**«Компоненты Delphi для работы с СУБД HyTech»**

Оглавление

[*1.* *Установка компонент работы с СУБД HyTech в среду Delphi. Проверка работоспособности (запуск тестового приложения на тестовой БД)* 6](#_Toc302479363)

[*1.1.* Установка компонент работы с СУБД HyTech в среду Delphi 6](#_Toc302479364)

[1.2. Развертывание тестовой схемы БД 10](#_Toc302479365)

[1.3. Запуск примера приложения 12](#_Toc302479366)

[*2.* *Типовые приемы работы с СУБД* 13](#_Toc302479367)

[2.1. Введение в технологию работы с БД в Delphi 13](#_Toc302479368)

[2.2. Соединение и завершение работы с БД HyTech 16](#_Toc302479369)

[2.3. Выполнение запросов на выборку данных в БД и работа с результатами запросов 19](#_Toc302479370)

[*2.3.1.* *Перемещение по набору данных и получение значений полей текущей записи набора данных* 23](#_Toc302479371)

[*2.3.2.* *Работа с полями компонента THtQuery* 26](#_Toc302479372)

[*2.3.3.* *Использование параметров в запросе* 29](#_Toc302479373)

[*2.3.3.1.* *Использование параметров, созданных в процессе разработки приложения* 30](#_Toc302479374)

[*2.3.3.2.* *Использование создаваемых в процессе работы приложения параметров* 33](#_Toc302479375)

[*2.3.4.* *Редактирование набора данных* 36](#_Toc302479376)

[*2.3.5.* *Добавление новой строки в набор данных* 41](#_Toc302479377)

[*2.3.6.* *Удаление строки из набора данных* 43](#_Toc302479378)

[*2.3.7.* *Использование справочников для ввода данных* 46](#_Toc302479379)

[*2.3.7.1.* *Создание полей в наборе данных* 49](#_Toc302479380)

[*2.3.7.2.* *Изменение набора данных – добавление и редактирование строк таблицы* 54](#_Toc302479381)

[*2.3.8.* *Поиск строки по значению поля* 58](#_Toc302479382)

[2.4. Фильтрация информации в наборе данных 60](#_Toc302479383)

[2.5. Журналирование изменений набора данных в памяти 62](#_Toc302479384)

[2.5.1. Извлечение данных из БД и копирование их в таблицу в памяти 66](#_Toc302479385)

[2.5.2. Работа с набором данных в памяти (добавление, удаление, редактирование строк) 67](#_Toc302479386)

[2.5.3. Работа с журналом изменений 68](#_Toc302479387)

[2.5.4. Применение и отмена изменений 71](#_Toc302479388)

[2.6. Пакетная загрузка данных в таблицу HyTech 78](#_Toc302479389)

[2.6.1. Загрузка информации из файла во временную таблицу 81](#_Toc302479390)

[2.6.2. Перенос информации из временной в постоянную таблицу 83](#_Toc302479391)

[2.6.3. Принудительный перенос информации из буфера компонента htpump во временную таблицу 85](#_Toc302479392)

[2.6.4. Использование компонента – наследника TDataSet как источника информации для пакетной загрузки 87](#_Toc302479393)

[2.7. Экспорт данных 88](#_Toc302479394)

[2.8. Работа с макросами и препроцессором программного кода 91](#_Toc302479395)

[2.8.1. Работа с макросами 91](#_Toc302479396)

[2.8.2. Использование прагм 100](#_Toc302479397)

[2.9. Работа с курсорами 103](#_Toc302479398)

[2.9.1. Создание курсора, выполнение запроса, открытие результата, получение первичной порции данных и разрушение курсора 107](#_Toc302479399)

[2.9.2. Получение информации о курсоре и информации о таблице 116](#_Toc302479400)

[2.9.3. Чтение данных, поиск данных в курсоре и сортировка курсора 119](#_Toc302479401)

[2.9.4. Добавление, удаление, изменение записи в курсоре 125](#_Toc302479402)

[*3.* *Компоненты для визуализации* 131](#_Toc302479403)

[3.1. Работа с MS Excel 131](#_Toc302479404)

[3.2. Доступ к данным, организованным в виде дерева 139](#_Toc302479405)

[*4.* *Комплексный пример приложения, использующего компоненты* 150](#_Toc302479406)

[*4.1.* *Настройка СУБД для комплексного примера* 150](#_Toc302479407)

[*4.2.* *Конфигурирование среды разработки для запуска тестового приложения* 152](#_Toc302479408)

