базу данных.
- Новое событие IBDatabase: OnCreateDatabase. Это событие возникает после того, как база данных успешно создана в результате вызова CreateDatabase или в случае автоматического создания в случае CreateIfNotExists=true
- Добавлена поддержка массивов полей базы данных. Для этого добавлен новый класс полей (TIBArrayField) см. 9 Использование массивов Firebird — и соответствующий визуальный элемент управления, потомок TCustomStringGrid.

3 Введение в базы данных, SQL и Firebird

Эта глава рассчитана на тех, кто мало знаком с базами данных, языком структурированных запросов SQL и сервером баз данных Firebird. Читатели знакомые с этими вопросами могут проверить свои знания, а могут перейти сразу к следующей главе.

3.1 Что такое базы данных?

Согласно словарю — база данных не более, чем собрание данных. Такое определение ничего не говорит о том, как данные организованы и как к осуществляется к ним доступ. Некоторые базы данных и в самом деле могут быть просто кучей неструктурированных данных, тогда как другие могут быть хорошо структурированными с четко определенными правилами. Здесь мы будем иметь дело со вторым вариантом хранения данных, а первый оставляем для Google.

Базы данных, которыми управляет Firebird и для которых был разработан язык структурированных запросов (Structured Query Language SQL) , структурированы в соответствии со строгими правилами таким образом, что данные могут обрабатываться детерминированным образом и воспроизводимым результатом. Такой тип баз данных широко применяется в бизнес приложениях, таких как учет и управление складами, управление персоналом и заработной платой.

3.1.1 В самом начале

Начиная с 1960-х и вплоть до 1980-х, основными устройствами хранения данных для больших кампаний были магнитные ленты (учет, склады и т.п.). Магнитная лента является устройством, допускающим только последовательный доступ. Данные записываются на нее в виде «записей» и обычно упорядочены при помощи некоторых общих связей, таких как учетный номер или имя служащего. Каждая запись содержит также данные для учета или регистрационные подробности на служащего.

Базы данных на магнитных лентах должны были обрабатываться последовательно. Они могли требовать большого времени для поиска необходимой записи, так как чтение должно было начинаться с первой записи и происходить последовательно, читая, возможно, несколько лент. Изменение данных было трудоемкой процедурой, требующей подготовки специальной ленты, где требуемые изменения записаны в том же порядке, что на исходной ленте. Внесение изменений требовало слияния данных на текущей ленте и ленте подготовленных изменений, в результате чего формировалось новое множество данных на новой ленте.

3.1.2 Возникновение памяти с произвольным доступом

Применение дисковых устройств стало распространяться в 1970-х. Первоначально они были слишком дорогими для того, чтобы хранить полную базу данных и использовались для временного хранения (кэширования) данных и ускорения операций. По мере того, как они становились более емкими и дешевели, стало возможным с самого начала хранить всю базу данных на магнитном диске.

Магнитные диски позволяют осуществлять произвольный доступ. Это означает, что к любому сектору диска можно получить доступ за такое же время, что и к другим секторам. Это открывало возможность получить доступ к записи базы данных за малое время и даже редактировать запись «на месте», не меняя ее положение на диске. Однако все еще оставалась проблема, как найти нужную запись на диске? Если, по-прежнему, начинать чтение с первой записи и читать последовательно до обнаружения нужной записи, то доступ все равно будет очень медленным и многое будет зависеть от того, где находится запись в наборе данных.

3.1.3 Индексы

Решением было создание индексов, где индексом является сравнительно небольшая таблица ссылок, то есть таблица, по которой можно быстро определить где находится нужная запись. Индекс предназначен для использования вместе с селективным ключом, хранящимся в данных (таким как учетный номер). В принципе, индекс может быть просто таблицей с учетными номерами (ключ к записи) и адресом сектора (на диске), где записана соответствующая запись. Таблицу индексов можно просмотреть значительно быстрее, чем просматривать всю базу данных и значительно быстрее получить доступ к данным.

Для небольших баз данных достаточно простой таблицы ссылок. Однако для больших баз поиск в индексной таблице становится слишком накладным и требуется лучшее решение. В результате индексы стали делать более структурированными. Например, одна таблица для первой части учетных номеров, и отдельная таблица для второй части.

Другой подход состоит в том, чтобы на основании значения ключевого поля (например, учетного номера) генерировать хэш-значение и использовать его в индексной таблице для определения места расположения записи с нужным учетным номером. Разработка эффективного индексирования стало важным направлением исследований.

3.1.4 Множественные индексы и наборы данных

Конечно, нет причин ограничиваться только одним индексом. База данных может иметь несколько индексов для доступа к тем же данным — по одному для каждого ключа, который вы определили. Таким образом внимание смещается на структуру базы данных. База данных разбивается на несколько небольших наборов данных, каждый со своими индексами. А совокупность наборов данных и их индексов становится базой данных. С учетом сказанного, набор данных является последовательностью одинаковых записей, далее именуемых таблицей. В такой таблице каждая запись является строкой таблицы, а одинаковые поля образуют столбец таблицы.

Поначалу базы данных на магнитных лентах часто содержали дублирующиеся данные, разбросанные по разным базам. Уже поэтому было довольно трудно организовать одновременную обработку нескольких лент. При повсеместном распространении дисков возникла необходимость в устранении дублирования в наборах данных. Это, с одной стороны, устраняет риск иметь два различных значения для одной величины, а с другой стороны уменьшает необходимую дисковую память.

Однако, такой подход требует создания большого количества небольших наборов данных со своими индексами и создания программ, которые обеспечат доступ к данным, осуществляя одновременный доступ к нескольким наборам данных и «связывание» данных из разных таблиц.

3.1.5 Необходимость промежуточного программного обеспечения

С началом разработки приложений решающих общие проблемы, часто появляется промежуточное программное обеспечение и доступ к базам данных не был исключением.

Вскоре стало появляться множество промежуточных программных решений. Они предназначались для управления различными наборами данных и их индексами и предоставляли стандартный способ для связывания таблиц. Они освобождали клиентское приложение от необходимости открывать множество отдельных файлов, становясь единой точкой доступа к данным: провайдер базы данных.

3.1.6 Введение в СУБД (RDBMS)

Такое промежуточное ПО вскоре эволюционировало в Системы Управления Базами Данных (Relational Database Management Systems RDBMSs) как мы его теперь называем. Хотя различные продукты имеют некоторые отличия, все они:

- Управляют базой данных, состоящей из нескольких наборов данных, где каждый набор данных представляет собой таблицу, имеющую один или несколько индексов.
- Поддерживают метаданные (данные о данных), которые описывают каждую таблицу и каждый индекс.
- Предоставляют единый центр для доступа к базе данных из клиентских приложений.
- Обеспечивают доступ к данным с помощью таблиц или связывания таблиц друг с другом с использованием общих ключей, порождая большие виртуальные наборы данных (часто называемые просмотрами).
- Обеспечивают средства для оптимизации просмотров путем ограничения количества выводимых записей или полей. Таким образом улучшается и эффективность и безопасность данных (благодаря ограничению доступа к данным).
- Обеспечивают средства для вставки, удаления и изменения записей, включая операции над всей таблицей.

Добавив оптимизацию производительности, средства создания резервных копий и восстановления из них, средства контроля за целостностью данных, мы получаем то, что называют «Современная СУБД». Некоторые серверы СУБД все еще хранят различные наборы данных в разных файлах (например, некоторые версии MYSQL), тогда как другие помещают всю базу данных в единый файл со сложной структурой.

3.1.7 Многопользовательский доступ

Старые базы данных на магнитной ленте были, по своей природе, однопользовательскими. Однако современные СУБД могут обслуживать множество пользователей считывающих данные и даже изменяющих различные части базы данных. Такие возможности, в свою очередь, порождают риски конфликтов при попытке разных пользователей менять одну и ту же запись.

Одним из путей решения проблемы является блокировка записи или всей таблицы. Пользователь, который хочет изменить таблицу или отдельную запись, сначала блокирует ее, потом производит необходимые изменения, а потом снимает блокировку. Если в каждый момент времени только один пользователь может создать блокировку, то это позволяет блокировать все связанные записи, которые нуждаются в изменении, изменить их и снять блокировку, сохраняя целостность данных. Другим пользователям будет запрещено изменение этих записей до окончания действия блокировки, а возможно, и чтение их. Пользователи, возможно, будут вынуждены ожидать снятия блокировки.

В то время как механизм блокировки записей может рассматриваться как важнейший для редактирования базы данных, современные СУБД делают еще один шаг и развивают идею транзакций. В соответствии с этой моделью каждое соединение клиента с базой данных (которая может быть как локальной, так и удаленной) может создавать несколько активных транзакций. При этом транзакции:

- Имеют четкое начало и конец и существуют столько времени, сколько нужно клиенту.
- Создают контекст, в рамках которого осуществляется доступ к данным и их редактирование.
- «Владеют» необходимыми блокировками для таблиц и записей, курсорами базы данных и другими ресурсами, используемыми клиентом.

- Создают «изоляцию» между различными пользователями, в смысле управления видимостью изменений, производимых другими пользователями.
- Когда транзакция заканчивается, изменения, проделанные в рамках транзакции, можно подтвердить, записать в базу и сделать видимыми для остальных, либо откатить к тому состоянию, в котором база была в момент начала транзакции.

Транзакции позволяют каждому клиенту иметь согласованное представление данных и предотвратить противоречивость данных из-за конфликтующих изменений.

3.2 Язык структурированных запросов Structured Query Language (SQL)

Как уже обсуждалось выше, основной функцией СУБД является предоставление средств для определения и поддержания метаданных: описания таблиц и индексов. Необходимо также описание способов связи таблиц и фильтрации данных с целью создания просмотров, как постоянных так и временных, а также команд для редактирования базы данных.

Такие требования можно удовлетворить разными способами. Однако, именно SQL стал стандартом *de facto* для таких задач. Создание SQL восходит к IBM в 1970-х и использовался для достижения поставленных целей при помощи языка, синтаксически близкому к английскому. Он был стандартизован организацией ANSI в 1986 и стал международным стандартом в 1987. Серьезные изменения были внесены в 1992 (SQL-92) и с тех пор вносились только небольшие изменения. Несмотря на наличие международного стандарта, каждая СУБД имеет собственные реализации и, следовательно, свой диалект SQL. SQL принято подразделять на:

- Язык описания данных (Data Definition Language DDL), который используется для описания таблиц, индексов и просмотров, то есть для поддержания метаданных.
- Язык обработки данных (Data Manipulation Language DML), который используется для извлечения (select) данных из базы, а также для изменения, вставки и удаления записей.
- Введение в базы данных, SQL и Firebird
- Язык процедур и триггеров (Procedure and Trigger Language PSQL), который используется для определения операций с базой данных, которые могут быть затребованы клиентом или вызываться автоматически при изменениях данных. В последнем случае их часто используют для проверки правильности изменений.

Ниже приводится пример DDL, который служит для описания таблицы данных:

```
CREATE TABLE EMPLOYEE
(
    EMP_NO smallint NOT NULL,
    FIRST_NAME varchar(15) NOT NULL,
    LAST_NAME varchar(20) NOT NULL,
    PHONE_EXT varchar(4),
    HIRE_DATE timestamp DEFAULT CURRENT_TIMESTAMP NOT NULL,
    DEPT_NO char(3) NOT NULL,
    JOB_CODE varchar(5) NOT NULL,
    JOB_GRADE smallint NOT NULL,
    JOB_COUNTRY varchar(15) NOT NULL,
    SALARY numeric(10,2) DEFAULT 0 NOT NULL,
    PRIMARY KEY (EMP_NO)
);
```

В этом примере определяется таблица, состоящая из 10 столбцов. Для каждого столбца задается имя и тип данных. Главным ключом таблицы является номер служащего (EMP_NO) который должен быть уникальным значением для каждой строки.

Отметим, что операторы SQL всегда нечувствительны к регистру символов, включая имена столбцов (полей) данных (хотя более поздние расширения позволяют использовать регистрочувствительные имена и даже использовать в них специальные символы, заключая имена в двойные кавычки).

Описание таблицы включает также важное понятие, которое еще не обсуждалось, а именно, значение "NULL" . Если не использовано ограничение "NOT NULL" как в приведенном выше примере, то в каждой строке столбца таблицы может находиться либо значение, имеющее тип заданный для столбца, либо не иметь никакого значения (то есть NULL). Значения NULL могут быть использованы для поиска, могут использоваться для выборки данных и могут быть присвоены полю в качестве значения. Более подробную информацию по использованию NULL можно прочитать в Firebird Null Guide . Ниже приводится пример оператора Select. В нем создается временный виртуальный набор данных, который может быть прочитан клиентом.

```
Select FIRST_NAME,LAST_NAME,EMP_NO,HIRE_DATE FROM EMPLOYEES
Where LAST_NAME LIKE 'P%';
```

Набор данных, возвращаемый приведенным примером, содержит только четыре столбца и отфильтрован так, что в него войдут только те служащие, у кого фамилия начинается с буквы 'P'.

Замечание: для SQL запросов символ '%' означает любую строку.

3.3 СУБД Firebird

Firebird является примером Системы Управления Базами Данных (СУБД). Он является Open Source продуктом с лицензией, разрешающей его использование в коммерческих приложениях. Может потребоваться небольшая настройка с помощью System Manager и, как следствие, хорошо подходит для приложений МСП.

. Надо отметить, что создаваемые на его основе приложения обычно хорошо масштабируются. Firebird возник в 2000 году, когда Borland/Inprise выпустило программное обеспечение InterBase 6.0 под Open Source лицензией. InterBase имеет уже долгую историю (ознакомиться с ней можно на http://www.firebirdsql.org/en/historical-reference/) и Firebird унаследовал от InterBase большое сообщество пользователей.

Современные реализации Firebird характеризуются:

- использование многопользовательского режима, СУБД основана на транзакциях
- использование SQL для описания данных, редактирования данных, управления транзакциями, для определения процедур и триггеров.
- Firebird может быть развернута либо как встроенное ПО (встроенный сервер) , либо как автономный сервер, доступ к которому осуществляется с использованием протокола TCP/IP и поддерживающего как локальные, так и удаленные соединения.
- использует единый файл для хранения базы данных (с возможностью создания вторичных файлов для распределения данных по разным файловым системам).
- имеются реализации для различных программных платформ, включая Windows (32 и 64 бит), Linux (32 и 64 бит) и OSX.

Firebird поддерживает также концепцию "событий". Они представляют собой асинхронные сигналы, которые создаются с помощью PSQL и которые могут посылаться заинтересованным клиентам. Обычно они создаются с помощью триггеров для оповещения других пользователей об изменениях в данных.

3.4 Что собой представляет IBX ?

Firebird предоставляет клиентскую библиотеку (DLL под Windows, разделяемые объекты

(.so) под Linux) посредством которых осуществляется доступ к программному интерфейсу приложений (API Firebird). Это API является "низкоуровневым", предоставляя базовый набор функций, предоставляющих доступ к данным с помощью нетипизированных указателей. Такой подход является гибким, позволяя использовать API при помощи многих программных оболочек, но требует от клиентских приложений значительных усилий для придания смысла полученным данным.

Для программ на 'C' Firebird предоставляет препроцессор (gpre) который позволяет встраивать в код операторы SQL, генерирую код на языке C, необходимый для передачи этих операторов в Firebird для выполнения и получения выходных данных в виде структур языка C. Однако для языка Паскаль такой препроцессор отсутствует.

Когда в середине 90-х Borland выпустила Delphi, она стала, пожалуй, революционной разработкой в визуальном программировании. Delphi предложила и новую модель программирования баз данных, включающую абстрактную модель набора данных и «независимые от данных» ("data aware") элементы управления. Такой подход позволял связывать элементы управления (или виджеты), размещенные на форме, с полем в наборе данных. Это замечательно работало, если удавалось связать абстрактный набор данных с набором данных конкретной СУБД, которую хочет использовать разработчик.

Первые версии Delphi хорошо работали с таблицами Paradox и включали промежуточное ПО - Borland Database Engine (BDE) для предоставления доступа к базам данных SQL, включая InterBase (демоверсия поставлялась вместе с Delphi). Использование BDE было, пожалуй не самым эффективным решением задачи.

Одним из усовершенствований на этом пути стали Free IB компоненты, разработанные Джоджем Дейтцем (Gregory H. Deatz) для компании Longo, Moran, Dunst & Doukas в Голландии. Эта разработка была лицензирована компанией Borland и включена в состав Delphi в виде InterBase Express (IBX). IBX предоставляет прямую реализацию абстрактных наборов данных для доступа к InterBase, используя прямые вызовы API из программ на Delphi (Pascal) . Этот набор компонентов позволяет определять и изменять данные с помощью SQL , оставаясь в рамках модели набора данных используемой средой Delphi. Использование прямых вызовов к InterBase API в перспективе обеспечивает лучшую производительность для программ на Delphi. Когда Borland/Inprise выпустила InterBase под лицензией Open Source в 2000 году, она также опубликовала программный код IBX под той же лицензией. В 2011был выпущен клон IBX Open Source который был использован компанией MWA Software для создания *IBX для Lazarus*. Эта версия IBX была разработана для работы в составе Lazarus LCL и для использования СУБД Firebird как в качестве встроенного, так и автономного сервера баз данных. По сравнению с первоначальной версией, *IBX для Lazarus* была улучшена и расширена путем введения дополнительных компонентов, включая парсер SQL и подсистему выполнения скриптов. В 2016 году IBX2 включила поддержку нового Firebird 3 API. Эта версия включает специальный набор процедур Pascal Language Bindings (пакет *fbintf*) который является основой IBX2. Пакет *fbintf* можно использовать для эффективного встраивания операторов SQL в программы на Паскале.

4 Обзор IBX

Назначением IBX является создание модели набора данных TDataset, предоставляя таким образом источник данных для виджетов управления данными. Эта задача решается с помощью прямых обращений к Firebird API. При этом отсутствие промежуточных звеньев обеспечивает высокую производительность. IBX нацелена на достижение наилучшей производительности при использовании программ на Паскале. Firebird является СУБД основанной на SQL и знание SQL является необходимым для всех применений IBX, кроме самых простых. IBX не пытается скрыть SQL от программиста. Вместо этого IBX предоставляет программисту возможность полноценного использования возможностей SQL.

Компоненты *IBX для Lazarus* должны вести себя также, как и их аналоги в Delphi и существует множество руководств по их использованию. Ниже дается введение по их использованию, а также прилагается множество примеров программ.

4.1 Конвертирование из Delphi IBX

При конвертировании вы должны иметь в виду следующее:

1. Компоненты IBX используют класс TThread и, следовательно, требуют, чтобы была разрешена многопоточность. В частности, в среде Linux в IDE должна быть включена опция "-dUseCThreads" посредством "Compiler Options->Options->Custom Options" и она должна быть установлена для всех проектов Lazarus, которые используют IBX.
2. При портировании программ из Delphi в Lazarus с компилятором младше FPC 2.6.0 использование TintegerField может вызывать проблемы.
3. Компонента FMTBcd не имеет реализации под FPC. Для представления таких полей IBX для Lazarus использует тип полей TFloatField (расширенные с плавающей точкой 64 бит данные). Это может вызвать проблемы при портировании Delphi программ в Lazarus. Рекомендуется изменить тип данных. Исключением являются компоненты TIBTable и TIBStoredProc. Они могут использоваться в простых приложениях с базами данных, не требуя использования программирования на SQL.

При преобразовании Delphi форм к Lazarus поля TFmtBcdField преобразуются к TIBBcdFields. Однако, иногда преобразование не дает нужного результата. Обычно это приводит к тому, что отображаемое значение на много порядков отличается от реального. Для решения проблемы удалите и снова создайте проблемное поле с помощью редактора полей. При этом будет создано поле с правильным типом данных

В качестве альтернативы можно удалить поля TFmtBcdField еще до преобразования, а затем создать их при помощи IDE.

4.2 Место IBX в программировании баз данных

Ниже приведена диаграмма, в которой сделана попытка показать место IBX по отношению к другим пакетам. Как видно, пакет *fbintf* является поставщиком данных от Firebird API и может непосредственно использоваться пользователем для обращений к Firebird API из IBX. Пользователь непосредственно обращается к IBX, но при этом использует и FCL где находится описание абстрактного класса TDataset. IBX позволяет использовать и компоненты LCL, но делает это только для не консольного режима и только для обеспечения встроенного диалогового окна входа .

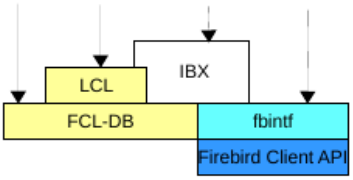


Рисунок 1: Место, которое занимает IBX по отношению к другим пакетам

4.3 Обзор компонент IBX

На закладке Firebird палитры компонентов находятся следующие компоненты:

TIBDatabase Каждый проект, который использует IBX должен содержать хотя бы одну компоненту TIBDatabase. Обычно она находится в модуле данных и представляет соединение с базой данных. В его свойствах прописан сервер, на котором находится база данных, ее имя или путь на сервере, параметры входа и набор символов сервера, если он отличается от клиентского набора. Также он может генерировать форму для подключения к серверу с указанием логина и пароля или использовать для этого специально созданный пользователем диалог. Подробнее см. 5.1 TIBDatabase.

TIBDatabaseInfo Этот компонент тесно связан с TIBDatabase и предоставляет доступ к свойствам базы данных и статистике обращений в режиме «только для чтения». Подробнее см. раздел 7.5 Компонент TIBDatabaseInfo.

TIBTransaction Каждый проект, который использует IBX должен содержать хотя бы одну компоненту TIBTransaction. Firebird является ориентированной на транзакции СУБД и все операции с базой данных должны происходить в контексте какой-либо транзакции. Его свойства определяют уровень изоляции транзакции (см. документацию Firebird). Компонент TIBTransaction обычно создается вместе с TIBDatabase и связан с ним посредством свойства DefaultTransaction. См. 5.2 TIBTransaction.

TIBQuery Этот компонент является потомком TDataset в качестве результата выполнения SQL запроса (оператора Select или выполнения хранимой процедуры) создает набор данных . Используемый запрос SQL задается с помощью его свойства SQL. Запрос может содержать параметры, которые должны получить значения прежде, чем запрос будет выполнен. При установке свойства "active" в значение true запрос выполняется и возвращается результирующий набор данных. При установке свойства "active" в значение false временный набор данных удаляется.
Свойства компонента должны определять базу данных и транзакцию к которой относятся операции с данными. См. 6.5 TIBQuery.

TIBUpdateSQL Этот компонент может быть связан с одним или несколькими компонентами TIBQuery и используется для создания редактируемых запросов. Он предоставляет операторы SQL для:

- Удаления текущей записи в наборе данных
- Обновления (из базы данных) текущей записи в наборе данных
- Изменения текущей записи для модификации соответствующих значений в таблицах базы данных.
- Вставки новых строк в базу данных.

См. 6.6 TIBUpdateSQL.

TIBDataSet Этот компонент также является потомком TDataset и соединяет функциональность TIBQuery и TIBUpdateSQL. См. 6.7 TIBDataSetСвойства компонента должны определять базу данных и транзакцию к которой относятся операции с данными.

Обычно удобнее использовать TIBDataset вместо пары TIBQuery и TIBUpdateSQL компонент. Главной целью использования пары компонент может быть, к примеру, форма, использующая TIBQuery для доступа к набору данных только для чтения и производная форма (потомок) нуждающаяся в редактировании набора данных. В этом случае можно добавить к форме-потомку компонент IBUpdateSQL чтобы обеспечить возможность редактирования.

 **TIBStoredProc** Этот компонент используется для выполнения хранимых процедур (на сервере базы данных). В том числе таких, которые не создают наборов данных. Свойства компонента должны определять базу данных и транзакцию для выполнения запроса. См. 6.4 TIBStoredProc.

 **TIBSQL** Этот компонент является базовым для обработки SQL с помощью IBX и используется внутри TIBQuery, TIBDataset и TIBStoredProc для выполнения SQL запросов. Программист может использовать его непосредственно для повышения эффективности встроенных SQL операторов. Свойства компонента должны определять базу данных и транзакцию для выполнения запроса.
TIBSQL является главным образом объектно-ориентированной инкапсуляцией Firebird DSQL API. См. 6.5 TIBQuery.

 **TIBEvents** Одной из очень полезных особенностей баз данных Firebird является возможность генерации «событий» при выполнении хранимых процедур или триггеров, которые впоследствии могут быть получены любым активным клиентом, который настроен на получение событий. Таким образом клиенты базы данных могут немедленно реагировать на изменения, сделанные другими клиентами, без необходимости делать постоянные запросы к базе.
Компонента TIBEvents используется для регистрации и получения событий, генерируемых базой Firebird. Можно настроить получение до 16 событий в одном компоненте. Имена событий, которые будет получать компонент, устанавливаются в его свойстве Events. Если необходимо получать более 16 событий, то можно использовать еще один компонент TIBEvents. Извещения о событиях возникают асинхронно и создаются в момент завершения транзакции в которой были сгенерированы события. Ожидание событий происходит в отдельном потоке, используемом TIBEvents. При поступлении события вызывается обработчик "OnEventAlert" для определения какое именно событие было получено. Отметим, что обработчик событий выполняется в контексте главного потока и, следовательно не нужно беспокоиться о синхронизации выполнения потоков. См. 5.3 TIBEvent.

 TIBSQLMonitor Этот компонент служит для отладки и настройки производительности, позволяя одному процессу следить за выполнением вызовов SQL функций в этом же или ином процессе (в той же операционной системе).
Прописанные в компоненте TIBDatabase флаги трассировк определяют те вызовы, которые будут отслеживаться в течении данного соединения с базой. Однако, процесс мониторирования начинается только после явного вызова процедуры IBSQLMonitor.EnableMonitoring и прекращается после вызова IBSQLMonitor.DisableMonitoring.
Для выполнения трассировки вызовов SQL функций, вам нужно поместить на форму компонент TIBSQLMonitor. Свойства этого компонента могут быть установлены так, чтобы отбирать те вызовы SQL, которые вам нужны. Для получения и обработки вызовов SQL функций служит событие OnSQL. См. 7.4 Монитор ISQL.
Отметим, что используемая в Windows реализация позволяет любому процессу мониторировать SQL, рассылая события другим процессам. В Linux реализация ограничена мониторированием только тех процессов, которыми владеет тот же пользователь.

 TIBTable Этот компонент представляет собой простой потомок TDataset,

который является набором данных, соответствующим одной таблице базы данных. Этот компонент удобно использовать для простых приложений, но для большинства а приложений предпочтительно использование TIBDataset. Компонент поддерживает также отношения главная/подчиненная для связанных таблиц. См. 6.3 TIBTable.

 TIBExtract Этот компонент служит для извлечения из базы метаданных. Компонент предназначен для поддержки расширений языка DDL включенных в реализацию Firebird 3. См. 7.6 Компонент TIBExtract

 TIBBatchMove Этот компонент предназначен для переноса данных между таблицами базы данных.

 TIBXScript Этот компонент используется для выполнения SQL скрипта, записанного в файле или потоке данных. Текст скрипта разбивается на SQL операторы, которые выполняются последовательно. По замыслу, компонент является совместимым с диалектом ISQL и его расширениями. См. 7.1 Процессор скрипов IBX.

Закладка Firebird Admin содержит компоненты Service API. Они поддерживают некоторую функциональность на стороне сервера, включая управление пользователями и резервное копирование/восстановление базы данных (см. раздел 10.1 Обзор компонентов Firebird Admin).

4.4 Базы данных и транзакции

IBX доступ к базе данных всегда осуществляет с помощью «соединения» , как к локальной, так и к удаленной базе. Данные считываются или записываются в контексте транзакции. Транзакции служат:

- для изолирования пользователей друг от друга,
- для того, чтобы пользователь видел свое представление данных, не зависящее от того, что делают другие,
- чтобы позволить пользователю накладывать блокировку на редактируемые данные, а остальным ожидать снятия блокировки
- транзакция задает четко определенные начало и конец сеанса редактирования данных, а в конце — возможность подтвердить сделанные изменения и сделать их доступными для других или отбросить их и откатить базу к тому состоянию, которое было на момент начала транзакции. Firebird является ориентированной на транзакции СУБД и все взаимодействия с базой данных происходят в контексте транзакций. Firebird позволяет одновременно производить множество независимых транзакций и предоставляет механизм разных способов изолирования для того, чтобы разные транзакции не мешали друг другу.

В приложении использующем IBX должны быть, как минимум, один компонент TIBDatabase (см. 5.1 TIBDatabase) и один компонент TIBTransaction (см. 5.2 TIBTransaction). Обычно они размещаются на главной форме приложения или в модуле данных. Компонент TIBDatabase предоставляет соединение с базой данных и только в редких случаях приложению нужны другие такие компоненты (например, когда нужно соединение с двумя или более базами данных).

В свойствах компонента TIBDatabase задается размещение базы данных и параметры подключения к ней. TIBDatabase может вызывать встроенный диалог для задания имени пользователя и пароля или использовать диалог, предоставляемый приложением. В свойствах TIBDatabase также задается транзакция по умолчанию. Наборы данных создаваемые на основе этой базы данных автоматически используют эту транзакцию, если она не будет явно заменена на другую

Компонента TIBTransaction задает транзакцию и вы можете использовать сколько угодно

таких компонент. TIBTransaction обычно связан с одной базой данных (компонентом TIBDatabase), но может быть связан и с несколькими — для того, чтобы синхронизировать проделанные в них изменения.

4.5 Наборы данных

Набор данных TDataset является связующим звеном между виджетами доступа к данным и таблицами базы данных как в Lazarus, так и в Delphi. Потомки TDataset , в частности, используются для представления данных одной таблицы базы данных. Виджеты доступа к данным (такие как TDBEdit) могут быть связаны с единственным столбцом (текстового типа) в наборе данных и позволяют просматривать и редактировать данные одной строки этого столбца. Такие виджеты данных, как TDBGrid позволяют видеть несколько записей в виде таблицы.

На рисунке 2 приводится пример. Этот скриншот примера приложения из каталога "ibx/examples/employee" демонстрирует использование TDBGrid для отображения таблицы базы данных.

4.5.1 Наборы данных и транзакции

Каждый набор данных связан с какой-либо одной транзакцией и все операции чтения или записи этого набора происходят в контексте этой транзакции.

В простых приложениях вам достаточно использовать один компонент TIBTransaction на все приложение.

В большинстве случаев тот набор данных, который активируется первым, неявно стартует транзакцию (если установлено свойство AutoActivateTransaction) и, если при закрытии набора данных отсутствуют другие активные наборы, то транзакция автоматически подтверждается (коммитится). При закрытии базы данных TIBTransaction по умолчанию подтверждает все изменения.

В более продвинутых приложениях (как в приводимом примере), вам, возможно, захочется самим управлять подтверждением или откатом транзакций и TIBTransaction предоставляет методы для запуска и подтверждения или отката транзакции. При этом обычно желательно явно запускать каждую транзакцию, а не полагаться на неявный старт, чтобы предотвратить нежелательное подтверждение при закрытии набора данных. В этом случае подтверждение или откат всегда будут происходить под управлением программы.

Отметим, что при завершении транзакции все наборы данных, связанные с ней, автоматически закрываются. Чтобы продолжать редактирование набора, необходимо вновь запустить транзакцию и переоткрыть нужные наборы данных.

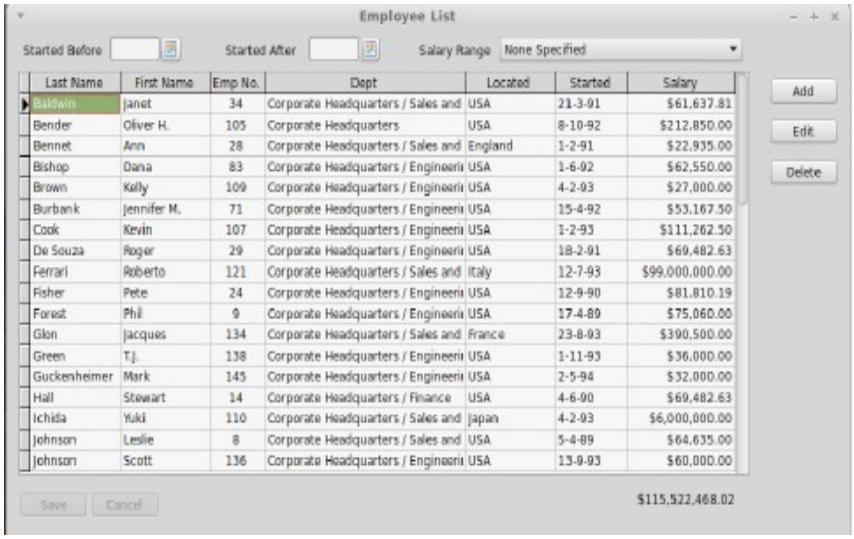


Рисунок 2: Список служащих из примера приложения Employee

Однако, существует возможность сохранить изменения (подтвердить транзакцию) без закрытия наборов данных. Это можно сделать с помощью метода TIBTransaction.CommitRetaining. Он подтверждает транзакцию, сохраняя ее контекст. В этом случае наборы данных останутся открытыми. Недостатком использования такой функции является то, что в случае многопользовательской базы данных другие пользователи узнают о сделанных изменениях только после переоткрытия данных.

4.5.2 Набор данных на основе одной таблицы

В случае несложных приложений, состоящих из единственной таблицы, набор данных обычно является результатом запроса, состоящим из одной или нескольких записей. В некоторых случаях такой набор состоит из единственной строки и достаточно одной формы, которая позволяет просматривать и редактировать ее содержимое. Такой подход предотвращает считывание из базы данных многих записей, когда нужна только одна.

В IBX компонет TIBTable является потомком TDataset, позволяющий читать и редактировать содержимое одной таблицы базы данных. Для этого он автоматически генерирует операторы SQL, необходимые для доступа и редактирования данных. Однако, такой подход имеет ограничения и не позволяет использовать всю мощь SQL.

4.5.3 Наборы данных на основе SQL

IBX предоставляет также компонент TIBQuery. Он тоже является потомком TDataset, но считывает набор данных, полученный в результате выполнения оператора выборки Select, заданного программистом на этапе проектирования или во время выполнения программы. Запрос может соединять несколько таблиц и ограничивать количество возвращаемых строк. Это открывает гораздо больше возможностей, чем доступ к единственной таблице.

Если вам необходимо редактировать базу данных на основе набора данных TIBQuery, то можно подключить другой компонент (TIBUpdateSQL), чтобы задать операторы SQL для вставки редактирования, удаления записей, а также для синхронизации данных с текущим состоянием базы.

Вообще говоря, набор данных TIBDataset является компонентом, который позволяет задать операторы Select, Update, Insert, Delete and Refresh.

4.6 Примеры

В каталоге ibx/example/employee приводится пример приложения, которое использует компоненты TIBDatabase, TIBTransaction, TIBQuery и TIBDataset, а также виджеты доступа к данным для просмотра и редактирования базы данных EMPLOEE

5 Компоненты доступа к базе данных

Представленные здесь компоненты IBX разбиты на три главных группы. В этом разделе рассматриваются компоненты доступа к базе данных ("Database Access"). Раздел 6 посвящен рассмотрению компонент — наследников TDataset, а раздел 7 — компонентам, обеспечивающим администрирование базы.

Компоненты IBX доступа к данным (Database Access) представляют собой компоненты, которые являются немногим больше, чем просто «оберткой» для интерфейса экспортируемого пакетом *fbintf*» и состоят из:

- TIBDatabase — реализует интерфейс IAttachment
- TIBTransaction — реализует интерфейс ITransaction
- TIBEvent — реализует интерфейс IEvents
- TIBSQL — реализует интерфейс IStatement.

Документация по Firebird Pascal API детально описывает каждый из этих интерфейсов и способы их использования. Однако, при использовании IBX предпочтительно использовать методы и свойства предоставляемых компонент, а не методы и свойства нижнего уровня, за исключением тех случаев, о которых будет сказано отдельно.

5.1 TIBDatabase

Этот невизуальный компонент является потомком TCustomConnection. Он представляет связь абстрактного соединения с базой в виде наборов данных и конкретной СУБД Firebird. Здесь мы выделим следующие свойства:

Connected — Установите это свойство в true для соединения с базой данных и в false для отключения.

AllowStreamConnected — Это свойство введено, чтобы исключить конфликты между соединением во время проектирования и автоматическое соединение во время выполения. (см. 5.1.8 Использование свойства AllowStreamConnected).

CreateIfNotExists — Новое в IBX2: Если установлено в true, и делается попытка соединения с базой, которая не существует, то если возможно, база данных создается. См. также событие OnCreateDatabase.

DatabaseName — Путь к базе данных, включая путь к серверу. Формат может различаться в зависимости от протокола соединения и правил написания пути для разных операционных систем. Подробности смотрите в документации на Firebird.

DefaultTransaction — Указывает транзакцию по умолчанию для базы данных. Это чисто условная вещь. Когда для набора данных указывается база данных, то свойству Transaction набора данных автоматически присваивается транзакция по умолчанию, если не было других присвоений.

IdleTimer — Если не равно нулю, то это время в миллисекундах, между последовательными опросами активности базы данных. Если активность отсутствует в течении двух последовательных опросов, то база данных автоматически отключается. При необходимости можно использовать для отключения простаивающих соединений по тайм-ауту.

LoginPrompt — Если равно true, то выводится диалог для соединения пользователя с базой данных для ввода имени пользователя и его пароля. Если задан обработчик события OnLogin, то он выполняется для создания собственного диалога входа. В

противном случае вызывается встроенный диалог.

Params Содержит список параметров и их значений, используемых в Database Parameter Block (DPB) при соединении с базой. Записываются в формате "имя_параметра=значение". См. также ниже.

SQLDialect SQL диалект по умолчанию, используемый для соединения (рекомендуется 3). См. документацию на Firebird.

SQLHourGlass Если установлен в true, то во время запросов к серверу базы данных курсор меняет свой вид на песочные часы.

UseSystemDefaultCodePage Если установлено в true, то в качестве набора символов базы данных используется кодовая страница по умолчаниют. Не рекомендуется использовать в программах на Lazarus, где для многих LCL функций используется набор символов UTF8.

5.1.1.1 Названия параметров в списке Params

Параметры предпочтительно редактировать на этапе проектирования, используя диалог подключения к базе данных (вызывается двойным кликом по иконке компонента TIBDatabase). Доступные имена параметров:

user_name имя пользователя

password пароль (использование этого параметра во время проектирования не рекомендуется).

lc_ctype имя набора символов по умолчанию (например,. UTF8). Для программ на Lazarus рекомендуется UTF8.

5.1.2 Основные события

События TIBDatabase обычно используются для выполнения действий при изменении состояния соединения. Список событий :

AfterConnect Вызывается после того, как соединение успешно установлено. Хорошее место для старта транзакций и открытия наборов данных.

AfterDisconnect Вызывается после того, как соединение разорвано.

BeforeConnect Вызывается перед попыткой подключения к базе данных. Может использоваться для изменения параметров соединения.

BeforeDisconnect Вызывается перед попыткой отключения от базы данных.

OnCreateDatabase Вызывается после успешного создания базы данных. Может использоваться для запуска скрипта DDL (например, используя TIBXScript см. 7.1 Процессор скрипов IBX) для инициализации базы данных.

OnIdleTimer Вызывается после того, как база данных была отключена из-за истечения idletimer.

OnLogin Вызывается, если LoginPrompt установлен в true и может быть использован для заполнения параметров входа запрашиваемых у пользователя. Если обработчик отсутствует, а LoginPrompt установлен в true, то используется стандартный диалог входа.

5.1.3 Соединение с базой данных

Рекомендуемый порядок действий:

- Установить AllowStreamedConnect в false
- Не записывать пароль в параметрах
- Использовать встроенный диалог для ввода логина и пароля. Ниже приводится код, рекомендуемый для установки свойства connected в true. Он может быть включен в код обработчика OnShow главной формы или OnCreate модуля данных:

```
repeat
   try
      IBDatabase1.Connected := true;
   except
      on E:EIBClientError do
      begin
         Close;
         Exit
      end;
      On E:Exception do
         MessageDlg(E.Message,mtError,[mbOK],0);
   end;
until IBDatabase1.Connected;
```

Пример такого кода можно найти в каталоге "ibx/example/employee". Назначением приведенного выше кода является вывод в случае неудачи сообщения об ошибке и позволить пользователю исправить ошибку или закончить выполнение, нажав cancel.

Замечание: чтобы использовать приведенный вариант кода, вам нет необходимости включать модуль "IB" в вашу секцию uses.

5.1.4 Отключение от базы данных

Вы можете отключиться от базы данных в любой момент, установив значение свойства Connected в false.

Однако, при завершении программы нет необходимости явно отключаться, поскольку это происходит автоматически при разрушении компонента TIBDatabase.

5.1.5 Создание новой базы данных

Для явного создания новой базы данных можно использовать метод CreateDatabase, а можно установить свойство CreateIfNotExists в значение true. В этом случае, если попытка соединения закончилась ошибкой «база данных не найдена», то будет сделана попытка автоматически создать базу данных.

Для определения пути к базе данных используется свойство DatabaseName , а свойство Params содержит параметры создания, включая имя собственника базы (user_name) и набор символов по умолчанию (lc_ctype). Свойство SQLDialect задает диалект базы данных.

Кроме того, можно вызвать метод CreateDatabase с оператором "CREATE DATABASE". В этом случае оператор SQL будет единственным источником параметров создаваемой базы. После создания базы данных вызывается обработчик события OnCreateDatabase. В нем можно выполнить инициализацию базы данных с помощью SQL скрипта (например, с помощью TIBXScript см. 7.1 Процессор скрипов IBX).

5.1.6 Удаление базы данных

После установления соединения с базой можно удалить (отключиться и стереть) базу данных, используя метод DropDatabase. При этом, если пользователь, подключившийся к базе, имеет достаточно прав на эту операцию, то будет удален файл базы данных.

5.1.7 Использование интерфейса Attachment

Компонент TIBDatabase также предоставляет интерфейс нижнего уровня IAttachment в виде опубликованного свойства Attachment. Оно может использоваться для встроенного SQL. Например,

```
var employees: integer;
begin
  employees := IBDatabase1.Attachment.OpenCursorAtStart(
  'Select count(*) From Employees').AsInteger;
```

получает текущее количество строк в таблице employe. Подробное описание использования интерфейса IAttachment приводится в руководстве Firebird Pascal API.

5.1.8 Использование свойства AllowStreamConnected

Для того, чтобы получать списки таблиц и полей и тестирования операторов SQL редакторы свойств (см. 6.5.3 Редактор свойства SQL) нуждаются в соединении с базой данных. Для этого они используют компоненту TIBDatabase и могут устанавливать ее свойство connected в true. Однако, это может привести к тому, что при сохранении формы свойство connected окажется равным "true".

В результате при запуске откомпилированной программы компонент TIBDatabase будет загружен со свойством connected установленным в true. Вот это и называется состояние "streamed connected" (потоковое подключение). В этом случае соединение с базой будет установлено сразу по окончании загрузки компонента. Возможно, именно этого вы и хотели — тогда все нормально. Но зачастую это нежелательно. Если TIBDatabase соединяется с базой сразу после загрузки компонента, это может вызвать проблемы при обработке ошибок. При таком исключении ошибка не может быть локализована и простая ошибка, такая как неверно введенный пароль, может привести к прекращению работы. При этом пользователь может думать, что программа падает по непонятным причинам.

Чтобы избежать такой ситуации, предпочтительно устанавливать свойство connected явно, например, по окончании загрузки главной формы или в обработчике OnShow (см. 5.1.3 Соединение с базой данных). В этом случае можно поместить "connected := true" внутрь обработчика исключений.

Этого можно достичь путем установки свойства AllowStreamedConnection в false. Такое значение препятствует переходу компонента в состояние "streamed connected" даже если свойство connected было установлено в true при сохранении формы.

5.2 TIBTransaction

Этот невизуальный компонент является оберткой для интерфейса ITransaction. Он служит для того, чтобы:

- Предоставить компонент реализующий транзакцию

- Предоставить средство для задания параметров транзакции во время выполнения.
- Позволить устанавливать связи между базой данных, наборами данных и транзакциями на этапе проектирования .
- Предоставить единое место для обработки событий при запуске или завершении транзакции или для выполнения действий при изменении набора данных связанного с транзакцией (например для, информирования об изменении данных).

Для изменения параметров транзакции на этапе проектирования проще всего использовать его редактор свойств. Его можно вызвать, щелкнув правой кнопкой на иконке транзакции и выбрав соответствующий пункт меню.

5.2.1 Ключевые свойства

Active — Установка в true начинает транзакцию. Возвращает true, если транзакция активна. Установка в false завершает транзакцию и откатывает ее. Установка значения true во время проектирования приводит к тому, что транзакция стартует сразу после загрузки компонента.

DefaultAction — Может принимать значения taCommit или taRollback. Определяет действие по умолчанию в случае неявного завершения транзакции.

DefaultDatabase — Ссылка на компонент TIBDatabase для транзакции.

IdleTimer — Если не рано нулю, то задает время в миллисекундах для последовательных опросов активности транзакции. Если активность не обнаруживается в течении двух последовательных опросов, то транзакция автоматически закрывается. Может использоваться при необходимости автоматического завершения транзакции по таймауту.

Params — Содержит список значений параметров используемых в блоке параметров транзакции (Transaction Parameter Block TPB) в момент соединения. Записываются в виде "ключевое_слово=". См. ниже

5.2.2 События

AfterDelete — Вызывается после удаления записи в наборе данных, связанном с транзакцией.

AfterEdit — Вызывается после перехода набора данных, связанного с транзакцией в состояние редатирования.

AfterExecQuery — Вызывается после того, как компонента TIBSQL, связанная с транзакцией выполнила запрос.

AfterInsert — Вызывается после перехода набора данных, связанного с транзакцией в состояние вставки записи

AfterPost — Вызывается после того, как набор данных, связанный с транзакцией зафиксировал изменения.

AfterTransactionEnd — Вызывается после завершения транзакции.

BeforeTransactionEnd — Вызывается перед завершением транзакции.

OnIdleTimer — Вызывается в случае завершения транзакции в виду отсутствия активности (по таймауту).

OnStartTransaction — Вызывается после старта транзакции.

5.2.3 Базы данных и транзакции

В большинстве случаев транзакции бывают связаны только с одной базой данных (DefaultDatabase) и транзакции происходят в контексте связанного с базой данных соединения. Однако, транзакции могут быть связаны и с несколькими базами данных.

Дополнительные базы данных добавляются во время выполнения с помощью вызова метода AddDatabase перед запуском транзакции. Дополнительную базу можно также удалить из транзакции с помощью вызова RemoveDatabase. Для получения списка баз данных, связанных с транзакцией можно использовать открытое (public) свойство Databases. Если более, чем одна база связана с транзакцией, то транзакция становится мульти-базовой для координации изменений нескольких баз данных.

5.2.4 Запуск транзакций

Транзакция может быть запущена автоматически, например, в результате открытия набора данных, а может быть и явно, путем установки свойства Active в true или путем вызова метода StartTransaction.

Замечание: Начиная с IBX2 неявный запуск транзакции во время выполнения должен быть разрешен с помощью свойства AllowAutoActivateTransaction.

Неявный запуск происходит, если связанный с транзакцией набор данных открывается, а транзакция еще неактивна. Неявный запуск происходит также, если свойство транзакции Active установлено в true во время проектирования. В этом случае запуск происходит сразу после загрузки компонента на форму во время выполнения.

Явный запуск транзакции можно делать в любой момент, если база данных соединена с сервером. Однако, если транзакция уже запущена, то возникнет исключение. Рекомендуемый код выглядит так:

```
IBTransaction1.Active := true;
```

Рекомендуется явный старт транзакции для всех приложений, кроме самых простых, поскольку он позволяет лучше управлять обработкой ошибок и помогает правильному пониманию использования транзакций.

5.2.5 Параметры транзакции

Ниже приводится список ключевых слов для параметров транзакции. За исключением специально оговоренных случаев, эти параметры не имеют связанных с ними значений и, следовательно, не содержат в конце знака"=".

Параметры транзакции можно также задать с помощью Редактора транзакций (см. 5.2.6 Редактор транзакций).

consistency	Модель транзакций с блокировкой таблиц.
throughput	Обеспечивает высокую пропускную способность при приемлемой согласованности. Использование этого параметра позволяет использовать преимущества модели множественных транзакций, используемую Firebird [Значение по умолчанию]
shared	Параллельный, разделяемый доступ к указанной таблице для всез транзакций. Используется в сочетании с lock_read и lock_write для установления

дополнительных блокировок [Значение по умолчанию]

protected Параллельный, ограниченный доступ к указанной таблице для всез транзакций. Используется в сочетании с lock_read и lock_write для установления дополнительных блокировок

exclusive То же самое, что и "protected"

wait В случае блокировки указывает, что транзакция должна ожидать снятия блокировки перед повторной попыткой выполнения операции [Значение по умолчанию]

nowait В случае блокировки указывает, что транзакция не должна ожидать снятия блокировок, а должна немедленно возвратить ошибку конфликта блокировок

read Доступ только для чтения позволяет транзакциям только операции выборки

write Доступ на чтение-запись разрешает транзакции операции выборки, вставки, редактирования и удаления данных [Значение по умолчанию]

lock_read Доступ только для чтения для указанной таблицы. Используйте совместно с shared, protected, или exclusive параметрами блокировки . Имя таблицы задается в качестве значения параметра.

lock_write Полный доступ для указанной таблицы. Используйте совместно с shared, protected или exclusive параметрами блокировки . Имя таблицы задается в качестве значения параметра.

verb_time *Этот параметр плохо документирован и его использование туманно. Используйте его только если вы точно знаете, что делаете.*

commit_time *Этот параметр плохо документирован и его использование туманно. Используйте его только если вы точно знаете, что делаете*

ignore_limbo Не ожидать Лимбо транзакции. Используется потоком сборщика мусора.

read_committed Высокая пропускная способность, транзакция с высоким параллелизмом, которая может считывать изменения, зафиксированные другими параллельными транзакциями . Использование этого параметра позволяет использовать преимущества модели множественных транзакций, используемую Firebird .

autocommit Сервер делает автоматическое подтверждение всех изменений.

rec_version Позволяет транзакции "read_committed" считывать последнюю зафиксированную версию записи, даже если есть другие незафиксированные версии.

no_rec_version Позволяет транзакции "read_committed" считывать только последнюю зафиксированную версию записи. Если есть незафиксированные версии записи и указано"wait", то транзакция для продолжения ожидает подтверждения или отката изменений. В противном случае генерируется исключение об ошибке блокировки.

restart_requests *Этот параметр плохо документирован и его использование туманно. Используйте его толко если вы точно знаете, что делаете.*

no_auto_undo В этом случае транзакция не сохраняет журнал, который обычно используется для отмены изменений в случае отката. Если транзакция в конце концов будет откатываться , другие транзакции соберут мусор (в конечном итоге) . Эта опция может быть полезна для массивных вставок, которые не нужно откатывать. Для транзакций, которые не выполняют никаких изменений, автоматическая отмена не имеет никакого значения

lock_timeout Этот параметр принимает неотрицательное целое число в качестве значения, задавая максимальное количество секунд, которое транзакция должна ждать при возникновении конфликта блокировки. Если время ожидания истекло, а блокировка еще не была снята, генерируется исключение.

Для обычной транзакции чтения/записи рекомендуются следующие параметры:

read_committed
rec_version
nowait

5.2.6 Редактор транзакций

Редактор транзакций можно открыть во время проектирования двойным кликом по компоненте TIBTransaction .

Рисунок 3: Редактор транзакций

Этот позволяет установить параметры транзакции с помощью выбора одного из вариантов в списке слева. В случае выбора "Read Committed" свойство Params получит список значений из окошка Settings.

5.2.7 Завершение транзакции

Транзакция может быть завершена неявно, путем закрытия по умолчанию (подтвердить или откатить) или явным образом. Транзакция завершается неявно в случае:

- Когда закрывается соединение с базой
- По истечении таймаута (idle timer)
- В случае, если закрывается набор данных, который неявно стартовал транзакцию и нет никаких других открытых наборов, связанных с транзакцией.

Для явного завершения транзакции используются методы Commit или Rollback, которые могут быть вызваны в любой момент, если транзакция активна.

При завершении транзакции связанные с ней наборы данных неявно закрываются, а сделанные в них изменения - отбрасываются

Замечание: важно учитывать, как наборы данных реагируют на закрытие транзакций . См. 6.6.5 Автоматическая фиксация (post) .

5.2.8 Сохранение состояния транзакции после закрытия

При явном завершении транзакций можно сохранить транзакцию активной с помощью методов CommitRetaining или Rol backRetaining.

В обоих случаях транзакция стартует сразу после завершения. В результате нет причин для

неявного закрытия связанных с ней наборов данных. Они остаются открытыми и нет необходимости в обновлении данных.

5.3 TIBEvent

Этот невизуальный компонент является оберткой для интерфейса IEvents. Он служит для:

- Предоставления компонента, который обеспечивает обработку событий
- Предоставляет способ указания на этапе проектирования списка событий, которых должно ожидать приложение.

5.3.1 Ключевые свойства

Database — Ссылка на компонент TIBDatabase для событий которого будут создаваться обработчики.

Events — Список строк, содержащий имена текущего множества событий, на которые хочет реагировать приложение.

Registered — При установке в true, обработчик становится активным и ожидает возникновения событий. При установке на этапе проектирования обработчик также становится активным, если соответствующая база данных подключена к серверу.

5.3.2 События

OnEventAlert Этот обработчик вызывается всякий раз, когда от базы данных приходит событие, указанное в спске Events.

5.3.3 Использование событий

Событие создается процессором базы данных, если в хранимой процедуре или триггере выполняется оператор POST_EVENT. Оператор POST_EVENT назначает при этом событию имя. Например:

```
POST_EVENT 'MYEVENT';
```

Как только завершается транзакция, в рамках которой было создано событие, оно посылается всем подключенным клиентам, у которых зарегистрировано получение таких событий. Такие клиенты могут реагировать на событие, например, путем актуализации своих наборов данных. Это особенно полезно в многопользовательских базах данных, поскольку предоставляет механизм, благодаря которому можно оповещать о сделанных изменениях других пользователей.

Имена событий, которые будет ожидать приложение обычно задаются на этапе проектирования. Чтобы начать получать события, надо установить свойство Registered в true. После этого в случае возникновения события будет вызван обработчик OnEventAlert. В нем приложение должно решить как реагировать на событие.

Замечание: вызов обработчика события делается асинхронно и может произойти в любой момент. Однако обработка происходит в главной ветке приложения и синхронизация веток не требуется. Обработчик событий вызывается для каждого именованного события один раз и получает сведения о количестве таких событий. Эту информацию можно использовать, чтобы исключить дублированные события.

5.4 TIBSQL

Этот невизуальный компонент является оберткой интерфейса IStatement. Его используют

потомки компонента TDataset для выполнения операторов SQL и для доступа к результатам выборки. Его можно использовать в приложении непосредственно для:

- Предоставления компонента, который представляет оператор SQL,
- Как средство для указания оператора SQL во время разработки
- Позволяет во время проектирования устанавливать связь между оператором SQL, используемой транзакцией и соединением с базой данных.

IBX2 поддерживает встроенный SQL посредством интерфейса TIBDatabase.Attachment (см. 5.1.7 Использование интерфейса Attachment) Тем не менее можно также указывать инструкцию во время разработки с помощью TIBSQL .

5.4.1 Ключевые свойства

Database Ссылка на компонент TIBDatabase к которому относится SQL оператор.

GenerateParamNames Если равно true, то позиционные параметры заменяются на имена парметров в формате "IBXParam*nnn*" , где *nnn* представляебой номер позиции параметра, отсчитываемый с нуля. Большинство пользователей могут спокойно игнорировать этот параметр.

UniqueParamNames Игнорируется IBX2. Уникальность имен параметров определяется автоматически.

GoToFirstRecordOnExecute Если установлен в true и оператор SQL является оператором выборки, то после выполнения курсор устанавливается на первую запись набора данных.

ParamCheck По умолчанию равен true. Оператор SQL анализируется на наличие параметров. Если известно, что в операторе их нет, то это свойство можно установить в false пропустив такой анализ.

SQL Список строк, составляющих оператор SQL.

Transaction Ссылка на транзакцию, используемую при выполнении оператора SQLt. Транзакция должна быть активной на момент выполнения запроса, иначе произойдет исключение.

5.4.2 Использование TIBSQL

TIBSQL можно использовать для создания во время проектирования оператора SQL и для связывания его с транзакцией и с базой данных. Однако инструкцию SQL по-прежнему необходимо выполнять и использовать программно. TIBSQL нельзя использовать как источник данных для виджетов.

5.4.2.1 Выполнение хранимой процедуры

Пусть компонент TIBSQL (с именем, например, IBSQL1) содержит оператор SQL «Execute Procedure MyProc :Param1;». Тогда его можно выполнить с помощью:

```
with IBSQL1 do
begin
   Transaction.Active := true; {проверяенм, что транзакция активна}
   ParamByName('MyProc').AsInteger := 1; {присваиваем параметру значениет}
   ExecQuery;
end;
```

В приведенном выше примере запрос сначала подготавливается, затем параметру присваивается значение, затем выполняется запрос.

Отметим, что доступ к параметру всегда можно делать по его номеру, используя свойство Params, даже если параметр является именованным. Если несколько параметров имеют одинаковые имена, то значение присваивается всем параметрам с одним именем.

Имена параметров выделяются и применяются с помощью пакета *fbinf* (более подробно см. раздел 6.1. Firebird Pascal API).

5.4.2.2 Хранимые процедуры возвращающие данные

Допустим, что хранимая процедура имеет вид:

```
Create Procedure MyProc(aParameter Integer)
Returns (SomeText VarChar(64))
As …
```

Тогда после выполнения должен быть возвращен результат. В приведенном выше примере его можно получить используя метод FieldByName. Также его можно получить с использованием свойства Fields. Например:

```
with IBSQL1 do
begin
   Transaction.Active := true;
   ParamByName('MyProc').AsInteger := 1; {считаем, что параметр целого типа}
   ExecQuery;
   writeln('Some Text = ',FieldByName('SomeText').AsString);
end;
```

Обычно имена полей такие же, как имена столбцов или их алиасы в операторе SQL. Однако, бывают и исключения. Для более подробной информации см. раздел 6.1.2 Firebird Pascal API Guide.

5.4.2.3 Выполнение оператора выборки

Если оператором TIBSQL является оператор Select, то результат может содержать множество строк. Он также может иметь входные параметры, как и другие операторы SQL. В результате выполнения SQL создается однонаправленный курсор к результирующему набору данных. К полям результирующего набора можно получить доступ при помощи метода FieldByName. Доступ к полю также можно получить по его номеру, с использованием свойства Fields. Вы можете переходить к следующей записи результирующего набора при помощи метода Next. Когда будут прочитаны все записи свойство EOF будет установлено в true. Для того, чтобы освободить память занимаемую результирующим набором данных, запрос должен быть явно закрыт.

Замечание: при первом открытии курсор устанавливается на первую запись результирующего набора, за исключением случая, когда свойство GoToFirstRecordOnExecute установлено в false. В последнем случае для доступа к записи должен быть вызван метод Next.

Например, если использовать базу данных "employee" и оператор SQL имеет вид:

```
Select EMP_NO,FULL_NAME From EMPLOYEES Order by 2;
```

то следующий код выполнит запрос и выведет результат по одной строке.

```
with IBSQL1 do
begin
    Transaction.Active := true;
    ExecQuery;
    while not EOF do
    begin
        write('EMP_NO = ',Fields[0].AsString);
        writeln('Full Name = ',Fields[1].AsString);
        Next;
    end;
    Close;
end;
```

6 Компоненты набора данных DataSet

Модель набора данных, введенная в Delphi и повторно реализованная на Free Pascal является общей моделью доступа к наборам данных и используется во многих приложениях Lazarus.

Виджеты доступа к данным можно разместить на форме Lazarus и связать с набором данных посредством TDataSource. В большинстве случаев они также привязываются с помощью имени поля к одному полю в наборе данных. Фокус ввода (события от клавиатуры) может получить только одна из записей набора данных. Изменения в данных этой записи могут делаться с помощью одного или нескольких виджетов доступа к данным. Изменения сохраняются в базе данных в результате «фиксации» ("posting")(например, путем вызова метода TDataset.Post). Для отмены сделанных изменений можно использовать метод Cancel.

TDataset является абстрактным классом, который предоставляет общего предка для многих наборов данных различного типа. IBX предоставляет множество потомков TDataset для осуществления эффективного доступа с помощью SQL к базе данных Firebird. В дополнение к заданию общего представления набора данных, класс TDataset содержит также общие методы для поиска конкретных записей в наборе, и перемещения вперед/назад по записям набора данных (First, Last, Next, Prior и MoveBy).

IBX определяет внутренний класс **TIBCustomDataset** , который занимается деталями предоставления доступа к наборам данных Firebird. Экземпляры этого класса нельзя создавать непосредственно, в отличие от его многочисленных потомков, находящихся на закладке Firebird палитры компонентов.

6.1 Набор данных IBX

IBX предоставляет следующие наборы данных:

- TIBTable
- TIBStoredProc
- TIBQuery
- TIBDataset

Набор данных состоит из одной или более строк данных, организованных в столбцы. Поле данных находится на пересечении строки и столбца и задается именем столбца. Его значение берется из текущей строки. Для изменения набора данных служит компонент TIBUpdateSQL, который позволяет добавлять или перекрывать операторы SQL, используемые для вставки, изменения, удаления или актуализации данных.

6.2 Общие понятия

6.2.1 Общие свойства

Все потомки IBX TDataset поддерживают следующие свойства:

Active Во время выполнения установка в true открывает набор данных, а установка в false закрывает его. Если оно установлено в true во время проектирования, набор данных открывается сразу после загрузки компонента.

AllowAutoActivateTransaction Если это свойство установлено в true, то открытие набора

данных (установка его свойства в active в значение true) автоматически запускает транзакцию.

AutoCalcFields Если равно true, то вычисляемые поля автоматически пересчитываются каждый раз, когда изменяются данные в строке. Это является действием по умолчанию для TDataset.

AutoCommit Может быть установлено в "disabled" (по умолчанию) или в "CommitRetaining". В последнем случае, если строка удаляется или фиксируется, то транзакция завершается по "CommitRetaining" , сохраняя данные в базу данных и оставляя транзакцию активной, а набор данных открытым.

BufferChunks **Важный параметр производительности**: Этот параметр определяет на сколько будет увеличиваться внутренний буфер базы, если прежний размер окажется исчерпанным. Значение по умолчанию равно 1000 строк.
IBX, как правило, кэширует данные во внутреннем буфере. Если известно, что набор данных состоит только из нескольких строк, то имеет смысл установить BufferChunks в небольшое значение (например, 10 , если количество строк обычно меньше 10) сэкономив таким образом память. Напротив, если количество строк очень большое (например, 100,000) , то установка BufferChunks в большое значение (например, 25000) предотвратит очень частое изменение размера буферного пула при чтении набора данных. Однако, к этому вопросу следует подходить осторожно, чтобы исключить создание большого количества неиспользуемых буферов. В некоторых случаях имеет смысл определить реальные потребности, узнав во время выполнения количество записей в наборе, и на основании полученных данных, установить размер BufferChunks непосредственно перед открытием набора.

CachedUpdates Если установлено в true, то изменения хранятся у клиента, вместо того, чтобы записываться в базу данных. Изменения передаются в базу только при явном вызове метода ApplyUpdates. Для отката изменений можно вызвать метод CancelUpdates, который возвращает базу к состоянию на момент последнего вызова ApplyUpdates. Это дает возможность сохранить/откатить изменения вообще без записи данных в базу.

Database Ссылка на компонент TIBDatabase управляющий соединением с базой к которой относится набор данных.

DatasetCloseAction Может быть установлен в "DiscardChanges" (по умолчанию) или "SaveChanges". Определяет, что делать с записью, которая изменена, но не зафиксирована, в случае закрытия набора данных. В случае "SaveChanges" текущая запись фиксируется перед закрытием набора данных, в противном случае изменения отбрасываются.

ReadOnly Установите в true, чтобы запретить изменять набор данных.

Transaction Ссылка на транзакцию, которая управляет чтением набора данных и применением изменений.

Unidirectional Если равно true, то по набору данных можно перемещаться только вперед. В этом случае нет необходимости создавать и поддерживать большой внутренний буферный пул (см. также описанный ранее BufferChunks).

6.2.2 Общие события

Обработчики событий могут быть заданы для задания действий как при подготовке к операции с набором данных, так и сразу по окончании операции. Например, перед открытием/закрытием набора и после открытия/закрытия, перед вставкой/редактированием/удалением записи и после этих операций.

BeforeOpen Это один из самых важных обработчиков. Вызывается после того, как свойство Active набора данных устанавливается в true, но набор данных еще не открыт.

Если запрос на выборку данных имеет параметры, то именно здесь удобно задать им значения. Здесь также можно открыть наборы данных, которые нужны открываемому.

AfterOpen — Этот обработчик можно использовать для выбора начальной записи в наборе данных. Здесь также удобно открывать наборы данных, которые зависят от отрываемого, например, находятся в отношении главная-подчиненная.
Если наборы данных связаны в иерархическую структуру и обработчики событий правильно настроены, то только набор данных верхнего уровня необходимо явно активировать. Остальные будут открываться последовательно, в порядке, определяемом обработчиками событий.

BeforeClose — Обычно используется, чтобы закрыть наборы данных, зависящие от данного, например в отношении главная-подчиненная.

AfterClose — Обычно закрывает те наборы данных, которые становятся ненужными после закрытия данного набора.

BeforeEdit — Если запись не должна редактироваться, вызывайте здесь исключение.

AfterEdit — Этот обработчик можно использовать для включения индикации, что строка находится в состоянии редактирования.

BeforeInsert — Здесь можно защитить набор данных от вставки записей в данный момент, вызывая исключение.

AfterInsert — Хорошее место для присвоения начальных значений для вставленной записи.

BeforeDelete — Здесь можно защитить набор данных от удаления записей, вызывая исключение.

AfterDelete — Можно использовать для включения индикации о том, что имеются отложенные изменения (если отключен AutoCommit).

BeforeCancel — Вызывайте здесь исключение, если сделанные изменения нельзя отменять.

AfterCancel — Здесь можно сбросить индикацию состояния редактирования записи.

BeforePost — Вызывайте здесь исключение, если сделанные изменения не должны сохраняться (напрмер, не прошли проверку на корректность данных). Можно использовать и для присвоения значений полям, для которых не созданы виджеты.

AfterPost — Может использоваться для включения индикации о том, что имеются еще не подтвержденные изменения (если отключен AutoCommit).

BeforeTransactionEnd — Вызывайте здесь исключение, если в данный момент нельзя завершать транзакцию.

AfterTransactionEnd — Может быть использован для сбрасывания индикации об отложенных изменениях

OnCalcFields — Набор данных может иметь кроме полей, извлекаемых из базы данных, дополнительные поля, значение которых вычисляется на основании данных текущей записи. Этот обработчик вызывается каждый раз при перемещении по набору данных, когда новая запись становится текущей. Здесь следует обновлять значения вычисляемых полей (и ни в коем случае не изменять значений обычных полей, иначе возникнет бесконечный цикл /примечание переводчика/).

BeforeScroll — Вызывается перед переходом к другой записи. Вызывайте исключение, если в вашей ситуации смена записи нежелательна.

AfterScroll — Вызывается после перехода к другой записи. Может использоваться для обновления данных в элементах управления, которые не являются виджетами данных.

6.2.3 Обработка исключений

Наборы данных IBX предоставляют несколько обработчиков событий, связанных с ошибками базы данных, включая OnDeleteError, OnEditError, OnPostError. Их можно использовать для обработки возникающих исключений.

Однако, следует отметить, что все исключения базы данных являются потомками EIBInterBaseError. Возможно, что более правильным будет использовать единый обработчик путем вызова TApplication.AddOnExceptionHandler и в нем перехватывать и обрабатывать EIBInterBaseError .

6.2.4 Наборы символов и кодовые страницы

Текстовые поля Firebird всегда относятся к одному из предопределенных наборов символов (например, UTF8).Начиная с FPC 3.0.0, тип AnsiString, содержит атрибут кодовой страницы, которая определяет набор символов, используемый строкой. IBX спроектирован так, чтобы набор символов поля данных и кодовая страница AnsiString были совместимы, а при необходимости производилась транслитерация при передаче данных между клиентом и базой данных. Более подробно можно прочитать в разделе 9 руководства Firebird Pascal API.

6.3 TIBTable

TIBTable предоставляет простое средство для доступа и изменения таблиц Firebird без необходимости изучать SQL. Необходимо только указать имя таблицы и IBX сделает все остальное. Наборы данных TIBTable можно связать отношением главная/подчиненная. См. Также пример в "ibx/examples/ibtable".

6.3.1 Ключевые свойства

FieldDefs — Позволяет редактировать и, следовательно, изменять определение поля таблицы базы данных. Можно использовать для ограничения списка полей или для добавления вычисляемых полей. Используется вместе со свойством "StoreDefs".

IndexDefs — Позволяет редактировать и, следовательно, изменять список индексов набора данных. Используется вместе со свойством "StoreDefs".

IndexFieldNames — Используется подчиненной таблицей для задания отношения главная/подчиненная. Указывает список полей (разделенных точкой с запятой) используемых в отношении главная/подчиненная и, соответственно, определяет порядок сортировки подчиненной таблицы.

IndexName — Указывает индекс, по которому сортируются записи таблицы. Это свойство нельзя использовать в подчиненной таблице.

MasterFields — При использовании отношения главная/подчиненная задает список полей главной таблицы, которые соответствуют IndexFieldNames.

MasterSource — Используется подчиненной таблицей для задания источника данных (TDataSource) главной таблицы.

StoreDefs — Если установлено в true, то описания полей и определения индексов сохраняются в файле формы.

TableName — Имя таблицы базы данных, которая является источником записей для набора данных.

TableTypes — Используется для управления списком таблиц, возвращаемым сервером во время проектирования. По умолчанию содержит только пользовательские

таблицы. При выборе соответствующей опции можно получить список системных таблиц или просмотров базы данных

6.3.2 Использование TIBTable

TIBTable, вероятно, самый простой в использовании набор данных IBX, хотя и с ограниченными возможностями. Для использования достаточно поместить компонент на форму, связать его с помощью инспектора объектов с компонентом TIBDatabase, а затем выбрать нужную таблицу из выпадающего списка в свойстве TableName. При установке свойства Active будет автоматически сгенерирован SQL, который считает содержимое таблицы в набор данных.

Порядок записей можно изменить путем выбора (существующего) индекса, или задавая в IndexFileNames список полей для сортировки.

Если не установлено в true свойство ReadOnly, то полученный набор данных можно изменять. Необходимые для редактирования операторы SQL будут созданы автоматически. Если свойства AllowAutoActivateTransaction и AutoCommit установлены в true, то управление транзакциями будет полностью автоматическим и все изменения будут автоматически сохранятся в базе. В этом случае следует также установить DatasetCloseAction в SaveChanges. В комплект примеров входит программа, работающая таким образом, с установленным в true AllowAutoActivateTransaction для главной таблицы.

6.3.2.1 Главная/подчиненная таблицы

Главным ограничением TIBTable является то, что он может отображать содержимое только одной таблицы. Получившийся набор данных должен один к одному соответствовать таблице базы. Однако, даже с таким ограничением его все же можно использовать в отношении главная/подчиненная.

Отношение главная/подчиненная возникает, если есть общие для двух таблиц поля, которые позволяют связать две таблицы так, что одной записи главной таблицы соответствует множество записей подчиненной таблицы. Например, одна таблица содержит список отделов, а другая — список служащих, с указанием отдела, где они работают. Если сделать таблицу с отделами главной, а со служащими подчиненной, то каждой строке в таблице отделов будет соответствовать список сотрудников, работающих в этом отделе, в таблице служащих

Чтобы связать две таблицы отношением главная/подчиненная:

1. Создайте два компонента TIBTable и подключите их к одной базе данных. Задайте для каждой таблицы их имена.
2. Поместите на форму компонент TDataSource и свяжите его с главной таблицей.
3. Установите свойство MasterSource подчиненной таблицы в значение компонента из пункта 2.
4. Откройте редактор свойств для MasterFields кликнув на кнопочку в конце его названия (см. Рисунок 4).

Рисунок 4: Редактор свойства MasterFields

5. "Detail Index Fields" представляет список доступных комбинаций индексных полей для подчиненной таблицы (как они определены в базе данных). Надо выбрать подходящий индекс для связи таблиц.
6. Теперь выберите имена полей главной таблицы в списке Master Fields и нажмите на кнопку "Join". Связанные поля будут отображены в нижнем окне.
7. Нажмите OK для завершения связи.

Во время выполнения вы должны сначала открыть главную таблицу, а потом подчиненную. Для этого открытие подчиненной таблицы можно сделать в обработчике события AfterOpen главной таблицы. Аналогично, закрытие подчиненной таблицы можно поместить в обработчик BeforeClose главной таблицы. Как только таблицы будут открыты, выбор строки в главной таблице приведет к тому, что в подчиненной будут отобраны записи, связанные с главной.

Отметим, что в примере программы, приводимом в ibx/examples/ibtable, главная таблица открывается в обработчике AfterConnected компонента TIBDatabase , а подчиненная в обработчике AfterOpen главной таблицы.

6.4 TIBStoredProc

Компонент TIBStoredProc используется для выполнения хранимой процедуры (на сервере базы данных), в том числе таких, которые не создают наборы данных . Он генерирует необходимый для выполнения процедуры SQL , избавляя программиста от написания собственного кода. Продвинутые пользователи, возможно, предпочтут использовать встроенный SQL (см. 5.1.7 Использование интерфейса Attachment).

6.4.1 Ключевые свойства

StoredProcName Имя хранимой процедуры в базе данных Firebird.

Params Список входных и выходных параметров, которые содержит процедура. Создается базой данных.

6.4.2 Использование TIBStoredProc

На этапе проектирования:

1. Поместите на форму компонент TIBStoredProc и выберите базу данных.
2. Установите в инспекторе объектов свойство имени StoredProcName, выбрав его из списка.

На этапе выполнения каждый раз при вызове процедуры вы можете задавать требуемые параметры, используя свойство Params или метод ParamByName, а затем запускать процедуру,

используя метод ExecProc.

Например:

```
with IBStoredProc1 do
begin
   Transaction.Active := true;
   ParamByName('EMP_NO').AsInteger := 8;
   ExecProc;
end;
```

В приведенном выше примере предполагается, что процедура имеет один целый параметр по имени EMP_NO. Как только задано его значение, процедуру можно выполнять. Заметим, что имена параметров берутся из определения процедуры в базе данных, например

```
Create Procedure MyProc(EMP_NO Integer)
```

Хранимая процедура также может возвращать значения в виде выходных параметров. Доступ к ним можно получить с помощью свойства Params с именем требуемого параметра (опять берущегося из определения процедуры в базе данных).

6.5 TIBQuery

Этот компонент является потомком TDataset и создает набор данных в результате выполнения SQL запроса (или оператора Select хранимой процедуры, возвращающей набор данных). Запрос SQL используемый компонентом, задается в его свойстве SQL. Это гораздо более мощное средство, чем TIBTable, поскольку создает набор данных с помощью произвольного оператора SQL, включающего столько таблиц, сколько необходимо. Однако, для его использования необходимо знание SQL.

Создаваемый набор данных имеет доступ «только для чтения», за исключением случая, когда использует компонент TBUpdateSQL.

6.5.1 Ключевые свойства

DataSource	Эта ссылка (может быть и пустой) позволяет указать другой набор данных, который может использоваться для получения значений параметров запроса. Может рассматриваться как более общая версия свойства MasterSource в TIBTable.
ForcedRefresh	В случае редактируемых запросов вынуждает автоматически актуализировать каждую запись после изменения. Полезно для обновления вычисляемых полей и получения данных, которые могут быть результатом выполнения триггеров.
GeneratorField	В случае редактируемых запросов задает автоматическую установку для поля с помощью генератора Firebird при добавлении записи к набору данных. См. раздел 6.6.3 Генераторы.
Params	Список параметров. Извлекается из запроса с параметрами.
SQL	Текстовое представление SQL запроса (в виде списка строк TStrings).
UpdateObject	Ссылка на необязательный компонент TIBUpdateObject (см. 6.6 TIBUpdateSQL)

6.5.2 Использование TIBQuery

Во время проектирования:

1. Поместите компонент TIBQuery на форму и установите значение свойства Database в соответствии с нужной базой данных.
2. Создайте запрос SQL с помощью редактора свойства SQL, кликнув на многоточие или кликнув правой кнопкой по компоненте и выбрав из контекстного меню «SQL Editor»(см. 6.5.3 Редактор свойства SQL).

Если SQL не содержит параметров, то все, что нужно сделать на этапе выполнения — это установить значение свойства Active в true. Запрос будет выполнен и результат станет доступен в виде набора данных

- Если база данных еще не подключена, когда запрос делается активным, то она неявно подключается.
- Если при открытии запроса транзакция еще не запущена, то возникает исключение, кроме случая, когда свойство AllowAutoActivateTransaction тоже равно true. Обработка запросов с параметрами обсуждается ниже в разделе 6.5.4.

Замечание: в операторе выборки результат запроса часто состоит из единственной записи. Использовать форму для просмотра/редактирования единственной записи является значительно более эффективным, чем считывание набора данных целиком. В разделе 12.1 приводится пример использования многострочных наборов, когда они используются в качестве источника данных для компонента TIBDynamicGrid.

6.5.3 Редактор свойства SQL

Редактор свойства SQL предоставляет средства для ввода и редактирования запроса SQL во время проектирования. . Он позволяет также протестировать запрос на корректность синтаксиса. Также можно сгенерировать стандартный запрос на основании информации, содержащейся в базе данных.

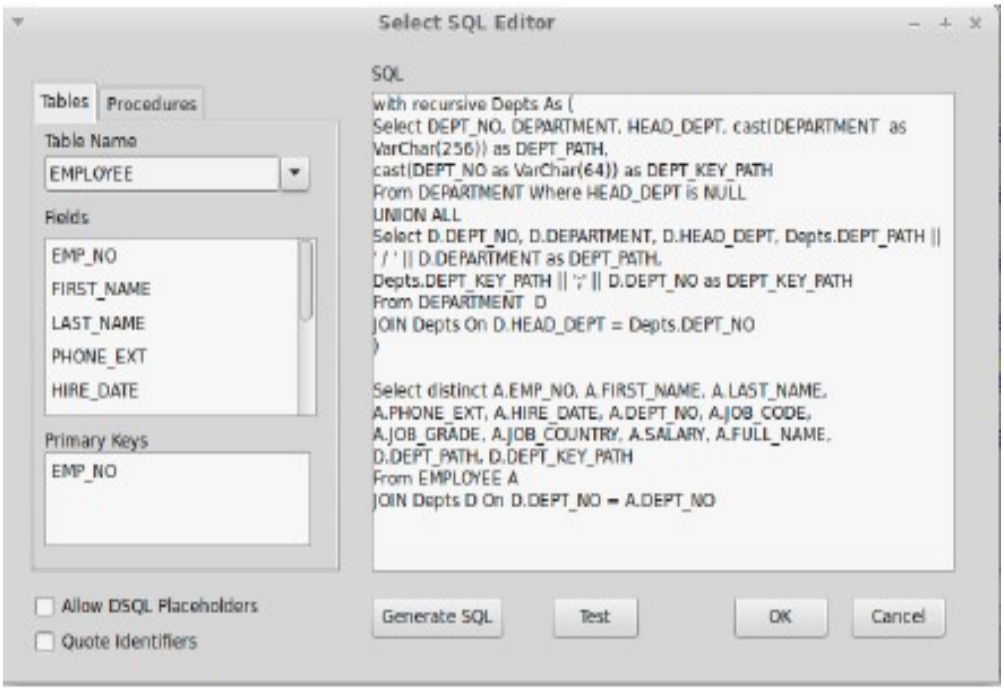


Рисунок 5: Редактор свойства SQL

Для нормальной работы редактора у компоненты TIBQuery должно быть установлено свойство TIBDatabase, указывающее на существующую базу данных. Должна быть указана транзакция по умолчанию и связана с TIBDatabase, а свойство Database должно указывать на компонент TIBDatabase у всех компонентов доступа к данным.

Редактор свойства SQL показан на рисунке 5. Этот редактор является частным случаем IBX SQL редакторов, которые имеются для Update, Insert и Delete SQL.

Все редакторы SQL пакета IBX используют одну основную схему с выпадающим списком в левой стороне окна редактора, содержащим доступные таблицы и простыми списками для полей и ключей для выбранной таблицы. Их можно использовать как для справочных целей, так и как источник данных для Generate SQL.

- Кнопка "Generate SQL" позволяет создать для выбранной таблицы SQL операторы для выборки, которые показаны в правом окне редактора. Если отмечены какие-то столбцы, то редактор ограничивает запросы этими столбцами. В противном случае в запросах участвуют все столбцы таблицы.
  Отметим, что оператор Update следует отредактировать, чтобы исключить изменение главного ключа таблицы. Если это внутренний ключ, невидимый пользователю, то мало смысла включать его в операции Update.
    - Двойной клик на имени столбца приводит к тому, что имя столбца включается в текст оператора SQL.
    - Кнопку "Test" можно использовать для проверки корректности текущего оператора SQL. При обнаружении ошибки она будет показана пользователю с указанием позиции символа, который вызвал ошибку.

Обычно функция "Generate SQL" используется для получения начального шаблона текста операторов SQL, который затем редактируют, чтобы они соответствовали задаче. Кнопка "Test" может использоваться для проверки правильности синтаксиса, чтобы для этого не приходилось компилировать и запускать программу. При включенном "Quote Identifiers" имена полей и таблиц будут взяты в двойные кавычки. Включение флага "Allow DSQL Placeholders" позволит использовать знак "?" для задания параметров запроса (см. ниже).

Оператор выборки SQL, обычно создается на основе таблиц и просмотров. Однако, хранимые процедуры, возвращающие наборы данных, также могут служить источником данных.

Для этого служит закладка со списком хранимых процедур.

6.5.4 Запросы с параметрами

Под такими запросами понимаются операторы SQL , которые содержат поименованные поля подстановки. Синтаксис операторов SQL поддерживаемый IBX такой же, что и синтаксис Dynamic SQL поддерживаемый Firebird. Поддерживаются как язык обработки данных (DML), так и язык определения данных (DDL). Однако имеется одно существенное различие в процессе обработки запросов с параметрами.

В обычном Firebird Dynamic SQL, параметры представляются с помощью знака '?' и используются как позиционные параметры. IBX разрешает использовать такой подход, но дополнительно позволяет использовать именованные параметры, синтаксис которых взят из языка для хранимых процедур и триггеров. В этом случае имена параметров предваряются двоеточием, а в остальном соответствуют правилам для именования столбцов базы данных. Например, оператор:

```
Select A.EMP_NO, A.FIRST_NAME, A.LAST_NAME
 From EMPLOYEE A
Where A.EMP_NO = :EMP_NO;
```

представляет собой оператор выборки с параметром, где параметром является ":EMP_NO" . Такое расширение является существенным для компонент изменения данных (см. 6.6 TIBUpdateSQL). Реализация применения запросов с параметрами содержится в пакете *fbintf*. См. раздел 6.1.1 Firebird Pascal API Guide.

Для оператора выборки значения параметрам должны быть присвоены до выполнения запроса. Например:

```
IBQuery1.ParamByName('EMP_NO').AsInteger := 1;
IBQuery1.Active := true;
```

Если компонент TIBQuery включает запрос с параметрами, которым на момент выполнения запроса не присвоены значения, и при этом его свойство DataSource указывает на активный набор данных, то в качестве значений параметров будут использованы значения из набора данных Datasource с тем же именем поля, что и имя параметра. Это правило позволяет реализовать отношение главная/подчиненная с помощью подходящего выбора имени параметров.

На практике установку значений параметров, которые не присваиваются автоматически через Datasource, удобно делать в обработчике BeforeOpen компонента TIBQuery Это гарантирует, что параметры получат значения до открытия набора данных.

6.6 TIBUpdateSQL

Сам по себе TIBQuery предоставляет набор данных «только для чтения». Если данные нуждаются в редактировании, то необходимо дополнительно использовать компонент TIBUpdateSQL и назначить его в качестве значения свойства UpdateObject у компонента TIBQuery.

Компонент TIBUpdateSQL предоставляет операторы SQL для изменения (Modify), вставки (Insert), удаления (Delete) и актуализации (Refresh), чтобы сделать набор данных редактируемым.

6.6.1 Ключевые сойства

RefreshSQL Список строк, представляющий собой оператор SQL на актуализацию одной строки набора данных.

ModifySQL Список строк, представляющий собой оператор SQL по изменению текущей строки набора данных.

InsertSQL Список строк, представляющий собой оператор SQL для вставки одной строки на основании значений текущей строки набора данных.

DeleteSQL Список строк, представляющий собой оператор SQL для удаления строки, соответствующей текущей строке набора данных.

Для создания и редактирования описанных выше операторов, предоставляется редактор соответствующих свойств, аналогичный редактору свойства SQL компонента TIBQuery. Компонент TIBUpdateSQL предоставляет редактор для доступа ко всем четырем операторам в одном диалоге.

6.6.2 Синтаксис SQL для операторов изменения данных

Все SQL операторы, используемые UpdateObject являются запросами с параметрами:

- Оператор SQL для актуализации данных очень похож на оператор компонента TIBQuery' с тем отличием, что использует раздел "where", для ограничения данных единственной записью, соответствующей текущей строке (используя главный ключ в качестве критерия отбора). Например,

```
Select A.EMP_NO, A.FIRST_NAME, A.LAST_NAME, A.PHONE_EXT, A.HIRE_DATE,
A.DEPT_NO, A.JOB_CODE, A.JOB_GRADE, A.JOB_COUNTRY, A.SALARY,
A.FULL_NAME From EMPLOYEE A
Where A.EMP_NO = :EMP_NO
```

- Запрос SQL на изменение данных является оператором UPDATE, который изменяет одну запись набора данных, соответствующую текущей записи (обратите внимание на использование конструкции:OLD_ для последнего параметра, см. 6.6.2.1 Параметры OLD(старый) и NEW(новый) , например:

```
Update EMPLOYEE A Set
A.EMP_NO = :EMP_NO,
A.FIRST_NAME = :FIRST_NAME,
A.LAST_NAME = :LAST_NAME,
A.PHONE_EXT = :PHONE_EXT,
A.HIRE_DATE = :HIRE_DATE,
A.DEPT_NO = :DEPT_NO,
A.JOB_CODE = :JOB_CODE,
A.JOB_GRADE = :JOB_GRADE,
A.JOB_COUNTRY = :JOB_COUNTRY,
A.SALARY = :SALARY
Where A.EMP_NO = :OLD_EMP_NO
```

- Оператор SQL для вставки записи, является оператором INSERT для вставки в таблицу одной записи, соответствующей текущей записи набора данных, например:

```
Insert Into EMPLOYEE(EMP_NO, FIRST_NAME, LAST_NAME, PHONE_EXT,
HIRE_DATE, DEPT_NO, JOB_CODE, JOB_GRADE, JOB_COUNTRY, SALARY)
Values(:EMP_NO, :FIRST_NAME, :LAST_NAME, :PHONE_EXT, :HIRE_DATE,
:DEPT_NO, :JOB_CODE, :JOB_GRADE, :JOB_COUNTRY, :SALARY)
```

- Оператор SQL для удаления данных, является оператором DELETE для удаления одной строки, соответствующей текущей, например:

```
Delete From EMPLOYEE A Where A.EMP_NO = :EMP_NO
```

Во всех приведенных примерах имена параметров следуют правилу, что имена параметров должны соответствовать имени поля данных набора данных. Это соглашение используется для того, чтобы при выполнении запроса значение параметра извлекалось из поля данных с тем же именем.

- В запросе актуализации обязательно должен присутствовать раздел "where", использующий одно или несколько полей соответствующих единственной строке. Обычно эти поля составляют главный ключ соответствующей таблицы. Если запрос возвращает несколько строк, то будет использована только первая.
- В запросе на изменение каждое поле таблицы изменяется в соответствии со значением в текущей строке. Отметим конструкцию "OLD_" использованную в разделе "where". Подробно она обсуждается в разделе 6.6.2.1. В данном случае она должна извлечь из базы данных значение, как при выполнении оператора refresh (актуализации).
- В запросе на вставку записи каждое поле таблицы изменяется в соответствии со значением такого же поля текущей записи.
- В запросе на удаление в разделе "where" выбирается удаляемая строка таблицы со значением ключа, как при выполнении оператора refresh (актуализации).

6.6.2.1 Параметры OLD(старый) и NEW(новый

Они обычно используются в запросах на изменение данных. Например, если изменяется значение главного ключа. В этом случае строка, подлежащая изменению, должна быть выбрана в соответствии со старым значением ключа, тогда как значения полей должны быть установлены в их новое значение. Для того, чтобы обеспечить такую возможность, IBX позволяет использовать префиксы "OLD_" для формирования ссылок на значения, которые были получены при считывании из таблицы (до вызова TDataset.Edit), и "NEW_" , которые являются значениями по умолчанию и ссылаются на уже измененные данные, после обращения к TDataset.Edit. Например, задавая оператор в виде:

```
UPDATE MYTABLE
Set Key1 = :NEW_KEY1, COL2 = :COL2 Where Key1 = :OLD_KEY1;
```

для правильной обработки с помощью:

```
IBDataset1.Next;
IBDataset1.Edit;
IBDataset1.FieldByName('key1').AsInteger := <новое значение>;
IBDataset1.Post;
```

Отметим, что NEW_ используется по умолчанию и его использование обычно диктуется желанием сделать текст более понятным.

Префикс "OLD_" может также использоваться для установления значений в поля, которые должны сохранить прежние значения (например, в другом поле). Например:

```
Update EMPLOYEE A Set
A.EMP_NO = :EMP_NO,
A.LAST_NAME = :LAST_NAME,
A.PREVIOUS_NAME = :OLD_LAST_NAME,
Where A.EMP_NO = :OLD_EMP_NO
```

Префикс "OLD_" нельзя использовать в запросах на актуализацию (Refresh), поскольку они могут выполняться только, когда набор данных не находится в состоянии редактирования. Запросы на удаление обычно используют текущие, а не прежние значения.

6.6.3 Генераторы

Генераторы Firebird (известные также как последовательности Sequence) используются для генерации последовательных номеров, не зависящих от транзакции. Типичным применением является создание главных ключей для таблиц, которые гарантировано не будут конфликтовать у разных пользователей, добавляющих записи в одну и ту же таблицу. Генераторы часто требуются при добавлении новой записи для присвоения полю, входящему в состав ключа, последовательных чисел. IBX поддерживает такой механизм для редактируемых наборов данных при помощи свойства GeneratorField.

Для указания на этапе проектирования генератора и соответствующего поля откройте редактор свойства GeneratorField нажав на многоточие в описании свойства в инспекторе объектов. Должен открыться редактор свойства.

Замечание: это свойство принадлежит компоненту TIBQuery,UpdateObject.

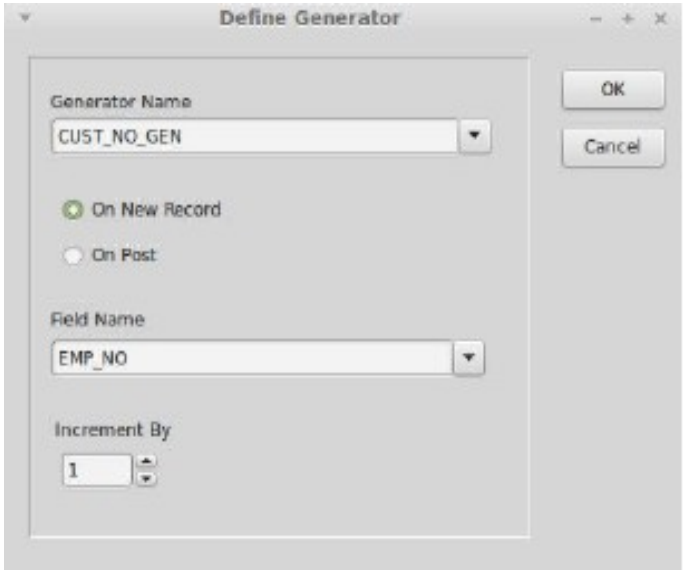


Рисунок 6: Редактор свойства GeneratorField

Здесь вы можете выбрать нужный генератор из числа определенных в базе данных и поле таблицы, которому будет присваиваться значения генератора. Присвоение можно делать в обработчиках событий либо при вставке записи или при ее фиксации (posted). В большинстве случаев предпочтительно событие "On New Record". Событие "On Post" может быть полезно для вставки способом "точно в срок" , когда хочется исключить неиспользуемые номера.

6.6.4 Изменение наборов данных

Применяются следующие правила:

1. В данный момент времени может редактироваться только одна запись.
2. Прежде, чем будет начато редактирование, набор данных должен быть переведен в состояние редактирования с помощью метода Edit.
   Заметим, что виджеты управления данными обычно вызывают "Edit" автоматически,

как только данные изменяются.

3. Новую строку можно добавить, используя методы Append, AppendRecord, Insert или InsertRecord. В этом случае набор данных переводится в состояние "вставки записи" (insert).
4. Изменения фиксируются в базе с помощью метода Post. Его можно использовать, если набор данных находится в состоянии "edit" или "insert". По завершении операции набор данных переходит в состояние «просмотра» (browse).
5. Изменения могут быть отброшены с помощью вызова метода Cancel.
   Для наборов данных, использующих вычисляемые поля, полезно включить свойство ForceRefresh, которое гарантирует, что отображаемые в наборе данные соответствуют хранящимся в базе данных.

6.6.5 Автоматическая фиксация (post)

Изменяемые наборы данных IBX автоматически фиксируют изменения при переходе к другой записи. Действия при закрытии набора данных от значения свойства DataSetCloseAction:

- Если оно установлено в «Discard changes» (по умолчанию), то происходит «отмена» и изменения теряются.
- Если оно равно «Save changes», то запись фиксируется (post) перед закрытием набора данных. Исключение составляет случай, если набор данных закрывается в связи с откатом транзакции. В этом случае изменения всегда отбрасываются.

6.6.6 Событие OnValidatePost

Его объявление имеет вид

```
TOnValidatePost = procedure (Sender: TObject; var CancelPost: boolean) of object;
```

Это событие доступно для всех редактируемых наборов данных и вызывается первым при вызове метода Post. Если оно возвращает "CancelPost" равное true, то вызывается метод "Cancel" и выполнение "Post" прекращается. Таким образом обработчик события имеет возможность решить, что фиксирование данных должно быть отменено, путем проверки действительных значений полей данных, или запросить подтверждение у пользователя.

Если говорить о последовательности вызовов, то событие возникает до события OnBeforePost . Таким образом, если происходит переход на другую запись в момент, когда запись находится в состоянии редактирования, то события происходят в следующем порядке:

- OnValidatePost (возвращает CancelPost = false)
- OnBeforePost
- OnAfterPost
- OnBeforeScrol
- OnAfterScrol

или

- OnValidatePost (возвращает CancelPost = true)
- OnBeforeCancel
- OnAfterCancel
- OnBeforeScrol

- OnAfterScrol

Заметим, что попытка вызвать Cancel в обработчике OnBeforePost не будет работать, поскольку все еще выполняется Post и в IBX будет вызвано исключение.

Сигнализировать об ошибке в данных можно вызывая исключения в обработчике OnValidatePost или OnBeforePost.

6.6.7 Кэширование изменений

Если свойство CachedUpdates у редактируемого набора данных установлено в true, то при фиксации данных изменения кэшируются, а не передаются базе данных. Эти изменения будут переданы базе, только при вызове метода ApplyUpdates. Этот метод очищает кэш и записывает изменения в базу данных. В случае вызова CancelUpdates кэш также очищается, но изменения отбрасываются без сохранения в базе данных; восстанавливаются исходные значения записей.

Отметим, что если DatasetCloseAction установлено в Save Changes, то ApplyUpdates вызывается автоматически.

Кэширование изменений часто очень полезно при редактировании многострочных наборов данных. В этом случае изменения сохраняются в базе только если форма закрывается по ModalResult=mrOK. Если же она закрывается с mrCancel, то можно вызвать CancelUpdates и изменения не будут переданы в базу данных.

Такого же эффекта можно добиться при использовании транзакций, но это будет менее эффективно.

6.7 TIBDataSet

Компонент TIBDataset представляет собой TIBQuery и TIBUpdateSQL свернутые в один компонент. Он формирует редактируемый набор данных в виде одного объекта. Для него применимо все вышесказанное.

6.7.1 Ключевые свойства

SelectSQL — Список строк определяющий оператор SQL для создания набора данных.

RefreshSQL — Список строк определяющий оператор SQL для актуализации одной строки набора данных.

ModifySQL — Список строк определяющий оператор SQL для изменения одной строки таблицы на основании текущей записи набора данных.

InsertSQL — Список строк определяющий оператор SQL для вставки одной строки таблицы на основании текущей записи набора данных.

DeleteSQL — Список строк определяющий оператор SQL для удаления одной строки таблицы, соответствующей текущей записи набора данных.

Для TIBDataset также имеется редактор компонента. Его можно вызвать из контекстного меню для компонента. Этот редактор позволяет создать все операторы SQL в один клик.

7 Компоненты поддержки IBX

7.1 Процессор скрипов IBX

Компонент TIBXScript позволяет выполнить скрипт SQL взятый из файла или из потока. Текст скрипта разбивается на отдельные операторы SQL, которые выполняются по очереди. Цель состоит в том, чтобы получить совместимость с языком утилиты командной строки ISQL , но с некоторыми расширениями:

- Поддерживаются все операторы DML и DDL.
- Поддерживаются CREATE DATABASE, DROP DATABASE, CONNECT и COMMIT.
- Поддерживаются следующие операторы SET:

SET SQL DIALECT
SET TERM
SET AUTODDL
SET BAIL
SET ECHO
SET COUNT
SET STATS
SET NAMES <character set>

Новая команда: RECONNECT. Делает подтверждение транзакций, а затем отключается и снова подключается к базе данных.

- Процедурные блоки (блоки BEGIN .. END) не нуждаются в специальном терминаторе. Если терминатор все-таки присутствует, то он обрабатывается как пустой оператор. Результат является совместимым с ISQL, но не требует использования SET TERM.
- Операторы DML могут использовать аргументы в IBX формате (например, UPDATE MYTABLE Set data = :mydata). Аргументы допустимы только для BLOB полей и получают значения с помощью обработчика события GetParamValue. В результате возвращается blobid для использования в дальнейшем. Типичное использование состоит в том, чтобы считать данные из двоичного файла, сохранить их в двоичный поток и вернуть blobid.
- Поддерживается простой формат XML для BLOB данных (см. 7.6.1 Извлечение двоичных данных Blobs), массивов полей (см. 7.6.2 Извлечение массивов данных) а также экспорт с помощью компонента TIBExtract (см. 7.6 Компонент TIBExtract).
- Строки комментариев в стиле C++ .

Операторы SQL выборки непосредственно не поддерживаются, но могут обрабатываться с помощью внешнего обработчика (в событии OnSelectSQL). Если обработчик не задан, то при обнаружении оператора SELECT вызывается исключение.

7.1.1 Свойства:

Database	Ссылка на компонент TIBDatabase
Transaction	Ссылка на компонент TIBTransaction. По умолчанию устанавливается на внутреннюю транзакцию (concurrency, wait)
AutoDDL	Если равно true, то операторы определения данных коммитятся сразу

после выполнения

Echo Если равно true, то все операторы SQL записываются в журнал

StopOnFirstError Если равно true, то процессор скриптов прекращает работу после первой ошибки SQL.

IgnoreGrants Если равно true, то операторы управления правами доступа тихо (silently) игнорируются. Это полезно в случае выполнения скриптов с использованием встроенного сервера.

ShowAffectedRows Если равно true, то после выполнения операторов DML количество измененных строк записывается в журнал.

ShowPerformanceStats Если равно true, то после выполнения операторов DML в журнал сохраняется статистика производительности (в формате ISQL).

DataOutputFormatter Указывает компонент, используемый для форматирования результатов выполнения операторов Select.

7.1.2 События:

GetParamValue вызывается при обнаружении параметра SQL (в формате PSQL :name). Используется только для BLOB полей. Обработчик должен вернуть BlobID для использования в качестве значения параметра. Если обработчик не задан, то при обнаружении параметра вызывается исключение.
Напоминание: используйте TIBBlobStream для создания и чтения BLOB данных из файла

OnOutputLog Вызывается при записи операторов SQL в журнал (stdout)

OnErrorLog Вызывается при записи других сообщений в журнал (stderr)

OnProgressEvent Служит для создания индикатора выполнения. Если параметр Reset равен true, то параметр value равен максимальному значению индикатора. Иначе вызывается метод step индикатора.

OnSelectSQL Обработчик для оператора SELECT. Если не задан, то для обработки используется DataOutputFormatter. Если ни OnSelectSQL, ни DataOutputFormatter не заданы, то вызывается исключение.
Обработчик OnSelectSQL может либо сам обработать оператор выборки, либо вызвать TIBXScript.DefaultSelectSQLHandler, чтобы инициировать обработку по умолчанию, как это описано выше.

OnSetStatement Обработчик для неопознанных операторов SET.

7.1.3 Использование

Для выполнения операторов SQL или скриптов можно использовать следующие функции компонента TIBXScript:

```
function RunScript(SQLFile: string): boolean; overload;
function RunScript(SQLStream: TStream): boolean; overload;
function RunScript(SQLLines: TStrings): boolean; overload;
function ExecSQLScript(sql: string): boolean;
```

SQL скрипт может быть передан как файл, поток (stream), список строк (TStrings) или в виде одной строки (string). Приведенные выше функции, отличаются только способом передачи скрипта. В остальном они тождественны. Скрипт разбивается на операторы и выполняется по одному оператору в порядке, задаваемом скриптом. Функция возвращает true, если операторы выполнены успешно, и false в случае неудачи.

7.1.4 Примеры

В каталоге"ibx/examples" приводится два примера программ, которые иллюстрируют использование компонента TIBXScript , как в графическом (GUI) режиме, так и в консольном. Примеры находятся в каталогах:

1. ibx/examples/scriptengine
2. ibx/examples/fbsql

7.1.4.1 Пример scriptengine

Здесь демонстрируется применение компонента TIBXScript. Он работает с использованием базы данных EMPLOEE и поставляется с несколькими тестовыми скриптами для демонстрации их работы. Скрипты находятся в каталоге "tests".

Откомпилируйте и запустите программу, если уверены, что база данных EMPLOEE доступна на локальном сервере. Если она находится на удаленном сервере, то вы должны соответственно изменить свойство IBDatabase1.DatabaseName.

Вы можете просто набрать текст SQL запроса в левом текстовом поле и нажать "Execute" для выполнения. Результат появится в правом текстовом поле. Запросы выполняются с помощью нового, динамически создаваемого окна, с сеткой данных, содержащей результат запроса. Это окно не является модальным, поэтому можно одновременно наблюдать результаты нескольких запросов. Сетка является компонентом TIBDynamicGrid и нажатием на заголовок столбцов можно упорядочивать данные в столбце.

Тестовые скрипты можно загрузить в левое текстовое поле при помощи кнопки "Load Script". Доступны следующие скрипты:

1. **CreateCountriesTable.sql**
   Добавляет новую таблицу "COUNTRIES" в базу данных employee и заполняет ее странами, включая название страны, ее ISO2 и 3 символа доменного имени. По окончании выполнения скрипта показывается содержимое новой таблицы.
2. **CreateCountriesTablewithError.sql**
   Делает то же, что в предыдущем примере, за исключением того, что содержит синтаксическую ошибку в первом операторе insert. Это можно использовать для экспериментов с флагом "Stop on First Error" , демонстрируя как скриптовый процессор может восстанавливаться после ошибок и продолжать выполнение.
3. **DeptListView.sql**
   Этот скрипт добавляет в базу данных сложный просмотр и тестирует скриптовый процессор в сложных сценариях, включая рекурсивные запросы.
4. **createproc.sql**
   Этот скрипт добавляет три простые хранимые процедуры. Он демонстрирует три различных способа для записи тела процедуры (совместимо с ISQL , с помощью стандартных терминаторов и без терминаторов). Демонстрируется также использование комментариев.
5. **ParameterisedQueries.sql**
   Этот скрипт иллюстрирует использование параметров в стиле PSQL для BLOB столбцов. В этом случае к таблице COUNTRY добавляется новый столбец "Image", и изображение в формате png (флаг святого Георгия) добавляется к записи об Англии. Значение столбца Image задается параметром ":MyImage". Приложение присваивает ему значение, запрашивая имя файла, в котором находится изображение. Вы должны найти и вернуть в качестве результат файл "flag_en.png".
   Отметим, что интерактивное присвоение значения параметру делается для примера.

Обычно назначение происходит в обработчике события, который может, к примеру, искать файл с именем "MyImage.bin" , соответствующему имени параметра.

6. Reverseall.sql
Откатывает назад все., что было сделано выше

7. SelectQuery.sql
Иллюстрирует выполнение запросов выборки данных.

7.1.5 Консольное приложение fbsql

fbsql является не только примером приложения, он является заменой консольной программе ISQL как для интерактивного так и для пакетного режима использования. fbsql использует компонент TIBXScript в качестве процессора скриптов и TIBExtract (см. 7.6 Компонент TIBExtract) для извлечения метаданных из базы данных. Результаты выборки выводятся на *stdout* в формате CSV , удобном для загрузки данных в электронные таблицы, в виде оператора insert, или в блоковом формате. Он также включает интерактивную версию TIBXScript.

Использование: fbsql <параметры> <имя базы данных>

Параметры:

-a записать метаданные в stdout
-A записать метаданные и содержимое таблиц в stdout
-b остановиться при первой ошибке
-e копировать операторы SQL в stdout
-i <filename> выполнить скрипт SQL , содержащийся в файле
-h показать этот список (help)
-o <filename> выводить вместо stdout в указанный файл
-p <password> задать пароль в командной строке (небезопасно)
-r <rolename> открыть базу данных, используя роль
-s <sql> выполнить SQL текст
-t задать формат вывода для операторов SQL
BLK (default) для блокового формата
CSV (default) для CSV формата
INS (default) для оператора Insert
-u <username> открыть базу данных от имени этого пользователя (по умолчанию SYSDBA)

Переменные среды (операционной системы):

ISC_USER имя пользователя по умолчанию для доступа к Firebird
ISC_PASSWORD пароль доступа к Firebird по умолчанию

Сохранение имени пользователя и пароля в качестве переменных среды избавляет от необходимости вводить их в командной строке и является более безопасным способом ввода пароля

Если пароль не задан ни в командной строке ни в переменных окружения, то пароль у пользователя будет запрошен дополнительно.

Если в командной строке не указаны ни "-s" ни "-i" параметры, то fbsql выполняется интерактивно.

fbsql использует IBX в консольном режиме. Перед открытием проекта вы должны информировать среду разработки Lazarus о пакете ibexpressconsolemode. Все, что вам нужно — это в меню Lazarus выбрать "Packages->Open Package File" и открыть файл ibexpressconsolemode.lpk, который вы можете найти в корневом каталоге ibx. Сразу после этого вы

можете его закрыть. Не нужно его ни компилировать, ни устанавливать. Открыть пакет достаточно, чтобы Lazarus его запомнил.

Поддерживаемые SQL операторы

- Все DML и DDL операторы.
- Операторы CREATE DATABASE, DROP DATABASE, CONNECT и COMMIT. Дополнительно введен оператор RECONNECT, который сбрасывает соединение и устанавливает его вновь.

Поддерживаемые команды ISQL

- SET SQL DIALECT
- SET TERM
- SET AUTODDL
- SET BAIL
- SET ECHO
- SET COUNT
- SET STATS
- SET NAMES <character set>
- SET HEADING
- SET ROWCOUNT
- SET PLAN
- SET PLAN ONLY
- QUIT
- EXIT

Чтобы использовать программу, откомпилируйте ее с помощью Lazarus IDE и запускайте ее из командной строки. Параметры команды приведены выше по тексту. Например:

```
fbsql -a -u SYSDBA -p masterkey employee
```

выведет в *stdout* метаданные для локальной базы EMPLOEE (предполагается пароль по умолчанию).

```
fbsql -A -u SYSDBA -p masterkey -o employeedump.sql employee
```

скопирует всю базу EMPLOEE, включая данные, в текстовый файл (employeedump.sql).

```
fbsql -u SYSDBA -p masterkey -i employeedump.sql
```

восстанавливает базу, скопированную в файл "employeedump.sql". Отметим, что оператор "CREATE DATABASE" в начале файла, следует изменить так, чтобы указать нужное имя для создаваемого файла.

Напротив,

```
fbsql -u SYSDBA -p masterkey -i employeedump.sql new-employee.fdb
```

восстановит базу данных в файл 'new-employee.fdb' в случае, если он уже создан в виде

пустой базы данных. Отметим, что в этом случае оператор "CREATE DATABASE" должен быть закомментирован.

fbsql -s "Select * From EMPLOYEE" -u SYSDBA -p masterkey employee

запишет содержимое таблицы EMPLOYEE из локальной базы данных в stdout (предполагается пароль по умолчанию).

fbsql -b -e ../scriptengine/tests/CreateCountriesTable.sql -u SYSDBA -p masterkey employee

выполнит для локальной базы EMPLOEE скрипт CreateCountriesTable.sql из набора тестовых скриптов. Каждый выполняемый оператор будет выведен в stdout, а в случае ошибок выполнение будет остановлено после первой ошибки.

Заметим, что в Linux, для запуска программы из командной строки, если она не прописана в PATH, вам нужно использовать синтаксис "./fbsql" , например

./fbsql -a -u SYSDBA -p masterkey employee

7.2 Компоненты форматирования вывода

Это вспомогательные компоненты, предназначенные главным образом, для использования совместно с TIBXScript, но их также можно использовать с TIBExtact (для подготовки данных для оператора Insert). Их назначением является форматирование результата вывода при выполнении оператора SELECT. В настоящее время доступны три компонента форматирования вывода, которые можно использовать для:

- Вывода в блоковом формате (TIBBlockFormatOut)
- Вывода в формате CSV (Comma Separated Values) (TIBCSVDataOut)
- Вывода в форме подходящей для использования в операторе Insert (TIBInsertStmtsOut).

7.2.1 Использование

Для использования вместе с IBXScript: поместите соответствующий компонент на форму и подключите его к свойству DataOutputFormatter компонента TIBXScript.

Компоненты форматирования вывода можно использовать и непосредственно. Список свойств приводится ниже. Все компоненты содержат следующие методы:

- procedure Assign(Source: TPersistent); override;
- procedure DataOut(SelectQuery: string; Add2Log: TAdd2Log);
- procedure SetCommand(command, aValue, stmt: string; var Done: boolean); virtual;
- class procedure ShowPerfStats(Statement: IStatement; Add2Log: TAdd2Log);

Assign: используется для копирования свойств одного компонента в другой.

DataOut: выполняет заданный запрос. Он форматирует результат вывода в виде одной или более строк и возвращает каждую строку при помощи прилагаемого обработчика событий "Add2Log".

SetCommand: используется компонентом TIBXScript для расширенной обработки команды SET с помощью компонент форматирования вывода. Используется для команд SET (HEADING | ROWCOUNT | PLAN | PLANONY).

ShowPerfStats: используется для форматирования вывода статистики производительности, создаваемой IStatement, аналогично выводу ISQL.

7.2.2 Свойства

Database	Ссылка на компонент TIBDatabase
Transaction	Ссылка на компонент TIBTransaction
PlanOptions	Определяет, какие составные части плана запроса будут выведены при форматировании результатов запроса: только план, план и результаты, только результаты.
RowCount	Если не равно нулю, то ограничивает количество строк вывода.
ShowPerformanceStats	Если равно true, то вместе с результатами запроса приводится совместимая с ISQL статистика производительности.
IncludeHeader	Если равно true, то в результаты включается заголовок данных (только для форматов CSV и блокового).
QuoteChar	Символ, используемый в качестве разделителя для вывода в формате CSV (по умолчанию одинарные кавычки).
IncludeBlobsAndArrays	Если равно true, оператор insert включает BLOB поля и массивы в виде данных в формате XML (см. 7.6.1 Извлечение двоичных данных Blobs и 7.6.2 Извлечение массивов данных)

7.3 SQL парсер

В IBX 1.2 появился класс TSelectSQLParser (расположен в модуле IBSQLParser). Этот класс проводит синтаксический разбор и изменение операторов выборки данных Firebird. Он предназначен для синтаксического разбора всех таких операторов, включая UNION и обобщенных табличных выражений (Common Table Expressions).

Замечание: основное его назначение состоит в том, чтобы дать удобный способ по изменению секций "Where", "Having" и "Order by" по частям, а не для контроля правильности SQL. Хотя в случае неправильного SQL часто вызывается исключение, это не гарантируется.

7.3.1 Парсер

Парсер может использоваться как самостоятельный класс, но обычно его используют как свойство "Parser" компонента TIBDataSet или TIBQuery, в их обработчике события "BeforeOpen". Обращение к свойству Parser вызывает создание объекта TSelectSQLParser , который может использоваться при открытии набора данных.

Пример использования приводится в каталоге ibx/examples/employee, где он используется для фильтрации таблицы набора данных EMPLOYEE по изменяемым условиям. В этом примере обработчик события BeforeOpen имеет вид

```
procedure TForm1.EmployeesBeforeOpen(DataSet: TDataSet);
begin
  if BeforeDate.Date > 0 then
    (DataSet as TIBParserDataSet).Parser.Add2WhereClause('HIRE_DATE < :BeforeDate');
  if AfterDate.Date > 0 then
    (DataSet as TIBParserDataSet).Parser.Add2WhereClause('HIRE_DATE > :AfterDate');
  case SalaryRange.ItemIndex of
  1:
```

```
      (DataSet as TIBParserDataSet).Parser.Add2WhereClause('Salary < 40000');
    2:
      (DataSet as TIBParserDataSet).Parser.Add2WhereClause('Salary >= 40000
        and Salary < 100000');
    3:
      (DataSet as TIBParserDataSet).Parser.Add2WhereClause('Salary >= 100000');
    end;
  {Значения параметров должны устанавливаться после того, как все изменения в SQL
  закончены !!!}
    if BeforeDate.Date > 0 then
      (DataSet as TIBParserDataSet).ParamByName('BeforeDate').AsDateTime
        := BeforeDate.Date;
    if AfterDate.Date > 0 then
      (DataSet as TIBParserDataSet).ParamByName('AfterDate').AsDateTime := AfterDate.Date;
  end;
```

В этом примере можно использовать два фильтра:

- Ограничение по "Hire Date" для выбора интервала даты приема на работу
- Ограничения по окладу, выбираются с помощью выпадающего списка, содержащего границы окладов.

В каждом случае фильтры нужно добавить в секцию "Where".

Когда к объекту Parser обращаются в первый раз, он создается с использованием первоначального текста SQL, созданного на этапе проектирования. Вызов метода "Add2WhereClause" делается в предположении, что задаваемые условия будут добавлены к прежним в виде AND. Для задания добавления в виде условия OR служит второй необязательный параметр Add2WhereClause (в примере отсутствует).

Если пользователь в приводимом выше примере выбирает заданный фильтр, то оператор SQL соответствующим образом изменяется. Add2WhereClause может вызываться несколько раз и каждый раз он добавляет условие к прежнему тексту секции "Where" . Скобки добавляются автоматически так, чтобы сохранить смысл первичного выражения.

В приводимом примере необходимое условие на дату приему на работу можно сразу отформатировать в виде текста, например:

```
HIRE_DATE < '2015-01-01'
```

Однако, обычно более удобно использовать вместо этого условие с параметром, значение которому присваивается позже.

Заметим, что пример демонстрирует важную роль обработчика события BeforeOpen: значения параметров следует устанавливать только после того, как все изменения в SQL закончены. Это происходит из-за того, что перед присваиванием значений параметров запрос должен быть подготовлен (prepared), а изменение SQL всегда переводит запрос в состояние "unprepared" и все значения параметров сбрасываются.

При использовании приводимого примера, когда пользователь изменяет фильтр, то набор данных закрывается, и переоткрывается, вызывая регенерацию SQL и набор данных открывается с уже примененным фильтром.

Свойства SelectSQL компонента TIBDataSet и свойство SQL компонента TIBQuery также можно изменять во время выполнения. Как уже указано, это приводит к закрытию набора данных и сбрасыванию значений параметров. Кроме того, изменяется начальный текст SQL,

используемый парсером в качестве исходного.

7.3.2 Использование вместе с IBControls

TSelectSQLParser используется и с другими элементами управления IBControls (см. 12 Виджеты управления данными от IBX). Эти элементы управления также имеют свойство Parser и могут обращаться к нему в обработчике BeforeOpen.

7.3.3 Пример

Пример непосредственного использования TSelectSQLParser можно найти в каталоге ibx/examples/sqlparser. Пример состоит из простой формы, которую можно использовать для экспериментов с парсером и видеть, как изменяются операторы SQL при вызовах Add2WhereClause.

В соответствии с рисунком 7, вы можете использовать этот пример для тестирования парсера с помощью:

- вставки текста SQL запроса в текстовое поле "Original SQL"
- записи SQL условия в одно или несколько текстовых полей ниже на форме
- установки нужных ключей,
- нажатием кнопки "Generate Updated SQL".

Модифицированный оператор SQL должен появиться в правом текстовом поле.

Пример, выбранный здесь, является довольно тривиальным, взятым из программы ibx/examples/employee и показывает одно предложение фильтра, добавляемое к SQL, используемому для создания списка сотрудников .

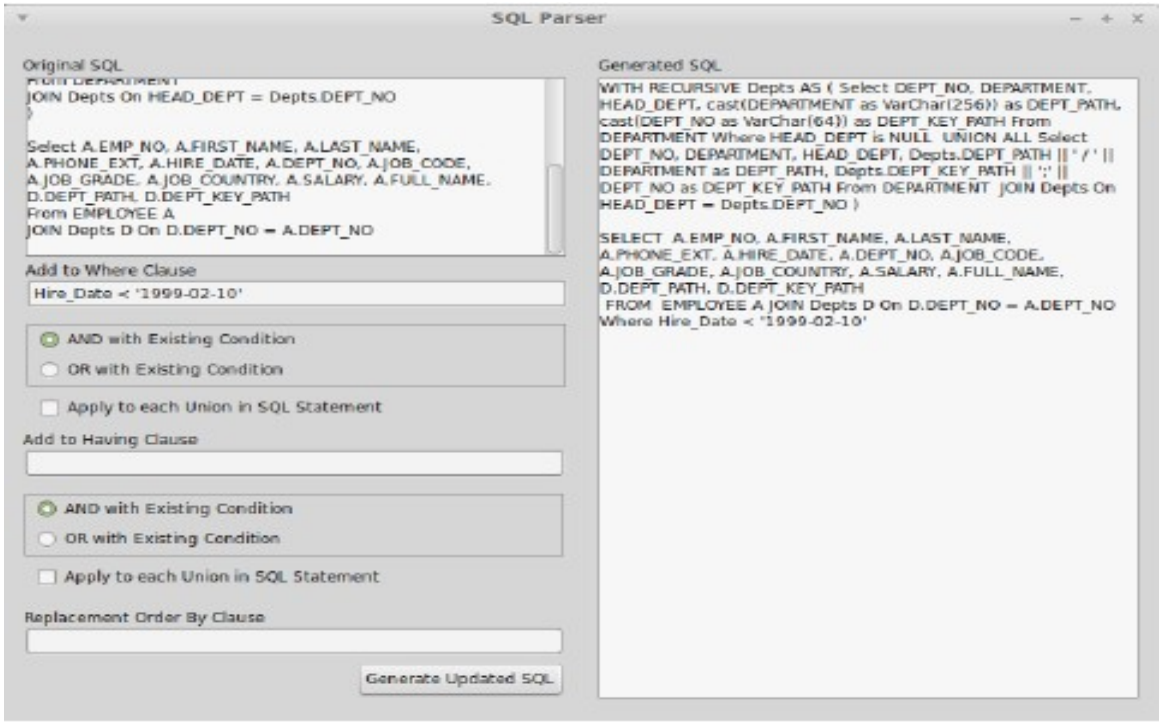


Рисунок 7: Пример использования SQL Parser

7.3.4 Обзор класса TSelectSQLParser

Для изучения полного списка свойств и методов смотрите исходный код. Приводимые ниже

свойства и методы предназначены для использования в обработчике события BeforeOpen:

```
procedure Add2WhereClause(const Condition: string; OrClause: boolean=false;
IncludeUnions: boolean = false);
```

Этот метод используется для добавления строки Condition к секции "Where". Если такой секции нет в оригинальном запросе, то она будет добавлена. По умолчанию условие добавляется в качестве AND к текущему условию "Where" . Если аргумент "OrClause" равен true, то используется OR. Если используется секция UNION, то по умолчанию условие добавляется только в первый оператор SELECT, но если аргумент "IncludeUnions" равен true, то условие добавляется ко всем операторам в UNION.

```
procedure Add2HavingClause(const Condition: string; OrClause: boolean=false;
IncludeUnions: boolean = false);
```

Этот метод делает то же, что и Add2WhereClause, за исключением того, что изменения применяются к секции "Having" оператора SELECT.

```
property Union: TselectSQLParser;
```

Если оператор выборки является объединением нескольких операторов, то доступ ко второму оператору SELECT можно получить с помощью свойства "Union". Каждый следующий оператор SELECT рекурсивно добавляется к предыдущему с помощью этого свойства.

```
property OrderByClause: string;
```

С помощью этого свойства можно читать и изменять содержимое секции "Order By" . Имеется в виду текст самого условия, а не ключевые слова "Order by".

```
property SQLText: string
```

Это свойство возвращает текущее значение оператора SQL, с учетом всех изменений. Его полезно использовать для отладки.

7.4 Монитор ISQL

Компонент TIBISQLMonitor служит для отладки и позволяет вам видеть, какие операторы SQL выполняются при помощи IBX. Кроме того, его можно применять для идентификации узких мест и проблем производительности.

Приложение IBX может мониторировать свое выполнение, а если имеет достаточно прав, то и другое приложение IBX,

7.4.1 Компонент TIBISQLMonitor

Достаточно поместить компонент на форму и присвоить его свойству Enabled значение true, чтобы начать мониторирование. Компонент не нуждается в подключении к другим компонентам IBX. Компонент TIBISQLMonitor может также выступать как источник или сливатель??? информации или и то и другое при трассировке выполнения SQL.

7.4.1.1 Выбор объектов мониторирования

Свойство TraceFlags компонента TIBDatabase определяет какие операции SQL будут

передаваться для объекта TIBSQLMonitor. Это свойство может принимать значения tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt, tfConnect, tfTransact, tfBlob, tfService, tfMisc

7.4.1.2 Отчеты SQL

Для формирования отчетов используется обработчик события OnSQL компонента TIBSQLMonitor. В нем можно формировать текстовые сообщения о происходящих событиях. Задайте подходящий объект для записи сообщений. Их можно записать в stdout, в файл журнала или добавлять к текстовому полю TMemo для просмотра отчета на экране. Компонент TIBSQLMonitorтакже имеет свойство TraceFlags, определяющее какие операции приведут к вызову обработчика OnSQL.

7.4.1.3 Мониторирование приложения

При включении TIBSQLMonitor его вывод становится доступен для других приложений, работающих на том же компьютере. В случае производных Unix , доступ ограничен приложениями, работающими под тем же пользователем.

Для мониторирования приложению требуется только компонент TIBSQLMonitor. Никакие другие компоненты IBX включать в приложение необязательно. Другими словами, TraceFlags и обработчик событий OnSQL используются одинаково в мониторинге любых приложений .

Замечание: если работает несколько IBX приложений, то будет идти мониторирование событий от всех приложений и отфильтровать их не представляется возможным.

7.4.2 Примеры

Здесь предлагается два примера приложений, демонстрирующих возможности TIBISQLMonitor. Они находятся в каталоге "ibx/examples/isqlmonitor"

7.4.2.1 Комплексное мониторирование

Этот пример делает небольшие изменения в примере Employee и добавляет вторую форму "Monitor Form" для записи выбранного множества событий в текстовое поле TMemo. Список мониторируемых событий можно изменить с помощью свойства TraceFlags компонента TIBDatabase.

7.4.2.2 Удаленное мониторирование

В этом примере иллюстрирует использование TIBISQLMonitor для мониторирования другого приложения. Это приложение включает одну форму, содержащую текстовое поле TMemo, используемое для ведения журнала событий SQL . Запустите его вместе с IntegratedMonitoring и вы увидите, что события поступают в оба приложения . Заметим, что список мониторируемых событий определяется в приложении IntegratedMonitoring, которое задает их с помощью свойства TraceFlags компонента TIBDatabase и должно вызвать "EnableMonitoring", без которого никакое мониторирование не начнется.

7.5 Компонент TIBDatabaseInfo

Firebird предоставляет доступ к свойствам базы данных и ее статистике посредством включенного в IBX компонента TIBDatabaseInfo.

Для использования компонента поместите его на форму и свяжите его с базой данных, о которой нужно получить информацию. Вы можете сделать несколько компонент TIBDatabaseInfo, связанных с одной базой данных.

При выполнении программы свойства компонента будут отражать состояние базы данных и доступную информацию о статистике работы. Ниже приводится список этих свойств:

DBFileName	Имя файла, где хранится база данных
DBSiteName	Имя сайта базы данных
Allocation	Количество страниц, отведенных для базы данных
BaseLevel	Номер версии базы данных
DBImplementationNo	Database Implementation Number???
NoReserve	Отводится ли в базе место для хранения резервных копий записей
ODSMinorVersion	Младший номер версии ODS
ODSMajorVersion	Номер версии ODS
PageSize	Размер страницы в байтах
Version	Номер версии для реализации базы данных.
Reads	Количество считанных страниц базы
CurrentMemory	Размер памяти, используемой сервером (в байтах) в настоящее время
ForcedWrites	Число указывающее режим, в котором сервер производит запись в базу данных (0 для асинхронного режима, 1 для синхронного)
MaxMemory	Максимальное количество памяти (в байтах), которое было использовано, начиная с первого подключения к базе данных
NumBuffers	Количество буферов памяти, используемых в данный момент
SweepInterval	???Number of transactions that are committed between "sweeps" to remove database record versions that are no longer needed
UserNames	Список подключенных пользователей
Fetches	Количество считываний из внутренней памяти
Marks	Количество записей во внутреннюю память
Reads	Количество считанных страниц
Writes	Количество записанных страниц
BackoutCount	Количество удалений версий записи
DeleteCount	Количество удалений с момента подключения базы данных
ExpungeCount	Количество удалений записи и всех ее предков для записей, удаление которых было подтверждено
InsertCount	Количество удалений с момента подключения базы данных
PurgeCount	Количество удалений старых версий полностью подтвержденных записей (записей, которые зафиксированы, так что более старые версии предков больше не нужны)
ReadIdxCount	Количество чтений индексов с момента подключения базы данных
ReadSeqCount	Количество последовательных чтений таблиц с момента подключения базы данных
UpdateCount	Количество сделанных изменений с момента подключения базы
DBSQLDialect	SQL диалект базы данных

7.6 Компонент TIBExtract

Этот компонент позволяет извлекать из базы данных метаданные. Его назначение состоит в том, чтобы достичь совместимости с расширениями языка определения данных (DDL) вплоть до Firebird 3.

В каталоге "ibx/examples/fbsql" приводится пример консольного приложения, использующего этот компонент.

Использование: во время проектирования поместите компонент на форму и свяжите его с нужным компонентом TIBDatabase.

Во время выполнения для извлечения метаданных можно использовать метод ExtractObject, если база данных подключена к серверу.

```
procedure ExtractObject(ObjectType : TExtractObjectTypes; ObjectName : String = '';
ExtractTypes : TExtractTypes = [])
```

По завершении работы метода список строк, содержащих метаданные, доступен в виде свойства Items компонента TIBExtract .

Типы извлекаемых объектов определяют тип извлекаемых метаданных, а Extract Type уточняет , что именно извлекается:

Извлекаемый объект	**Тип объекта**	**Метаданные**
eoDatabase		Полная схема базы данных
	etData	Полная схема базы данных плюс данные в виде оператора INSERT сразу после определения таблицы. Данные включают двоичные поля Blobs (см. 7.6.1 Извлечение двоичных данных Blobs) и массивы данных (см.7.6.2) в текстовом формате
eoDomain	etDomain	Все домены
	etTable	Домены используемые при определении указанных таблиц
eoTable		Указанные таблицы, или все, если таблицы не заданы
	etDomain	Включая домены, используемые таблицами
	etIndex	Включая индексы заданные для таблиц
	etForeign	Включая внешние ключи, заданные для таблиц
	etCheck	Включая ограничения данных, заданные для таблиц
	etTrigger	Включая все триггеры, заданные для таблиц

Извлекаемый объект	Тип объекта	Метаданные
	etGrant	Включая права доступа, заданные для таблиц
	etData	Включая данные в виде оператора INSERT
eoView		Список всех просмотров или только тех, что заданы в ObjectName
	etTrigger	Включая триггеры, заданные для просмотров
	etGrant	Включая права доступа для просмотров и их триггеров
eoProcedure		Список всех процедур или только тех, что заданы в ObjectName
	etGrant	Включая права доступа для процедур
eoFunction		Список всех функций или только тех, что заданы в ObjectName
eoGenerator		Список всех генераторов или только тех, что заданы в ObjectName
	etData	Включая оператор “ALTER SEQUENCE” для установки значений генераторов в текущие значения
eoException		Список всех исключений или только тех, что заданы в ObjectName
eoBLOBFilter		Список всех фильтров BLOB или только тех, что заданы в ObjectName
eoRole		Список всех ролей или только тех, что заданы в ObjectName
eoTrigger		Список всех триггеров или только тех, что заданы в ObjectName
	etTable	Список всех триггеров для ObjectName (имя таблицы)
	etGrant	Включая права доступа для триггеров
eoForeign		Список всех внешних ключей или только тех, что заданы в ObjectName
	etTable	Список внешних ключей для ObjectName (имя таблицы)
eoIndexes		Список всех индексов или только тех, что заданы в ObjectName

Извлекаемый объект	Тип объекта	Метаданные
	etTable	Список всех индексов для ObjectName (имя таблицы)
eoChecks		Список всех условий проверки или только тех, что заданы в ObjectName
	etTable	Список всех условий проверки для ObjectName (имя таблицы)
eoData		Включить все данные в виде оператора INSERT для указанных таблиц

При извлечении базы данных полностью вывод производится в следующем порядке:

- Комментарии к оператору CREATE DATABASE
- Фильтры
- Функции
- Домены
- Таблицы
- Данные (если запрошены)
- Индексы
- Ограничения внешних ключей
- Генераторы, включая установку значений, если требуется
- Просмотры
- Ограничения на значения (Check Constraints)
- Исключения
- Create Procedure Stubs (заглушки с названиями процедур)
- Триггеры
- Операторы Alter Procedure для описания тела процедуры
- права доступа

Такой порядок важен для исключения конфликтов с зависимостями объектов. Данные добавляются сразу после описания таблиц. Это исключает проблемы с внешними ключами и ограничениями на значения.

Заглушки для процедур (объявление заголовка с пустым телом) необходимо создать раньше триггеров, чтобы можно было в триггерах использовать процедуры. Однако полностью определить процедуры нельзя, если не заданы триггеры по которым они вызываются. С другой стороны, если хранимая процедура используется для изменения просмотров, то возникает ошибка. Просмотры доступны для записи только после определения триггеров .

Тела процедур задаются также с учетом зависимостей. Так, чтобы одна процедура могла использовать другую в запросе выборки. Это допустимо только после того, как процедура была определена и включает оператор SUSPEND .

7.6.1 Извлечение двоичных данных Blobs

TIBExtract экспортирует двоичные данные в простом формате XML в виде шестнадцатеричных символов. Например:

```
<blob subtype="0">
89504E470D0A1A0A0000000D49484452000000122000000AE0803000000A565F093000000155
04C5445FFFFFFCE1124CC0005EDB8BBEEBCBFCD0016E1858B135C5E2200000189494441
54789CEDDAB10DC3301004415AA4D47FC97607133E04EF5470D8F8D69AB5AF0F5C7B78
E2B4125189A84454222A1195884A4425A2125189A84454222A1195884A4425A2125189A844
54222A1195884A4425A2125189A84454222A1195884A4425A2125189A84454222A1195884A
4425A2125189A84454222A1195884A4425A2125189A84454222A1195884A4425A2125189A84
454222A1195884A4425A2125189A844749CE84C6FDCA3CE732BD1FD9CD98DEB9AC542BF
46C3139727FEBB125189A84454222A1195884A4425A2125189A84454222A1195884A4425A21
25189A84454222A1195884A4425A2125189A84454222A1195884A445DB0A8231F4D5F2DDF7
0071DD6A9984A4425A2125189A84454222A1195884A4425A2125189A84454222A1195884A44
25A2125189A84454222A1195884A4425A2125189A84454222A1195884A4425A2125189A84454
222A1195884A4425A2125189A84454222A1195884A4425A2125189A84454222A1195884A4425
A2125189A84454222A1195884A4425A21724FA0255D0459DFD53A3DD0000000049454E44AE
426082
</blob>
```

В приведенном выше примере двоичные данные с подтипом '0' форматируются в виде строк из шестнадцатеричных символов.

Данные ограничены тэгом </blob>. Пробелы при считывании игнорируются. В каждой строке всегда должно быть четное количество символов.

Приведенная выше конструкция может использоваться в операторах DML в качестве значения blob. Например:

```
INSERT INTO MyTable (KeyValue,BlobData) Values(1,<blob subtype="0">
89504E470D0A1A0A0000000D49484452000000122000000AE0803000000A565F093000000154
...
</blob>);
```

7.6.2 Извлечение массивов данных

Компонент TIBExtract может также экспортировать массивы данных при помощи простого формата XML . Например:

```
<array dim = "1" sqltype = "448" length = "60" relation_name = "JOB"
 column_name = "LANGUAGE_REQ" charset = "NONE" bounds="1:5">
<elt ix="1">Japanese
</elt>
<elt ix="2">Mandarin
</elt>
<elt ix="3">English
</elt>
<elt ix="4">
</elt>
<elt ix="5">
</elt>
</array>
```

Пример, приведенный выше, является более сложным, чем в случае BLOB и отражает информацию о структуре данных, необходимую для определения массива. Этот пример взят из базы данных employee и является значением поля в столбце LANGUAGE_REQ в таблице JOB.

тэги "array" определяют:

- dim: размерность массива
- sqltype: SQL тип массива данных (???using blr type codes)
- length: размер каждого элемента в байтах.
- relation_name: имя таблицы для которой задан массив.
- column_name: имя столбца в этой таблице.
- charset: имя набора символов (только для текстовых данных)
- bounds: список нижних и верхних границ, разделенных запятыми. По одной паре на каждое измерение. Нижняя и верхняя граница разделены двоеточием (':').

Значения элементов массива вложены с использованием тэга "elt". Тэг элементов первого уровня используется для первого измерения и указывает номер элемента для первого измерения и так далее. Самый внутренний тег "elt" содержит данные.

Например, для двумерных массивов:

```
<array dim="2" ... bounds="1:5,1:2">
<elt ix="5">
  <elt ix="2">English</elt>
</elt>
</array>
```

Этот пример содержит единственный элемент в позиции (5,2).

8 Использование BLOBS в Firebird

Большие двоичные объекты (Binary Large Objects BLOBS) являются в базах данных Firebird контейнерами для данных практически неограниченного размера. На практике размер BLOBS ограничен ограничениями СУБД и доступным дисковым пространством, но, возможно, самым важным моментом является то, что ограничение на их индивидуальный размер не является частью метаданных . IBX поддерживает использование полей BLOB в СУБД.

8.1 Типы BLOB

С точки зрения IBX существует два типа BLOB:

- Текстовые BLOB (подтип 1), которые содержат символьные данные некоторого набора символов (например,UTF8), или
- Двоичные BLOB, для которых тип данных неизвестен для IBX.

Замечание: Firebird позволяет использовать различные типы BLOB в дополнение к текстовому типу не придавая им никакого конкретного смысла. IBX также не придает никакого дополнительного смысла для нетекстовых типов BLOB.

8.1.1 Текстовые BLOB

Текстовые BLOB для набора данных представлены типом поля TIBMemoField. Он является потомком TBlobField и предоставляет доступ к BLOB данным в виде строк AnsiString:

- Для чтения или записи текста целиком используйте свойство AsString. При чтении используется кодовая страница AnsiString, соответствующая набору символов поля данных. При записи возможна транслитерация, если кодовая страница AnsiString отличается от набора символов для поля данных.
- Для сохранения значения текстового поля в файл используйте метод SaveToFile.
- Для загрузки текстового BLOB поля из файла используйте метод LoadFromFile.

Для отображения и редактирования текста можно использовать виджет управления данными TIBMemo,. На практике главным различием между TIBMemoFields и обычными строковыми полями, является то, что текстовые поля BLOB практически не имеют ограничений на размер данных в поле.

8.1.2 Двоичные BLOBs

Двоичные поля BLOB используются для многих целей, включая хранение изображений, музыки и т.п.. Двоичные поля BLOB представлены типом поля набора данных TBlobField :

- Для чтения или записи всего поля в виде строки текста используйте свойство AsString. Строки всегда считываются с кодовой страницей NONE , а при записи значение кодовой страницы игнорируется.
- Для сохранения значения двоичного поля BLOB в файл используйте метод SaveToFile.
- Для считывания значения двоичного поля BLOB из файла используйте метод LoadFromFile.

Если в двоичном поле хранится изображение, то для их отображения можно использовать виджет управления данными TDBImage.

8.2 Доступ к полям BLOB с использованием потоков (Stream)

IBX позволяет читать и записывать поля BLOB при помощи класса TStream. Наборы данных IBX, являющиеся потомком TDataset, имеют метод CreateBlobStream.

function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): Tstream;

Для полей BLOB следует использовать TBlobField или TIBMemoField. Аргумент mode определяет, создается ли поток для чтения (bmRead) или для записи (bmWrite). Возможен также доступ на чтение/запись. Для использования записи набор данных должен быть в состоянии Edit. Поток BLOB может читать или писать данные так же, как любой потомок TStream.

Изменения поля BLOB с использованием Blob потока изменяет данные в текущей записи набора данных. Для того, чтобы они сохранились в базе данных, их необходимо зафиксировать (posted).

9 Использование массивов Firebird

Firebird позволяет вам создавать типизированные массивы. Использование массивов позволяет хранить в одном поле множество элементов данных. Firebird может производить операции с массивом целиком, обрабатывая его как один элемент, или использовать срез — подмножество элементов массива. Срез массива может состоять из единственного элемента или непрерывного ряда элементов.

Начиная с IBX2, IBX теперь предоставляет полную поддержку массивов Firebird:

- Firebird Language Bindings в пакете *fbintf* предоставляет интерфейсы IArray и IArrayMetaData. Первый используется для доступа к массивам или их срезам, а у последнего можно запрашивать информацию о массивах (например, его тип, количество размерностей и т.п.).
- В состав IBX теперь входит TIBArrayField. Он является потомком TField и, как и все остальные потомки TField (например, TStringField) предоставляет средства для доступа к элементам массива. TIBArrayField также предоставляет интерфейс IArray в виде одного из своих свойств, позволяя прямой доступ к массиву.
- Пакет IBControls теперь включает виджет управления данными TIBArrayGrid (см. 12.5 TIBArrayGrid). Он является потомком TCustomStringGrid и может использоваться для просмотра и редактирования элементов массива. Приводятся примеры как для одномерных, так и для двумерных массивов.

Документация на интерфейс IArray входит в состав документации на пакет *fbintf* . См. Раздел 8 Firebird Pascal API Guide.

9.1 Задание элементов массива

Полное описание содержится в документации Firebird. Например:

```
Alter Table MyData (
...
MyArray VarChar(16) [0:16, -1:7] Character Set UTF8
);
```

Массив может иметь для каждой записи различные наборы значений. В приведенном выше примере задан двумерный массив строк, где первый индекс меняется в пределах от 0 до 16, а второй от -1 до 7.

9.2 Поле TIBArrayField

Поле TIBArrayField используется и ведет себя также, как и другие потомки TField. Его можно создать с помощью IDE, используя редактор полей и сохранить, как часть формы (а затем обращаться как к свойству формы) или динамически, при открытии потомка TIBCustomDataSet (например, TIBTable, TIBQuery, и т.д..) . В этом случае к нему можно обращаться с помощью метода FieldByName.

TIBArrayField определяет следующие дополнительные свойства:

ArrayID: TISC_QUAD Предоставляет доступ к внутреннему массиву идентификаторов Firebird хранимых в самой записи. Обычно оно малоинтересно для пользователей.

ArrayIntf: IArray Предоставляет доступ к массиву интерфейсов, используемых для

доступа и изменения элементов данных. Если значение поля равно null, то возвращается пустой массив, а установка любого элемента массива в непустое значение , если запись будет зафиксирована, делает поле непустым.
Присваивание полю пустого массива имеет противоположный смысл: значение поля становится пустым (null). (Рекомендуется просто устанавливать свойство IsNul).

ArrayDimensions: integer Возвращает размерность массива.

ArrayBounds: TArrayBounds Это динамический массив Паскаля, возвращающий элемент для каждого измерения, предоставляя верхнюю и нижнюю границу для каждого измерения.

TIBArrayField также предоставляет дополнительный метод:

```
function CreateArray: IArray;
```

Обычно он используется для получения интерфейса к новому, пустому массиву, который совместим с полем данных. После заполнения его можно присвоить свойству ArrayIntf и его содержимое будет сохранено в качестве значения поля, если набор данных будет "posted".

Замечание: можно сохранить копию ArrayIntf и обращаться к ней после того, как набор данных перейдет к другой записи. Однако при попытке изменить содержимое массива будет вызвано исключение, поскольку IArray не связан с текущей записью TDataSet.

10 Использование служб Firebird

Firebird Services API появились в InterBase 6.0 и присутствуют во всех версиях Firebird. Они обеспечивают:

- Доступ к свойствам и статистике сервера и базы данных
- Резервное копирование и восстановление базы данных
- Управление базой данных безопасности (учетные данные пользователя)
- Проверка базы данных и исправление ошибок
- Управление параметрами конфигурации базы данных .

Эти функции соответствуют функциональности предоставляемых Firebird утилит командной строки gbak, gfix и gsec.

IBX предоставляет доступ к службам API с помощью нескольких невизуальных компонент расположенных на закладке Firebird Admin палитры компонентов. Каждый компонент предназначен для решения своего круга задач, решаемых службами API. Примеры программ демонстрирующих использование каждого компонента с закладки Firebird Admin находятся в каталоге "ibx/examples/services".

10.1 Обзор компонентов Firebird Admin

 **TIBBackupService** Служба резервного копирования позволяет делать резервное копирование базы данных в архив формата gbak. Поддерживается создание резервных копий как на клиенте, так и на сервере.

 **TIBRestoreService** Служба восстановления поддерживает восстановление базы данных из архивов, сделанных в формате gbak. Файлы резервных копий могут находиться как на клиенте, так и на сервере.

 **TIBConfigService** Служба конфигурации позволяет изменять параметры базы данных, включая проверку есть ли подключения к базе данных, устанавливать синхронный или асинхронный способ записи и т.д.

 **TIBServerProperties** Эта службы запрашивает различные свойства сервера, включая версию сервера, его параметры и текущее состояние соединения с базой.

 **TIBLogService** Эта служба служит для чтения журнала сервера.

 **TIBStatisticalService** Эта служба извлекает статистику по базе данных.

 **TIBSecurityService** Эта служба управляет базой данных по безопасности пользователей.

 **TIBValidationService** Эта служба поддерживает выполнение различных действий по восстановлению базы данных, включая проверку и очистку . Могут использоваться и Лимбо транзакции.

10.2 Общие свойства служб

Все компоненты Firebird Admin имеют общего предка и, соответственно, используются

похожим образом . На этапе проектирования может использоваться общий редактор для установки параметров входа по умолчанию. Следующие свойства являются общими для служб:

Active — Установите это свойств в true для подключения к серверу и установления с ним соединения. Установите его в false для завершения активного соединения.

LoginPrompt — Установите свойство в true для вызова встроенного диалога для запроса параметров входа (пользователя и пароля).

Params — Содержит имя пользователя и пароль в виде списка строк вида имя_параметра=значение (например, user_name=SYSDBA, password=masterkey). Задавать пароль на этапе проектирования не рекомендуется. Для установки параметров имеется специальный редактор свойства.

Protocol — Определяет тип соединения (локальное, TCP, SPX или именованный канал). В windows могут использоваться только первые два.

ServerName — Имя сервера (доменное).

Открытое свойство **ServiceIntf** предоставляет интерфейс IServiceManager, используемый для связи с сервером. Этот интерфейс доступен (не равен nil), если Active установлено в true. Его значение можно присваивать другим компонентам Firebird Admin, позволяя пользоваться одним соединением без необходимости логиниться для каждой службы.

Замечание: Установка свойства active в false разорвет соединение и сделает неактивными все разделяемые интерфейсы.

10.3 Служба резервного копирования

Служба резервного копирования позволяет делать резервные копии базы данных в формате gbak . Файлы резервных копий можно делать как на клиенте, так и на сервере. Перед тем, как запустить резервное копирование, надо установить общие свойства, плюс следующие:

BackupFile — Только для создания копий на сервере. Это список путей к одному или более файлам резервных копий на сервере. Если указан более, чем один файл, то все имена , кроме последнего должны оканчиваться на "-nnn", где nnn максимальная длина файла в байтах.

BackupFileLocation — И для клиента и для сервера. Задает, где должен создаваться файл копии — на клиенте или на сервере.

BlockingFactor — Смотрите документацию на gbak (вероятно, устарела)

DatabaseName — Алиас или полный путь к базе данных на сервере.

Options — Смотрите документацию на gbak для понимания возможных значений.

Verbose — Только для копий на сервере: если равно true, то генерируются дополнительные текстовые сообщения.

10.3.1 Создание копии на сервере

Следующий код иллюстрирует как проводится создание копии на стороне сервера. Предполагается, что описанные выше свойства уже заданы:

```
IBBackupService1.Active := true;
IBBackupService1.ServiceStart;
while not IBBackupService1.Eof do
   writeln(IBBackupService1.GetNextLine);
Application.ProcessMessages;
```

```
IBBackupService1.Active := false; {в случае, если соединение больше не нужно}
```

После того, как служба запущена, выполнение сводится к простому циклу проверки на "EOF" после вызова метода GetNextLine. Этот метод возвращает строку текста, полученную от сервера (наиболее акуально в режиме verbose). В приведенном примере строки выводятся в stdout.

10.3.2 Создание копии на клиенте

Следующий код иллюстрирует как проводится создание копии на стороне клиента. Предполагается, что описанные выше свойства уже заданы:

```
var bakfile: TFileStream;
begin
  bakfile := TFileStream.Create('<path to backup file>',fmCreate);
  try
    IBBackupService1.Active := true;
    IBBackupService1.ServiceStart;
    while not IBBackupService1.Eof do
    begin
      IBBackupService1.WriteNextChunk(bakfile);
      Application.ProcessMessages
    end;
  finally
    bakfile.Free;
  end;
end;
```

Этот код аналогичен приведенному для сервера с тем отличием, что пользователь должен предоставить поток (в примере TFileStream), куда будет записана копия. Вместо GetNextLine, на клиенте используется WriteNextChunk.

10.4 Служба восстановления

Служба восстановления поддерживает восстановление базы данных из архивов формата gbak. Поддерживается восстановление из файлов, хранящихся как на клиенте, так и на сервере. До запуска восстановления должны быть заданы общие свойства службы плюс приведенные ниже:

BackupFile — Только восстановления из копий на сервере. Это список путей к одному или более файлам резервных копий на сервере. Если указан более, чем один файл, то они считываются в том же порядке, в котором заданы.

BackupFileLocation — И для клиента и для сервера. Задает, где находится файл копии — на клиенте или на сервере.

DatabaseName — Алиас или полный путь к базе данных на сервере.

Options — Смотрите документацию на gbak для получения возможных значений.

- CreateNewDB (по умолчанию) или Replace, но не оба вместе.

PageBuffers — Смотрите документацию на gbak

PageSize — Смотрите документацию на gbak

Verbose — Только для копий на сервере: если равно true, то генерируются дополнительные текстовые сообщения.

10.4.1 Восстановление на стороне сервера

Ниже приводится код восстановления на стороне сервера после того, как установлены все необходимые параметры:

```
IBRestoreService1.Active := true;
IBRestoreService1.ServiceStart;
while not IBRestoreService1.Eof do
begin
  writeln(IBRestoreService1.GetNextLine);
  Application.ProcessMessages
end;
```

После того, как служба запущена, выполнение сводится к простому циклу проверки на "EOF" после вызова метода GetNextLine. Этот метод возвращает строку текста, полученную от сервера (наиболее актуально в режиме verbose). В приведенном примере строки выводятся в stdout.

10.4.2 Восстановление из файлов на клиенте

Ниже приводится код восстановления на стороне клиента после того, как установлены все необходимые параметры:

```
var bakfile: TFileStream;
line: string;
begin
  bakfile := TFileStream.Create('<path to backup file>',fmOpenRead);
  try
    IBRestoreService1.Active := true;
    IBRestoreService1.ServiceStart;
    while not IBRestoreService1.Eof do
    begin
      IBRestoreService1.WriteNextChunk(bakfile,line);
      if line <> '' then
        writeln(line);
      Application.ProcessMessages
    end;
  finally
    bakfile.Free;
  end;
end;
```

Этот код аналогичен приведенному для сервера с тем отличием, что пользователь должен предоставить поток (в примере TFileStream) куда будет записана копия. Вместо GetNextLine, на клиенте используется WriteNextChunk. Оба эти метода осуществляют чтение из потока и могут вернуть строку, если получат ее от сервера

10.5 Служба конфигурации

Служба TIBConfigService также имеет общие для служб свойства, но кроме этого, она содержит набор методов, каждый из которых выполняет определенное действие.

ShutdownDatabase Переводит базу данных в состояние завершения работы в соответствии с выбранными параметрами и в течение заданного времени ожидания (секунд). После

этого только пользователь SYSDBA сможет войти в базу данных.

BringDatabaseOnline	Перевод базы данных в оперативное состояние (обратная по отношению завершению работы операция) .
SetSweepInterval	Установка интервала автоматической чистки
SetDBSqlDialect	Значение SQL диалекта базы по умолчанию (1 или 3)
SetPageBuffers	Количество кэш-буферов по умолчанию.
ActivateShadow	Активирует "теневой файл" базы данных. См. Документацию по Firebird .
SetReserveSpace	Предписывает базе данных заполнять страницу данных при вставке новых записей (false), или резервировать 20% на каждой странице для будущих изменений (true)
SetAsyncMode	Переключатель для асинхронной записи (true) и синхронной (false).
SetReadOnly	Устанавливает режим «только для чтения» или чтения/записи.

10.6 Служба свойств сервера

Эта служба показывает различные свойства сервера, включая версию сервера, его параметры и текущее состояние соединения с базой данных. Получаемая информация делится на:

- Информация о версии сервера
- Информация об активных базах данных
- Параметры конфигурации

Для каждого бока информации имеются соответствующие методы, которые устанавливают значения, связанных с ними свойств. Значения свойств могут быть считаны для получения нужной информации. Например:

```
var i: integer;
begin
  with IBServerProperties1 do
  begin
    Active := true;
    FetchVersionInfo;
    writeln('Server Version = ' + VersionInfo.ServerVersion);
    writeln('Server Implementation = ' + VersionInfo.ServerImplementation);
    writeln('Service Version = ' + IntToStr(VersionInfo.ServiceVersion));
    FetchDatabaseInfo;
    writeln('No. of attachments = ' + IntToStr(DatabaseInfo.NoOfAttachments));
    writeln('No. of databases = ' + IntToStr(DatabaseInfo.NoOfDatabases));
    for i := 0 to DatabaseInfo.NoOfDatabases - 1 do
      writeln('DB Name = ' + DatabaseInfo.DbName[i]);
    FetchConfigParams;
    writeln('Base Location = ' + ConfigParams.BaseLocation);
    writeln('Lock File Location = ' + ConfigParams.LockFileLocation);
    writeln('Security Database Location = ' + ConfigParams.SecurityDatabaseLocation);
  end;
end;
```

10.7 Служба чтения журнала

Это простая служба, позволяющая получение текущего содержимого журнала сервера. Например:

```
with IBLogService1 do
begin
    Active := true;
    ServiceStart;
    while not Eof do
    begin
        writeln(GetNextLine);
        Application.ProcessMessages;
    end;
end;
```

10.8 Служба статистики базы данных

Эта служба позволяет извлекать статистику для заданной базы данных в виде текстовых данных. Использование аналогично получению журнала, за исключением:

- Свойство DatabaseName должно указывать алиас или полный путь к базе данных, для которой запрашивается статистика.
- Свойство Options должно определять, какая именно статистика требуется. В остальном процесс аналогичен получению журнала. Например:

```
with IBStatisticalService1 do
begin
    DatabaseName := 'myDatabase';
    Options := [HeaderPages];
    Active := true;
    ServiceStart;
    while not Eof do
    begin
        writeln(GetNextLine);
        Application.ProcessMessages;
    end;
end;
```

В этом примере возвращается статистика главной страницы. Допустимые значения:

HeaderPages	Запрос только информации по главной странице базы данных
DataPages	Запрос статистики по пользовательским страницам данных
IndexPages	Запрос статистики по индексным страницам пользователя
SystemRelations	Запрос статистики по системным таблицам и индексам (в дополнение к пользовательским)

10.9 Служба безопасности

Эта служба управляет базой данных безопасности пользователей. Она обеспечивает:

- Отображение списка Имен Пользователей и другой идентификационной информации
- Добавление новых пользователей
- Изменение существующих пользователей (включая изменение паролей)
- Удаление пользователей

10.9.1 Отображение списка пользователей

Для извлечения списка имен пользователей и другой идентификационной информации используется метод DisplayUsers, который заполняет свойство UserInfo. Затем эта информация может быть показана пользователю. Например:

```
var i: integer;
begin
   with IBSecurityService1 do
   begin
      Active := true;
      DisplayUsers;
      for i := 0 to UserInfoCount - 1 do
       with UserInfo[i] do
      begin
         writeln('User ID = ',UserID);
          writeln('Group ID = ',GroupID);
          writeln('User Name = ',UserName);
         writeln('First Name = ', FirstName);
         writeln('Middle Name = ', MiddleName);
         writeln('Last Name = ', := LastName);
      end;
   end;
end;
```

10.9.2 Добавление пользователя

Для добавления пользователя в базу данных безопасности используется метод AddUser . Перед вызовом метода должны быть установлены свойства UserName и Password для задания имени пользователя и пароля. Полный список свойств, которые могут быть заданы:

UserName	Имя пользователя (или Login)
Password	Пароль пользователя
FirstName	Имя пользователя
MiddleName	The user's middle name???
LastName	Фамилия пользователя
UserID	UID для ОС UNIX
GroupID	GID для ОС UNIX

Ниже приводится пример, иллюстрирующий использование метода AddUsers:

```
with IBSecurityService1 do
begin
   Active := true;
   UserName := NewUserName;
   Password := NewPassword;
   AddUser;
end;
```

10.9.3 Изменение пользовательских данных

Для изменения логина и пользовательских данных служит метод ModifyUser. Свойство

UserName действует как ключ, идентифицирующий пользователя . Остальные свойства, описанные выше в разделе 10.9.2 могут быть изменены путем обновления соответствующей записи в базе данных . Например:

```
with IBSecurityService1 do
begin
    Active := true;
    UserName := 'SYSDBA';
    FirstName := 'Donald';
    LastName := 'Duck';
    ModifyUser;
end;
```

10.9.4 Удаление пользователя

Для удаления пользовательских данных из базы данных безопасности используется метод DeleteUser. Например:

```
with IBSecurityService1 do
begin
    Active := true;
    UserName := 'ALICE';
    DeleteUser;
end;
```

10.10 Служба проверки и восстановления

Эта служба поддерживает выполнение различных действий по восстановлению базы данных, включая проверку и чистку. Служба также позволяет разрешить Limbo транзакции. Служба фактически состоит из двух служб. Первая занимается выполнением действий по восстановлению данных. Вторая служит для разрешения Limbo транзакций.

10.10.1 Ремонт базы данных

Ниже описаны доступные способы восстановления, которые могут быть заданы с помощью свойства options:

Заголовок	**Option**	**Описание**
Список Limbo транзакций	LimboTransactions	Выдает список limbo транзакций в текстовом виде
Проверка базы данных	CheckDB	Проверка базы данных без коррекции продлем
Пропуск ошибок контрольных сумм	IgnoreChecksum	Уточняет режим проверки

Заголовок	Option	Описание
Удаление теневых файлов	Kil Shadows	Удаляет ссылки на недостижимые теневые файлы
Чинить базу данных	MendDB	Пометить испорченные записи как недоступные, чтобы другие операции их игнорировали
Чистка базы данных	SweepDB	Запрос очистки базы данных для пометки устаревших записей как свободного места
Проверка базы данных	ValidateDB	Найти и освободить страницы, не назначенные никаким структурам данных
Полная проверка базы данных	ValidateFull	Проверить структуры записей и страниц, освободив неназначенные записи

Например:

```
with IBValidationService1 do
begin
    DatabaseName := 'MyDatabase';
    Options := [ValidateDB,ValidateFull];
    Active := true;
    ServiceStart;
    while not Eof do
    begin
        writeln(GetNextLine);
        Application.ProcessMessages;
    end;
end;
```

10.10.2 Разрешение Limbo транзакций

Разрешение limbo транзакций проводится в два этапа. На первом этапе составляется список всех limbo транзакций. На втором этапе транзакции подтверждаются или откатываются в соответствии с составленным заданием.

Для создания списка limbo транзакций используется метод FetchLimboTransactionInfo. По завершении список будет находиться в свойстве LimboTransactionInfo. Например:

```
var i: integer;
begin
with IBValidationService1 do
begin
    Active := true;
    ServiceStart;
    FetchLimboTransactionInfo;
```

```
    for i := 0 to LimboTransactionInfoCount - 1 do
      with LimboTransactionInfo[i] do
      begin
        write('ID = ',ID);
        if MultiDatabase then
          write(', Multi DB')
        else
          write(' ,Single DB');
        write(', Host Site = ', HostSite);
        write(', Remote Site = ', RemoteSite);
        write(', Database Path = ', RemoteDatabasePath);
        write(', State = ', StateToStr(State));
        writeln(', Advise = ', AdviseToStr(Advise));
      end;
  end;
end;
```

где,

```
function StateToStr(State: TTransactionState): string;
begin
  case State of
  LimboState:
    Result := 'Limbo';
  CommitState:
    Result := 'Commit';
  RollbackState:
    Result := 'Rollback';
  else
    Result := 'Unknown';
  end;
end;

function AdviseToStr(Advise: TTransactionAdvise): string;
begin
  case Advise of
  CommitAdvise:
    Result := 'Commit';
  RollbackAdvise:
    Result := 'Rollback';
  else
    Result := 'Unknown';
  end;
end;
```

Этот список идентифицирует каждую limbo транзакцию и ее текущее состояние, а также предлагает действие (советует). Пользователь может просмотреть список транзакций и установить свойство TLimboTransactionInfo.Action в желаемое значение.

Затем можно исправить limbo транзакции, установив свойство GlobalAction и вызвав метод FixLimboTransactionErrors.

GlobalAction определяет как будет FixLimboTransactionErrors обрабатывать limbo транзакции и может иметь значения:

CommitGlobal	Подтвердить (коммитить) все limbo транзакции

RollbackGlobal Откатить все limbo транзакции.

RecoverTwoPhaseGlobal Двухфазное подтверждение всех limbo транзакций

NoGlobalAction Limbo транзакции подтверждаются или откатываются в зависимости от значения их свойства Action.

Например:

```
with IBValidationService1 do
begin
   GlobalAction := NoGlobalAction
   FixLimboTransactionErrors;
   while not Eof do
   begin
      writeln(GetNextLine);
      Application.ProcessMessages;
   end;
end;
```

11 Личные базы данных

Личные базы данных это базы, которые размещаются на клиентской файловой системе и, если возможно, файловая система гарантирует, что только пользователь имеет доступ к данным. В отличие от доступа к базе данных с использованием удаленного сервера, в этом случае сервер встроен в клиента и наследует ограничения и права доступа пользователя .

В версии Firebird 2.5 и более ранних встроенный сервер распространялся в виде отдельного пакета, тогда как в Firebird 3 одни и те же библиотеки программ могут быть как частью автономного сервера, так и встроенного сервера. Встроенный сервер используется, если к нему имеется доступ, а файл базы данных расположен на локальном компьютере. Если пользователь имеет недостаточно прав, то доступ к база данных может быть запрещен,. Пакет *fbintf* предоставляет прямое и практически прозрачное использование встроенного сервера. Для более подробной информации см. раздел 4.10 документации Firebird Pascal API Guide. Рекомендации по развертыванию доступны в главе 13 того же руководства .

Дополнительно IBX распознает случаи, когда база данных расположена на локальном компьютере, но ошибка открытия не позволяет использовать встроенный сервер. В этом случае путь к базе автоматически дополняется префиксом "localhost:" и попытка повторяется в надежде использовать локальный сервер. IBX предоставляет также дополнительную поддержку для личных баз данных с использованием встроенного сервера.

11.1 Компонент TIBLocalDBSupport

TIBLocalDBSupport является невизуальным компонентом, используемый TIBDatabase и разработанный для того, чтобы упростить применение встроенного сервера firebird для личных баз данных как на платформе Linux, так и Windows. Компонент TIBLocalDBSupport предназначен для использования с графическим интерфейсом GUI, а компонент TIBCMLocalDBSupport предоставляет аналогичную поддержку для консольных программ. Приводятся примеры программ как для графического, так и для консольного режима. Примеры расположены в каталоге ibx/examples/local-employeedb.

После активации, TIBLocalDBSupport позволяет:

- Проверить доступность встроенного сервера Firebird.
- Задать значение для переменной окружения FIREBIRD для встроенного сервера.
- Задавать имя базы данных и параметры входа.
- Использовать службы Firebird API для инициализации пустой базы данных на основе архива в формате gbak.
- Создавать резервные копии локальной базы данных в формате gbak с использованием служб FirebirdAPI.
- Восстанавливать содержимое базы данных из резервной копии локальной базы данных в формате gbak с использованием служб FirebirdAPI.
- Использовать процессор выполнения скриптов TIBXScript для автоматического обновления полей локальной базы данных.

Для использования компоненты, просто поместите ее на форму и свяжите с компонентом TIBDatabase.

11.1.1 Свойства

Database	ссылка на компоненту TIBDatabase с локальной базой данных
DatabaseName	имя файла (без пути) , где находится база данных Firebird.
EmptyDBArchive	имя файла (необязательный путь) содержащий архив инициализируемой базы данных. Может содержать абсолютный путь или путь относительно каталога с разделяемыми данными.
Enabled	Если равен false, то компонент неактивен
FirebirdDirectory	Полный путь к каталогу, содержащему файл конфигурации firebird.conf. Может быть абсолютным или заданным относительно каталога разделяемых данных. Если не указан, то принимается за каталог разделяемых данных.
Name	Имя компоненты
Options	iblAutoUpgrade: Автоматически апргейдить, если версия схемы базы данных младше текущей. IblAllowDowngrade: Если возможно, то автоматически понижать версию схемы базы данных, чтобы была совместима с текущей. iblQuiet: Если равно true, то перезапись базы производится без вывода предупреждений
RequiredVersionNo	Номер версии схемы, запрашиваемый приложением. TIBLocalDBSupport попытается поднять или понизить схему, чтобы удовлетворить это требование.
UpgradeConfFile	Путь к файлу апгрейда конфигурации. Может быть абсолютным или заданным относительно каталога разделяемых данных.
VendorName	Используется для создания пути к каталогу базы данных.

Отметим, что на этапе проектирования пути в качестве разделителей каталогов могут использовать как '/' так и '\'. Во время выполнения они должны использовать разделитель использующийся данной платформой.

11.1.2 События:

OnGetDatabaseName	Путь к базе данных обычно формируется автоматически. Однако, это событие позволяет проверить и исправить результат.
OnNewDatabaseOpen	Вызывается после успешной инициализации пустой базы данных.
OnGetDBVersionNo	Вызывается для получения схемы базы данных нужной версии. Если обработчик не задан, апгрейд/даунгрейд никогда не проводится.
OnGetSharedDataDir	Каталог разделяемых данных обычно формируется автоматически. Однако, это событие позволяет проверить и исправить результат.

11.1.3 Каталог разделяемых данных

Каталог разделяемых данных это базовый каталог для всех постоянных файлов данных, используемых TIBLocalDBSupport. Его размещение задается в соответствии:

- В ОС Windows: место размещения выполняемого файла.
- В ОС Unix: место размещения выполняемого файла, за исключением случая, когда он находится в каталогах /usr/bin, /usr/local/bin, /usr/sbin или /usr/local/sbin, когда каталог разделяемых данных устанавливается в /usr/share/<имя_приложения> или /usr/local/share/<имя_приложения> в зависимости от того, находится приложение в каталоге /usr или /usr/local.

Отметим, что <имя_приложения> берется из sysutils.ApplicationName и по умолчанию равно имени исполняемого файла без расширения.

11.1.4 Управление именем базы данных и параметрами входа

В случае использования TIBLocalDBSupport свойство TIBDatabase.DatabaseName игнорируется и вместо этого используется следующий алгоритм:

- В ОС Windows: "User Application Directory"\VendorName\DatabaseName
- В ОС Unix: "User Home Directory"/."VendorName"/DatabaseName

Где "DatabaseName" берется из свойства TIBLocalDBSupport.DatabaseName .

"VendorName" берется из свойства TIBLocalDBSupport.VendorName. Если оно не задано, то используется Sysutils.VendorName. Если и оно не задано, то VendorName составляющая пути убирается совсем.

Помните об использовании скрытых каталогов в ОС Unix.

Если автоматически создаваемое имя базы данных не подходит, то его можно изменить в обработчике события TIBLocalDB.OnGetDatabaseName.

Свойство Params копируется из компонента TIBDatabase , за исключением параметров "user_name" и "password", которые удаляются, если были. При запуске в ОС Windows, "user_name" устанавливается в "SYSDBA" , а "password" в "masterkey". При запуске в ОС Unix, эти параметры опускаются.

11.1.5 Инициализация базы данных

Когда свойство connected компонента TIBDatabase устанавливается в "true", то компонент TIBLocalDBSupport генерирует имя файла базы данных (как описано выше) и, если такой файл отсутствует, то TIBLocalDBSupport использует службы Firebird API для создания файла с помощью архива "empty database" в формате gbak. На практике, этот архив кроме метаданных может содержать начальные данные.

Архив "Empty database" задается свойством TIBLocalDBSupport.EmptyDBArchive. Это должно быть имя файла (обычно с расширением .gbk) и может включать необязательный путь. Относительные пути трактуются относительно каталога разделяемых данных.

Для создания из архива начальной базы данных используются службы API (см. 10.4 Служба восстановления). Если архив не найден, то вызывается исключение.

Инициализацию базы можно делать в любой момент при помощи метода TIBLocalDBSupport.

11.1.6 Сохранение текущей базы данных

Текущее содержимое базы данных можно сохранить в любой момент с помощью вызова TIBLocalDBSupport.SaveDatabase. Имя файла создаваемого архива можно задать при вызове метода. Если оно не задано, то пользователя попросят ввести имя файла (расширение по умолчанию .gbk).

Затем будет вызвана служба API (см. 10.4 Служба восстановления10.3 Служба резервного

копирования) которая сделает резервную копию в файла с указанным именем в формате gbak.

11.1.7 Восстановление базы данных из архива

Локальная база данных может быть перезаписана (восстановлена) из любого архива в формате gbak (включая созданные с использованием метода SaveDatabase) с помощью вызова метода TIBLocalDBSupport.RestoreDatabase. Имя файла архива можно задать при вызове метода. Если оно не задано, то пользователя попросят ввести имя файла .

Затем будет вызвана служба API (см. 10.4 Служба восстановления) которая восстановит базу данных из файла с указанным именем.

11.1.8 Апгрейд схемы базы данных

Обновление программного приложения может потребовать и соответствующего обновления схемы базы данных. С помощью встроенных серверных приложений Firebird, где пользователь может даже не знать, какой сервер базы данных используется, важно иметь средства для обновления схемы базы данных как можно более прозрачным и автоматическим способом . Компонент TIBLocalDBSupport предоставляет соответствующий механизм с помощью процессора скриптов TIBXScript (см. 7.1 Процессор скрипов IBX).

Базовая идея состоит в том, что схема базы данных поставляется вместе с номером версии, задаваемым целым числом. Первая версия имеет номер 1, вторая 2 и т.д. Текущий номер версии должен храниться где-нибудь в базе данных. Поскольку это место само зависит от схемы базы данных, то TIBLocalDBSupport не знает, как определить текущий номер схемы базы данных и вместо этого полагается на приложение, обрабатывающее событие OnGetDBVersionNo .

Каждое приложение должно содержать минимальный и максимальный номер версии схемы базы данных с которой оно работает и предполагается, что проверка соответствия версии схемы происходит в обработчике TIBDatabase OnConnect. Однако, перед вызовом этого обработчика TIBLocalDBSupport сам производит проверку текущей версии схемы на соответствие его свойству RequiredVersionNo (в котором должен быть установлен максимальный поддерживаемый номер версии).

- Если в свойстве Options задано iblAutoUpgrade и текущая схема меньше, чем Required Version номер., то TIBLocalDBSupport пытается применить правила апгрейда для того, чтобы повысить номер версии до требуемого.
- Если в свойстве Options задано iblAllowDowngrade и текущая схема больше, чем Required Version номер, то TIBLocalDBSupport делает попытку найти подходящую резервную копию и восстановить из нее базу данных. Такая ситуация обычно складывается только в маловероятном случае неудачного обновления, когда пользователь установил более старую версию программного обеспечения, чтобы избавиться от проблемы.

Правила обновления схемы считываются из файла конфигурации обновления. Это текстовый файл в формате "ini" со следующими разделами

[status]
Должен содержать единственный параметр по имени "current", задающий текущий номер схемы в виде целого числа, например, current = 2
Обычно он должен быть равен тому же значению, что задано для свойства RequiredVersionNo и служит для проверки соответствия версий.

[Version.nnn]

Где nnn представляет собой целое с ведущими нулями. Например, "Version.002" является разделом для обновления схемы базы данных с версии 1 до версии 2 . Этот раздел может содержать следующие ключевые поля:

Имя	Тип	Назначение
Upgrade	string	Имя и, возможно, путь к скрипту SQL, используемому для проведения апгрейда. Путь может быть как абсолютным так и относительно файла конфигурации. В качестве разделителей при указании пути могут использоваться как прямой /,так и обратный слэш \.
Msg	string	Текстовое сообщение в окне хода выполнения скрипта. По умолчанию "Upgrading Database Schema to Version nnn".
BackupDatabase	yes/no	Если присутствует и установлено в "yes", то перед началом апгрейда создается резервная копия. Файл резервной копии расположен там же, где и файл базы данных и имеет то же имя, но расширение вида ".nnn.gbak". Где "nnn" текущий номер версии схемы (т. е. номер до запуска апгрейда).
<Parameter Name>	string	Имя и необязательный путь к двоичному файлу данных. Путь может быть как абсолютным так и относительно файла конфигурации. В качестве разделителей при указании пути могут использоваться как прямой /,так и обратный слэш \.

Например:

```
[Version.002]
Msg = Upgrading to Version 2
BackupDatabase = yes
Upgrade = patches/02-patch.sql
mugshot = images/man.png.gz
```

В приведенном выше примере предполагается, что "mugshot" будет использовано для установления значений параметров при выполнении Update, Insert или Delete запросов в файле 02-patch.sql file. Например

```
Update EMPLOYEE Set Photo =:MUGSHOT Where Emp_no = 2;
```

Это применимо только к полям BLOB и в примере, приведенном выше, используется для изменения записи таблицы EMPLOYEE у которой Emp_no равно "2" , чтобы установить значение поля Photo в двоичные данные из файла "images/man.png.gz". Расширение ".gz" означает файл сжатый командой gzip и перед записью в поле разжимается.

Если текущая схема базы более, чем на 1 отличается от требуемой, то правила апгрейда применяются итеративно.

11.2 Пример локальной базы EmployeeDB

Этот пример предназначен для демонстрации использования компонента TIBLocalDBSupport . Этот компонент используется вместе с TIBDatabase для доступа к базе данных с использованием встроенного сервера Firebird . Компонент TIBLocalDBSupport

занимается проверкой и установкой переменный окружения FIREBIRD и параметров БД. Он позволяет также инициализировать локальную базу из архива в формате gbak, а также проводить резервное копирование и восстановление базы данных. Также он позволяет выполнять скрипты для апгрейда схемы базы данных при выпуске новых версий программного обеспечения. Пример расположен в каталоге: ibx/examples/local-employeedb/project1.lpi

Смотрите также пример для консольного режима.

Для компиляции и запуска примера программы необходимо установить встроенный сервер Firebird. Инструкция по установке встроенного сервера находится в разделе 13 Firebird Pascal API Guide.

11.2.1 Запуск приложения

Пример необходимо откомпилировать и запустить. Вместе с примером программы предоставляется архив базы данных employee. Его можно использовать для создания начальной базы данных. Затем он должен автоматически проапргрейдить базу данных до версии "version 2" используя скрипты, расположенные в каталоге "patches". (см. также файл upgrade.conf).

Заметим, что у вас не будут запрошены имя пользователя и пароль. Встроенный сервер для контроля доступа использует только обычные файловые права. В остальном вы можете редактировать базу данных employee так же как и в случае использования клиент/сервера.

Локальная база данных будет создана в:

- Linux: $HOME/.MWA Software/employee.fdb
- Windows: <User Application Data Folder>\MWA Software\employee.fdb

Меню File программы позволяет делать резервное копирование базы данных в формате gbak, и восстанавливать ее обратно (заменяя текущую базу данных) или восстанавливать ее к первоначальному виду.

11.2.2 Консольный режим

Также предоставляется пример консольного приложения, главный файл которого ibx/examples/local-employeedb/ConsoleModeExample.lpi.

Как и все консольные приложения, он использует пакет ibexpressconsolemode. Компонент TIBCMLocalDBSupport расположен в модуле IBCMLocalDBSupport. Приложение такое же, что и описанное выше и использует ту же базу данных и скрипты по апгрейду. В отличие от показа таблицы, консольное приложение при запуске выводит две первые строки.

12 Виджеты управления данными от IBX

В IBX 1.2 появилась новая закладка палитры компонентов "Firebird Data Controls". Она содержит четыре новых виджета управления данными которые относятся к IBX и используют синтаксический анализатор (парсер) SQL (см. 7.3 SQL парсер). В IBX2 был добавлен еще TIBArrayGrid.

Список виджетов IBX :

- TIBLookupComboEditBox
- TIBDynamicGrid
- TIBTreeview
- TDBControlGrid
- TIBArrayGrid

TIBLookupComboEditBox является потомком TDBLookupComboBox, который реализует «автоматическое дополнение» набираемого текста и «автовставку» новых записей. Автодополнение использует SQL для пересмотра доступного списка и ограничения его элементами, которые имеют префикс набранного текста (либо чувствительного к регистру, либо нет). Автовставка позволяет вновь набранную строку добавлять к списку набора данных и включать его в список доступных для выбора элементов.

TIBDynamicGrid является потомком TDBGrid , используемого для:

- автоматического изменения ширины столбцов так, чтобы заполнить доступную длину строки
- автоматически размещать и подгонять ширину для элемента управления "totals", обычно в конце столбца.
- переупорядочивание набора данных с помощью клика на заголовке столбца. Повторный клик на том же столбце меняет порядок сортировки
- Поддержка «Панельного редактора». Панельный редактор появляется при клике на столбец индикатора состояния записи (крайний левый на сетке) . При этом строка автоматически расширяется и ее замещает панель. Такая панель может иметь любое количество виджетов управления данными с таким же datasource, что и сетка, позволяя редактировать дополнительные поля и использовать более сложные редакторы.
- Повторный выбор той же строки после пересортировки.
- Новый редактор ячейки, который предоставляет ту же функциональность, что и TIBLookupComboEditBox. Его свойства задаются на основе столбца и позволяют одному или нескольким полям получать значения на основе списка, предоставляемого набором данных. Также доступны автозаполнение и автовставка. Существующий редактор раскрывающегося списка остается без изменений.

TIBTreeView является виджетом управления данными на основе TCustomTreeView.

TDBControlGrid является двойником, а не клоном для TDBCrtlGrid из состава Delphi. TDBControlGrid представляет собой таблицу в один столбец, в котором в каждой строке повторяются компоненты TWinControl — обычно TPanel или TFrame. Каждая такая строка соответствует строке подключенного набора данных. Все виджеты управления данными на повторяющихся элементах (например, TPanel) будут отображать данные соответствующей строки.

TIBArrayGrid является виджетом управления данными, порожденным от

TCustomStringGrid, который позволяет просмотр/редактирование содержимого одно или двумерных массивов полей Firebird.

Прилагаются примеры программ, иллюстрирующие применение новых виджетов управления данными.

12.1 TIBDynamicGrid

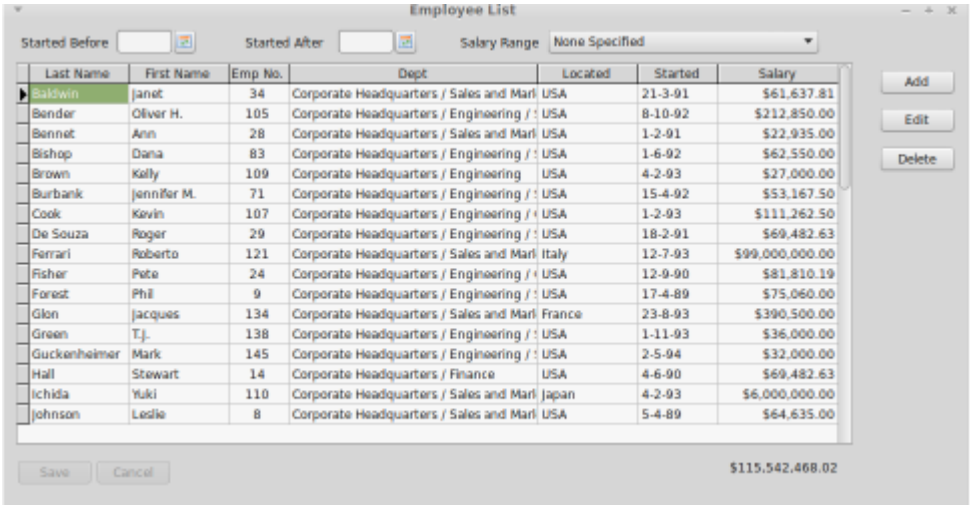


Рисунок 8: Компонент TIBDynamicGrid

Компонент TIBDynamicGrid показан на рисунке 8 с использованием базы данных "employee" из демонстрационного пакета Firebird. При использовании сетка данных выглядит практически также, как и TDBGrid и является ее потомком. Любой проект, использующий IBX и TDBGrid можно легко переделать для использования TIBDynamicGrid. Для управления сортировкой по столбцам этот компонент использует SQL.

Показанный пример можно найти в каталоге "ibx/examples/employee", где демонстрируются основные преимущества TIBDynamicGrid.

- Измените размеры формы и вы увидите, как столбец "Dept" автоматически увеличивается/сжимается так, чтобы заполнить сетку данных и как виджет "Total" для поля Salary (TDBText) перемещается так, чтобы быть выравненным с сеткой данных. Изменение ширины столбца задается во время проектирования с помощью свойства AutoSizeColumn для каждого столбца, которые должны динамически изменять размеры, причем его ширина при проектировании рассматривается как минимальная ширина. Ширина остальных столбцов остается постоянной.
- Щелчок по заголовку столбца "Started" (или любого другого заголовка) и таблица будет отсортирована по этому столбцу. Повторный клик по тому же столбцу меняет порядок сортировки.
- Выберите строку и нажмите F2, или кликнете по индикатору состояния записи (крайний левый столбец сетки) и появится Панельный редактор (см. рисунок 9). В результате вы сможете редактировать запись в свободном формате, не ограничиваясь простой строкой полей.
- При переоткрытии данных (например, после пересортировки или применения фильтра) курсор остается на той же записи.
- Фильтры, такие как "salary range", демонстрируют также, как работает новый SQL парсер совместно с TIBDynamicGrid. Например, если выбран диапазон окладов (salary), то набор данных переоткрывается с применением фильтра в обработчике события BeforeOpen.
- Строку данных по прежнему можно редактировать без применения панельного

редактора. Столбец "located" представляет собой пример использования в качестве редактора компонентf TIBLookupComboEditBox. Обратите внимание, что список стран создается динамически и меняется в зависимости от «Job Code» (ограничение базы данных Employee).

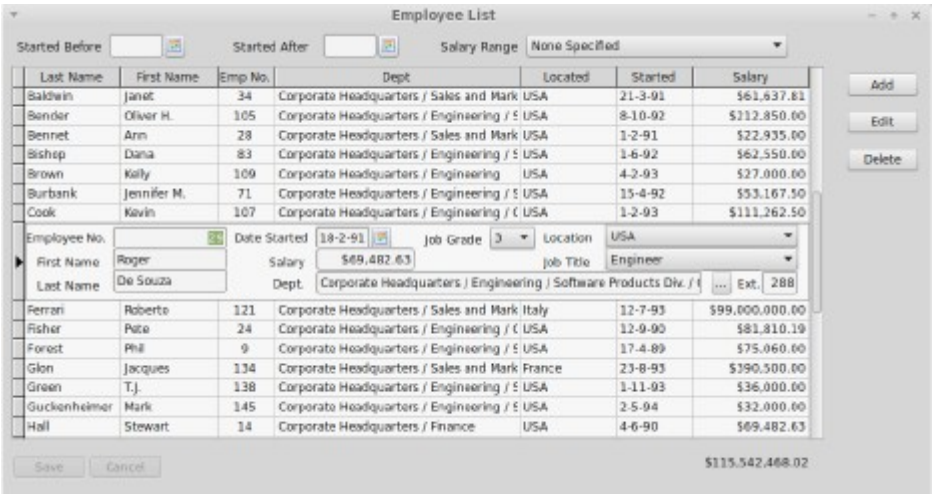


Рисунок 9:Сетка TIBDynamicGrid с открытым Панельным редактором

12.1.1 Свойства столбцов

Доступ к большинству новых возможностей TIBDynamicGrid происходит с помощью редактора столбцов, где задаются свойства для каждого столбца. Ниже приводятся новые свойства столбцов.

Свойство	Тип	Пояснения
AutoSizeColumn	Boolean	Если равно true, то столбец автоматически меняет ширину, чтобы заполнить сетку данных . Это свойство можно установить для несколких столбцов.
ColumnTotalsControl	TControl	Необязательное. Используется для указания виджета данных (обычно TDBEdit или TDBText) который должен быть выровнен по вертикали и иметь ту же ширину, что и этот столбец. Отметим, что итоговое поле может располагаться как выше, так и ниже сетки данных.
InitialSortColumn	Boolean	Отмечает столбец, по которому будет проведена сортировка при первом открытии набора данных.
DBLookupProperties	TDBLookupProperties	Это свойство копируется в TIBLookupComboBox если он используется в качестве редактора столбца. Установка свойства ListSource компонента TDBLookupProperties приводит к тому, что вместо выпадающего списка используется редактор столбца. Если свойство DataFieldName компонента

Свойство	Тип	Пояснения
		TDBLookupProperties не установлено, то элемент работает как «выпадающий список» со значениями из набора данных List Source.Если же свойство DataFieldName компонента TDBLookupProperties установлено, то оно действует как полный список подстановки. Свойство DataFieldName указывает поле в родительском наборе данных TIBDynamicGrid.DataSource.DataSet. Это поле не показывается в сетке данных. Когда редактирование заканчивается, в поле DataFieldName устанавливается значение поля TDBLookupProperties.KeyField из набора данных ListSource.

12.1.2 Новые свойства TIBDynamicGrid

Свойство	Тип	Комментарии
EditorPanel	TControl	Если установлен, то используется в качестве Панельного редактора (обычно это TPanel или TFrame) (см. ниже).
ExpandEditorPanelBelowRow	Boolean	Если установлено в true, то при показе Панельного редактора высота строки устанавливается в высоту строки плюс высота Панельного редактора и Панельный редактор показывается под строкой. То есть, строка по прежнему отображается. По умолчанию панельный редактор замещает строку.
AllowColumnSort	Boolean	Разрешает сортировку с помощью клика по заголовку столбца (по умолчанию true).
Descending	Boolean	Определяет начальный порядок сортировки. По умолчанию равен false, т. е. сортировка по возрастанию.
DefaultPositionAtEnd	Boolean	Определяет начальную строку при открытии набора данных. Если равно true, то устанавливается на последнюю строку. По умолчанию false.
IndexFieldNames	String	Список из одного или нескольких имен полей, разделенных точкой с запятой. Обычно это главный ключ набора данных. Используется для автоматической установки указателя строки при переоткрытии набора данных. На этапе проектирования можно использовать специальный редактор для выбора полей.

12.1.3 Новые события TIBDynamic

Название	Описание
OnBeforeEditorHide	Событие вызывается перед закрытием панельного редактора. Может использоваться для проверки корректности данных.
OnEditorPanelShow	Вызывается при показе Панельного редактора
OnEditorPanelHide	Вызывается **после** закрытия панельного редактора. Может использоваться для дополнительных действий.
OnKeyDownHander	Для обработки нажатий клавиш TIBDynamicGrid использует обработчик KeyDown, если активен Панельный редактор. Например для обработки "escape" клавиши как отмену редактирования. Для изменения поведения вы можете написать свой собственный обработчик.
OnColumnHeaderClick	Вызывается при щелчке по заголовку столбца, но до начала пересортировки. Можно использовать для изменения столбца, по которому будет проводиться сортировка.
OnUpdateSortOrder	Вызывается, если изменяется оператор выборки SQL, но до начала пересортировки. Может использоваться для изменения секции "Order by", например, для добавления столбца сортировки Полезно например, если один столбец содержит "year", а другой "month". Кликнув по "year" можно добавить сортировку по "month".
OnRestorePosition	Вызывается при открытии набора данных и может использоваться для изменения выбора начальной записи. Обработчик предоставляет var параметр, содержащий массив вариантов. Он может быть пустым с нулевой длиной или содержать столько же элементов, сколько задано в indexnames (см. свойство IndexFieldNames). В последнем случае он содержит значения соответствующих полей для ранее выбранной строки (т. е. Когда набор данных последний раз закрывался). При открытии набора первый раз этот массив пустой Этот указатель можно проверить и изменить на другие (значения ключей) или сделать пустым. В первом случае будет сделана попытка найти соответствующую запись и установить курсор на нее..Во втором случае курсор будет установлен на запись по умолчанию (см. свойство DefaultPositionAtEnd).

12.1.4 Панельный редактор

В качестве Панельного редактора может использоваться любой TControl, доступный на

форме. На практике это обычно TPanel или TFrame. В приводимом примере в качестве Панельного редактора используется TPanel. Вы можете создать Панельный редактор просто поместив ее на ту же форму, что TIBDynamicGrid , а затем выбрав ее, в качестве значения свойства EditorPanel компонента TIBDynamicGrid

Чтобы быть полезным, Панельный редактор должен быть заполнен виджетами управления данными, которые используют тот же источник данных (DataSource), что и сетка и будут использоваться для редактирования отдельных полей той же записи. Высота панели должна быть минимально необходимой, так как она определяет высоту Панельного редактора. Во время выполнения Панельный редактор автоматически делается невидимым, пока не будет вызван с помощью:

- a) Нажатия «F2» когда сетка является выбранной.
- b) Кликом по левому столбцу индикатора, или
- c) Вызова метода TIBDynamicGrid.ShowEditorPanel.

Для показа панели редактора производятся следующие действия:

- Высота текущей строки устанавливается равной высоте Панельного редактора
- Размер и положение Панельного редактора изменяется так, чтобы он точно накрывал текущую строку.
- Панельный редактор делается видимым.

Теперь текущую строку можно редактировать, используя ее виджеты управления данными.

Панельный редактор закрывается (делается невидимым и все изменения фиксируются в базе) в следующих случаях:

- a) Выбрана другая строка при помощи мыши или клавиш вниз/вверх
- b) Нажата клавиша Escape (отменяет все изменения)
- c) Нажата «F2».
- d) Вызван метод TIBDynamicGrid.HideEditorPanel.

После закрытия Панельного редактора текущая строка возвращает свою прежнюю высоту.

12.2 TDBControlGrid

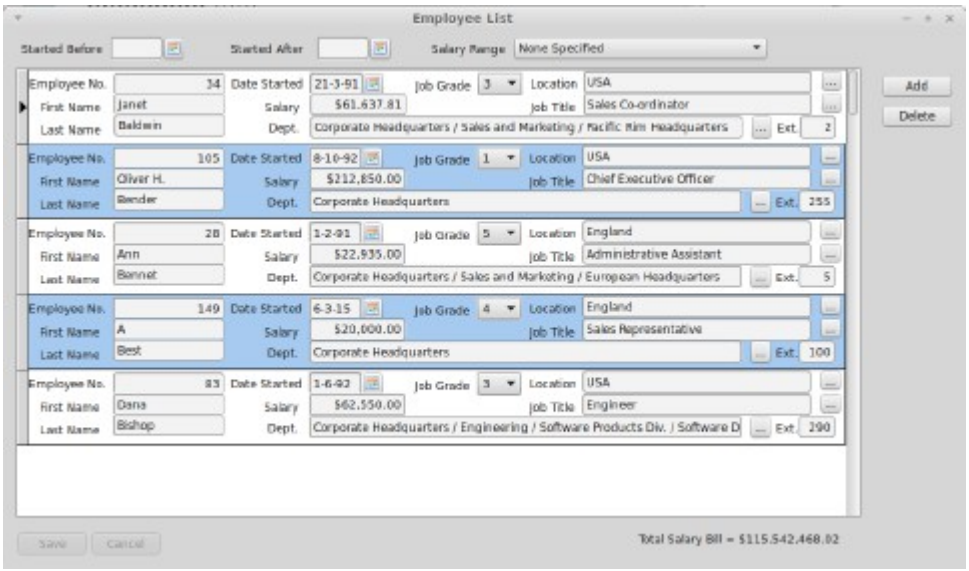


Рисунок 10: Пример TDBControlGrid

TDBControlGrid это двойник, а не клон компонента TDBCrtlGrid из Delphi. TDBControlGrid является таблицей из одного столбца, который повторяет TWinControl (обычно TPanel или TFrame) для каждой строки данных. Каждая строка соответствует строке набора данных, на который указывает свойство DataSource. Любые виджеты управления данными на дублируемых (к примеру) TPanel содержат значения для соответствующей строки.

В отличие от TDBCtrlGrid среды Delphi, здесь нет ограничений на элементы управления, которые можно использовать. В принципе, можно использовать любые визуальные элементы. Свойство "csReplicable" не применяется в TDBControlGrid. Однако, возможны проблемы с производительностью, если на панели слишком много элементов управления или имеются значительные задержки в отрисовке некоторых элементов.

Для использования нового компонента просто поместите его на форму и задайте ему нужные размеры. Затем отдельно поместите TPanel на ту же форму и наполните ее нужными элементами управления, обычно виджетами управления данными с тем же источником данных DataSource.

Теперь задайте свойству DrawPanel компонента TDBControlGrid ссылку на эту панель. При этом панель будет перемещена, став дочерним элементом TDBControlGrid заняв верх высотой в одну строку сетки. Высота строки будет установлена равной высоте панели, а ширина панели — равной ширине сетки. В любой момент панель можно отсоединить.

Теперь установите свойство DataSource компонента TDBControlGrid в то же значение, что у виджетов панели.

Важное замечание: Строго рекомендуется НЕ открывать набор данных, связанный с DBControlGrid в обработчике события формы "OnShow" . В случае использования GTK2 это может приводить к неправильной отрисовке строк при первом отображении. Если необходимо, используйте "Application.QueueAsyncCall" для задержки открытия набора данных (см. примеры DBControlGrid) пока создается окно формы. Смотрите примеры приложений.

Когда вы запускаете свое приложение и открываете набор данных, указанный в DataSource, TDBControlGrid должен показать панель для каждой строки набора и дочерние элементы каждой строки должны быть установлены в значения для этой строки.

Когда сетка получает фокус ввода, вы можете перемещаться по записям с использованием

стрелок вверх/вниз, Ctrl+Home и Ctrl+End перемещают в начало/конец набора данных. Для перехода между строками вы также можете использовать мышь. Нажатием стрелки вниз на последней строке вы добавите новую строку, если только не установлена опция "Disable Insert" .

Все строки можно редактировать на месте . Перемещение между строками автоматически фиксирует изменения. Для отмены изменений можно использовать клавишу "escape" .

Стереть строку можно с помощью метода Delete соответствующего набора данных.

Смотрите примеры программ в качестве руководства по использованию TDBControlGrid. Примеры требуют использования IBX и базы данных employee .

12.2.1 Свойства TDBControlGrid

Свойство	Тип	Назначение
DrawPanel	TWinControl	Этот элемент управления будет размножен для каждой строки набора данных. Обычно он является компонентом TPanel или TFrame.
Options	TPanelGridOptions	Аналогичны свойству TDBGrid, но ограничены значениями □ Cancel On Exit □ Disable Insert □ Show Indicator Column
DataSource	TDataSource	Как обычно, указывает набор данных.
DefaultPositionAtEnd	Boolean	Если равно true, то набор данных устанавливается на последнюю запись,

12.2.2 События TDBControlGrid

Событие	Описание
OnKeyDownHander	TDBControlGrid использует событие KeyDown для перехвата клавиш управления, если Draw Panel активен. Например, для использования клавиши «escape» для отмены редактирования. Для изменения поведения Панельного редактора вы можете написать собственный обработчик нажатия клавиш.

12.3 TIBTreeView

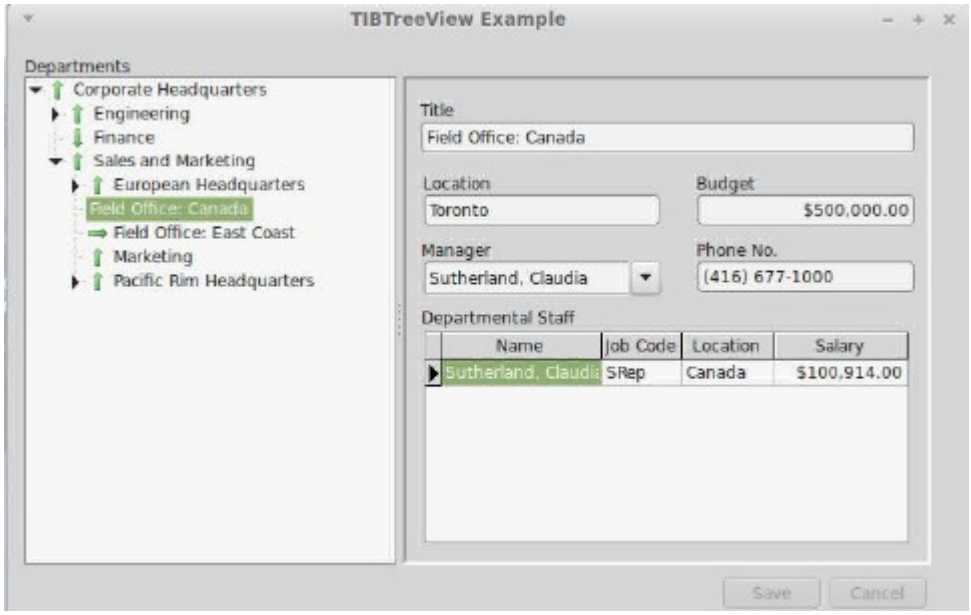


Рисунок 11: Пример компонента TIBTreeView Example

TIBTreeView является виджетом управления данными, производным от TCustomTreeView и используется для отображения иерархически организованных данных в виде дерева. Поддерживается вставка, удаление и изменение узлов дерева путем перемещения (например, перетаскиванием мышью) узлов из одного места дерева в другое. Указатель записи нижележащего набора данных всегда указывает на выбранный в данный момент узел. Таким образом, его можно использовать для выбора записи, подлежащей редактированию. Загрузка дерева производится с помощью серии SQL запросов.

Рисунок 11 взят из примера, находящегося в каталоге ibx/examples/ibtreeview и использующего базу данных "employee". Эта база данных включает иерархически организованную таблицу "DEPARTMENT", которая используется в этом примере.

Для использования компонента TIBTreeView, просто поместите его на форму, установите свойство DataSource и, как минимум, TextField, ParentField и KeyField как описано ниже.

Набор данных должен иметь простой главный ключ.

12.3.1 Свойства TIBTreeView

Свойство	Тип	Описание
DataSource	TDataSource	Источник данных для набора, представляемого в виде дерева
TextField	string	Имя поля для столбца, используемого для отображения названия узла
KeyField	string	Имя поля столбца, используемого для получения первичного ключа каждого узла
ParentField	string	Имя поля, используемого для определения главного ключа родительской записи. Для корневого элемента равно NULL.

Свойство	Тип	Описание
HasChildField	string	Необязательное. Имя поля, используемое для определения имеет ли узел дочерние узлы. Если присутствует, то должно возвращать ненулевое целое значение, если есть дочерние узлы.
RelationName	string	Необязательное. ChildField обычно результат связывания таблицы с собой и равно количеству дочерних строк. Однако, это может приводить к конфликту названий полей при построении запросов. Это поле должно содержать алиас для таблицы, используемый для ключевого, текстового и родительского поля (см. пример).

12.3.2 Методы TIBTreeView

function GetNodePath(Node: TTreeNode): TVariantArray

Возвращает массив вариантных значений, содержащий значения первичных ключей для узлов и их родителей от корневого и ниже.

function FindNode(KeyValuePath: array of variant; SelectNode: boolean): TIBTreeNode;

Возвращает TIBTreeNode задаваемую с помощью KeyValuePath. KeyValuePath является массивом содержащим список главных ключей, обходящих дерево от корневого и до требуемого узла. Если SelectNode равен true, требуемый узел делается выбранным.

Эту функцию можно использовать для выбора узла при помощи пути, возвращаемого функцией GetNodePath.

function FindNode(KeyValue: variant): TIBTreeNode;

Возвращает значение узла с главным ключом, указанным в KeyValue. Этот вызов принудительно загружает все дерево вызовом метода TCustomTreeView.FullExpand.

12.3.3 Перетаскивание

Перетаскивание поддерживается классом TCustomTreeView без необходимости дополнительных действий от TIBTreeView. В приводимых примерах для перетаскивания:

- DragMode устанавливается в automatic
- Событие OnDragOver обрабатывается при помощи:

```
procedure TForm1.IBTreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
State: TDragState; var Accept: Boolean);
begin
   Accept := Source = Sender
end;
```

- Событие OnDragDrop Event обрабатывается при помощи:

```
procedure TForm1.IBTreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
var Node: TTreeNode;
tv: TTreeView;
begin
   if Source = Sender then {Dragging within Tree View}
   begin
      tv := TTreeView(Sender);;
      Node := tv.GetNodeAt(X,Y); {Drop Point}
      if assigned(tv.Selected) and (tv.Selected <> Node) then
         begin
         if Node = nil then
            tv.Selected.MoveTo(nil,naAdd) {Move to Top Level}
         else
         begin
            if ssCtrl in GetKeyShiftState then
             begin
                Node.Expand(false);
                tv.Selected.MoveTo(Node,naAddChildFirst)
             end
          else
                tv.Selected.MoveTo(Node,naInsertBehind)
         end;
      end;
   end;
end;
```

Заметим, что в приведенном выше примере используется соглашение, что если в момент «отпускания» нажата клавиша «ctrl» , то узел добавляется в качестве дочернего. В противном случае он добавляется к предку.

12.4 TIBLookupComboEditBox

TIBLookupComboEditBox является потомком TDBLookupComboBox , реализующим «автодополнение» набираемого текста и «автодобавление» новых элементов.

- Автодополнение использует SQL операции для пересмотра списка выбора и ограничения его элементами имеющими уже набранный префикс (м.б. чувствительным к регистру или нет).
- Автодобавление позволяет добавлять вновь набранные элементы к списку выбора.

Хотя TDBLookupComboBox также поддерживает автодополнение, улучшение при использовании TIBLookupComboEditBox происходит при работе с длинными списками, так как набор нескольких символов резко сокращает список выбора.

Автодобавление обычно использует запрос на вставку записи для добавления новой строки и, в зависимости от обработчика "After Insert" позволяет задать для других полей строки соответствующие значения и/или генератор, назначенный набору данных .

12.4.1 Пример TIBLookupComboEditBox

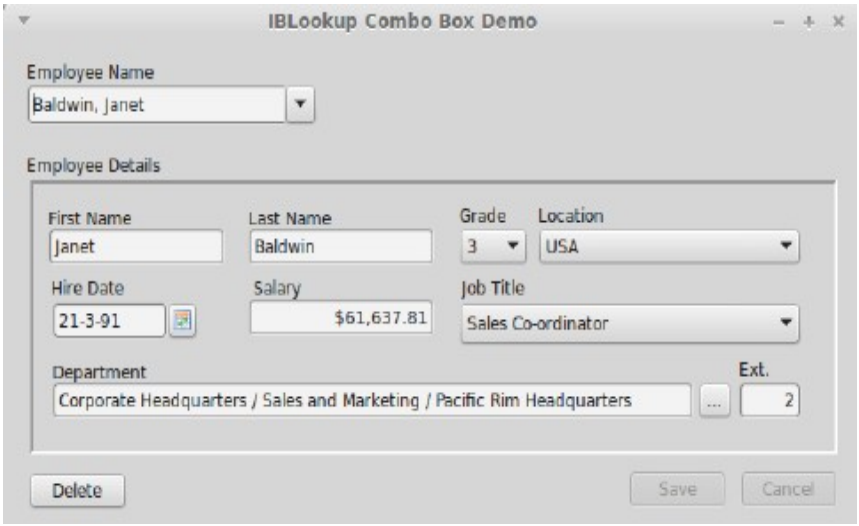


Рисунок 12 Использование TIBLookupComboEditBox

Изображенный выше пример использует базу данных "employee" и находится в каталоге ibx/examples/lookupcombobox . Виджет "Employee Name" представляет собой TIBLookupComboEditBox и используется в этом примере для:

- a) Выбора записи о сотруднике для редактирования
- b) Начала ввода новой записи о сотруднике.

Сначала следует изучить использование нового элемента управления. Нажмите на стрелочку выпадающего списка и раскроется список имен сотрудников (в виде Фамилия/Имя) .Обычно он длиннее, чем можно поместить на одном экране.

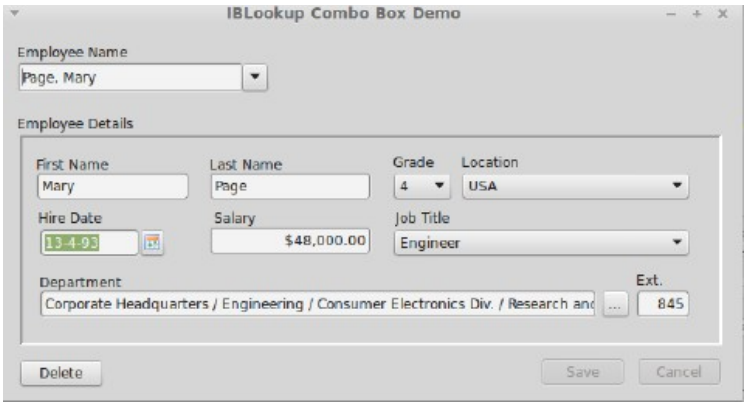


Рисунок 13: Выбор другого сотрудника

Теперь закройте список, выделите все символы в текстовом поле Employee Name и наберите на клавиатуре "pa". После короткой (600мс) задержки вид формы станет таким, как на рисунке 13, т. е. Для первого сотрудника, фамилия которого начинается на "pa", т.е. Mary Page.

Конечно, автодополнение может и не дать сразу сотрудника, которого вы хотели. Раскройте теперь список и вы увидите всех сотрудников, у который фамилия начинается на "pa". Этот список значительно короче, чем полный список и позволяет быстро выбрать нужного сотрудника.

На самом деле все это можно сделать используя только клавиатуру. Начните сначала и введите "pa", теперь нажмите стрелочку вниз, список раскроется и вы сможете перемещаться по списку сотрудников с фамилией на "pa". Нажмите Enter для выбора записи о сотруднике.

Если после ввода "pa" и увидев в поле Mary Page, нажать "r", расширив ввод до "par", то вы получите запись для Bil Parker.

Чтобы вернуть полный список просто нажмите клавишу escape.

12.4.1.1 Автодобавление

Автодобавление позволяет быстро вставлять записи о новых сотрудниках. Например, начните с выделения всего текста в текстовом поле Employee Name и введите имя нового сотрудника (например, Smith, John), и нажмите клавишу "Enter" . Вы получите запрос на потверждение добавления нового сотрудника:

Рисунок 14: Подтверждение добавления нового пользователя

Если вы нажмете на "yes", то будет создана новая запись о сотруднике, как видно ниже

Введенное имя будет разделено на имя и фамилию. Остальные поля получат значения по умолчанию в обработчике "OnInsert" . Теперь можно изменить значения по умолчанию .

12.4.2 Свойства TIBLookupComboEditBox

TIBLookupComboEditBox наследует свойства TDBLookupComboBox. Дополнительно он определяет следующие свойства:

Свойство	Тип	Пояснения
AutoInsert	Boolean	Установите в true, чтобы разрешить автодобавление
AutoComplete	Boolean	Установите в true (по умолчанию) чтобы разрешить автодополнение
KeyPressInterval	Integer	Задержка в миллисекундах между последним нажатием клавиши и автозавершением (по умолчанию 500ms).
RelationName	String	TIBLookupComboEditBox изменяет секцию "Where" в SQL запросе набора данных ListSource для того, чтобы переделать список и использует свойство

Свойство	Тип	Пояснения
		"ListField" в качестве имени столбца. Если это имя недопустимо в SQL операторе, то свойство "RelationName" надо установить в имя таблицы или ее алиас, чтобы уточнить имя столбца и устранить неоднозначность .

12.4.3 TIBLookupComboEditBox Event Handlers

Событие	Описание
OnAutoInsert	TIBLookupComboEditBox обычно использует оператор Insert для набора данных ListSource, чтобы сделать автодобавление. Если это невозможно, или нежелательно, то нужно создать обработчик события OnAutoInsert, который сделает необходимое добавление. Обработчик получает значение текста для вставки и должен вернуть новое значение ключа.
OnCanAutoInsert	Этот обработчик вызывается автоматически перед тем, как будет сделано автодобавление и обычно используется для подтверждения добавления и получения согласия пользователя (например, при помощи диалога). Обработчик получает текст для вставки и должен установить значение "Accept" в true для выполнения добавления, или false для отмены.

12.5 TIBArrayGrid

TIBArrayGrid является визуальным элементом, который можно связать с TIBArrayField и использовать для присмотра/редактирования содержимого одно или двумерных массивов полей Firebird. Он находится на закладке "Firebird Data Controls" палитры компонентов.

Для использования TIBArrayGrid, просто поместите его на форму и установите свойство DataSource в значение, соответствующее набору данных, а в свойстве DataField выберите поле, соответствующее массиву полей. Сетка данных автоматически примет размер, соответствующий размерности массива.

Заметим, что границы массива можно в любой момент обновить при помощи IDE, нажав на компонент правой кнопкой мыши и выбрав в контекстном меню пункт "Update Layout".

Во время выполнения TIBArrayGrid будет отображать значения элементов массива, соответствующие текущей записи для просмотра/редактирования. Если элемент массива равен null, то соответствующая ячейка пуста. В пустую ячейку можно записать новое значение. При фиксации (post) текущей записи будут сохранены и новые значения массива полей.

12.5.1 Свойства

Большая часть свойств TIBArrayGrid соответствует свойствам TStringGrid. Ниже приводится список новых свойств TIBArrayGrid. Заметим, что вы не должны явно устанавливать количество строк или столбцов, поскольку они должны соответствовать массиву полей.

Открытые свойства (public)

Свойство	Описание
ArrayIntf	Предоставляет прямой доступ к массиву.
DataSet	Набор данных связанный с DataSource (только для чтения).
Field	Поле — источник данных массива полей

Опубликованные свойства (published, доступны в инспекторе объектов):

Свойство	Описание
DataField	Имя поля, содержащего массив полей
DataSource	Источник данных
ReadOnly	Установите в значение true, чтобы запретить изменения
ColumnLabels	Список строк, который задает значения для каждого столбца данных. Если значение задано, то вверху таблицы создается строка заголовка.
ColumnLabelAlignment	Устанавливает выравнивание для заголовков столбцов
ColumnLabelFont	Устанавливает шрифт для заголовка столбцов
RowLabels	Список строк, который задает подписи для каждой строки в таблице данных. Если задана, то создается столбец из подписей в сетке данных.
RowLabelAlignment	Устанавливает выравнивание для подписей строк
RowLabelFont	Устанавливает шрифт для подписей строк
RowLabelColumnWidth	Ширина столбца для подписей строк.
TextAlignment	Задает выравнивание для элементов массива полей.

12.5.2 Примеры

Приводятся примеры приложений с использованием одномерного и двумерного массива полей. Для каждого случая приложение создает собственную базу данных и при первом запуске заполняет ее тестовыми данными. Заметим, что вам нужно запустить приложение, прежде чем пытаться обратится к свойствам базы в IDE. Это необходимо для создания базы данных.

12.5.2.1 Создание базы данных

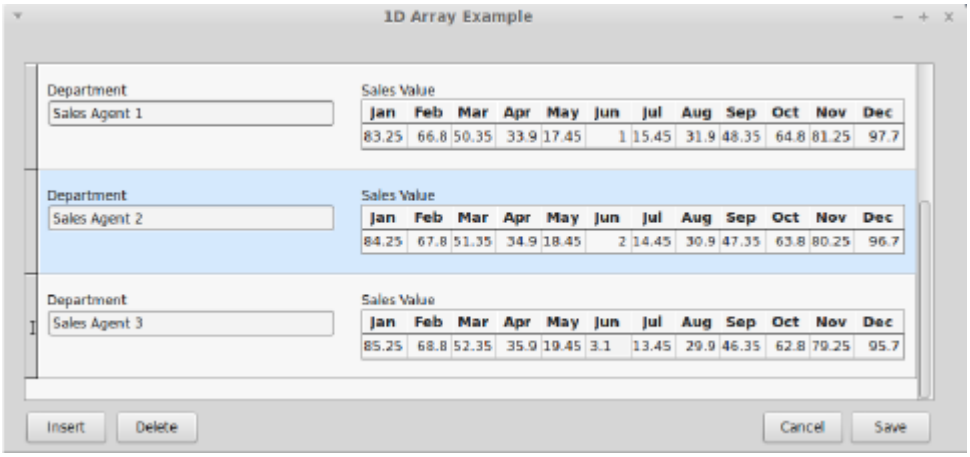
В обоих примерах свойство "CreateIfNotExists" компонента TIBDatabase установлено в true . Это означает, что если при запуске программы файл базы данных отсутствует, то будет сделана попытка создать базу данных. После ее создания используется обработчик события "OnCreateDatabase" для добавления таблиц в созданную базу и заполнения ее тестовыми данными. Далее приложение продолжит работу, как если бы база данных уже существовала.

По умолчанию база данных создается во временном каталоге. Такое поведение можно изменить, отредактировав модуль "unit1" , удалив директиву "{$DEFINE LOCALDATABASE}" и установив константу "sDatabaseName" в желаемое значение, например

```
const
sDatabaseName = 'myserver:/databases/test.fdb';
```

12.5.2.2 Приложение 1D Array

Ниже приводится скриншот приложения



В данном случае определение таблицы выглядит так

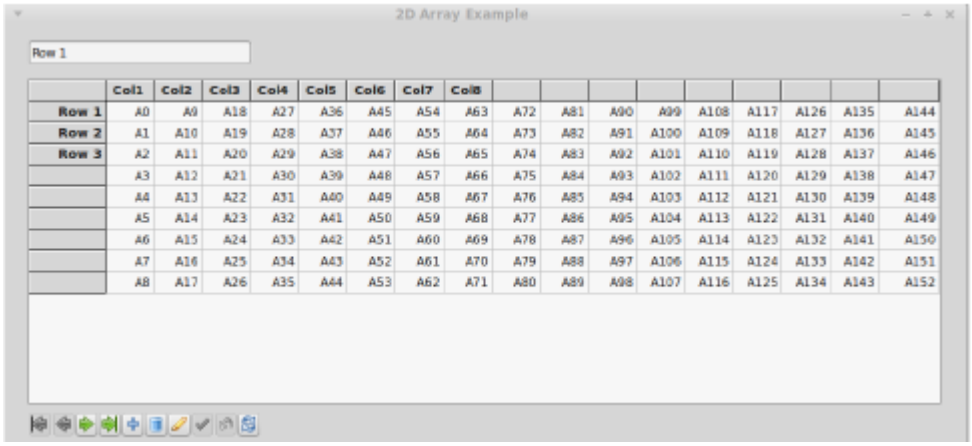
```
Create Table TestData
(RowID Integer not null,
Title VarChar(32) Character Set UTF8,
MyArray Double Precision [1:12],
Primary Key(RowID)
);
```

Каждая строка таблицы содержит массив чисел с плавающей точкой из 12 элементов. В примере приложения таблица показывается и редактируется с использованием компонента DBControlGrid. Поле title интерпретируется как "Department" и показывается с помощью элемента TDBEdit. Массив полей интерпретируется как объем продаж по месяцам и показывается с помощью одномерной сетки TIBArrayGrid содержащей заголовки столбцов. Пример позволяет менять названия отделов и объем продаж по месяцам, сохраняя результаты.. Можно вставлять и удалять записи в таблице TestData.

Заметим, что библиотека LCL содержит ошибку (http://bugs.freepascal.org/view.php?id=30892) , что является причиной неправильной работы одномерных массивов этого примера в ОС Windows. Ошибка состоит в том, что массив отображается только для выбранной строки. В приведенной ссылке с описанием проблемы есть патч LCL для исправления ошибки.

12.5.3 Пример 2D Array приложения

Ниже приводится скриншот программы



В этом случае таблица задана, как

```
Create Table TestData (
RowID Integer not null,
Title VarChar(32) Character Set UTF8,
MyArray VarChar(16) [0:16, -1:7] Character Set UTF8,
Primary Key(RowID)
);
```

Каждая запись содержит двумерный массив строк с индексами 0..16 и -1 to 7. В сетке данных первый индекс считается номером столбца, а второй - номером строки (типа. x,y декартовых координат).

В примере программы отображается одна запись набора с полосой навигатора для перемещения по записям, а также сохранения, отмены, редактирования, вставки и удаления. Пример показывает использование подписей к столбцам и строкам.