[*4.3.* *Описание приложения* 154](#_Toc302479409)

[*4.3.1.* *Обновление справочников* 156](#_Toc302479410)

[*4.3.2.* *Реестры* 157](#_Toc302479411)

[*4.3.3.* *Реестр «Кафедры»* 159](#_Toc302479412)

[*4.3.4.* *Реестр «Книги»* 161](#_Toc302479413)

[*4.3.5.* *Реестр «Студенты»* 170](#_Toc302479414)

[*4.3.6.* *Отчет «Студенты - должники»* 177](#_Toc302479415)

[*4.4.* *Демонстрация работы приложения* 181](#_Toc302479416)

[*5.* *Справочник по компонентам* 190](#_Toc302479417)

[*5.1.* *Закладка HyTech:* 192](#_Toc302479418)

[5.1.1. THtBase 192](#_Toc302479419)

[5.1.2. THtQuery 204](#_Toc302479420)

[5.1.3. THtStoredProc 218](#_Toc302479421)

[5.1.4. TCustomMemTable 223](#_Toc302479422)

[5.1.5. TMemTable 231](#_Toc302479423)

[5.1.6. TCustomSqlMem 234](#_Toc302479424)

[5.1.7. TSqlMem 236](#_Toc302479425)

[5.1.8. THtPump 242](#_Toc302479426)

[5.1.9. THtDsPump 246](#_Toc302479427)

[5.1.10. TDataMove 249](#_Toc302479428)

[5.1.11. TDBFMove 252](#_Toc302479429)

[5.1.12. TTXTMove 254](#_Toc302479430)

[5.1.13. THTDBFTable 255](#_Toc302479431)

[5.1.14. TMacList 261](#_Toc302479432)

[5.1.15. TMacrosList 262](#_Toc302479433)

[5.1.16. TSQLParser 264](#_Toc302479434)

[5.1.17. THtUpdateSQL 267](#_Toc302479435)

[*5.2.* *Закладка HtComp* 270](#_Toc302479436)

[5.2.1. TExcel 270](#_Toc302479437)

[5.2.2. THtxDbFilter 274](#_Toc302479438)

[*5.2.2.1.* *THtxCondition* 279](#_Toc302479439)

[5.2.3. TXTreeView 282](#_Toc302479440)

[5.2.4. THTree 293](#_Toc302479441)

[5.2.5. THTreeView 295](#_Toc302479442)

[*Приложение* 298](#_Toc302479443)

# Установка компонент работы с СУБД HyTech в среду Delphi. Проверка работоспособности (запуск тестового приложения на тестовой БД)

Перед началом работы необходимо установить и настроить СУБД HyTech. Подробно процесс развертывания сервера описан в 1 части курса в теме «Администрирование СУБД HyTech». В дальнейшем будем считать, что сервер установлен в директории с:\hytech, также будем предполагать, что htw - приложение, предназначенное для выполнения запросов и административных команд СУБД HyTech - установлено в директории с:\hytech\htw.

Кроме того, должна быть установлена среда разработки Delphi: Borland Delphi 7.0, Delphi 8, Delphi 2005(9), Delphi 2006 (10) или Delphi 2007 (11). Все примеры, использованные в курсе, разработаны с помощью среды «Borland® Delphi® for Microsoft® Windows™ Version 10» (Delphi 2006), однако при работе с курсом не обязательно использовать именно это приложение.

## **Установка компонент работы с СУБД HyTech в среду Delphi**

Для установки компонентов в среду разработки Delphi необходимо выполнить следующую последовательность шагов:

* установить клиент SVN (графический - <http://tortoisesvn.net/>, консольный <http://www.collab.net/downloads/subversion/>), если SVN клиент отсутствует;
* выбрать каталог, в который будут скачаны исходные коды компонентов, например это будет C:\HytTech\HTVCL. Тогда в переменную окружения PATH добавляем пути C:\HytTech\HTVCL и C:\HytTech\HTVCL\RTL;
* скачать в выбранный каталог (C:\HytTech\HTVCL) с помощью клиента SVN исходные коды компонентов из репозитария <https://my-svn.assembla.com/svn/ht-vcl/trunk>;
* в каталоге (C:\HytTech\HTVCL) сделать копию файла setBor.cmd с именем \_setBor.cmd, после чего отредактировать эту копию, указав правильные пути до всех версий Delphi, которые планируется использовать при разработке. Пути необходимо указывать в формате 8.3 (получить путь в формате 8.3 можно либо посредством команды dir с ключом /x или /n, либо, например, при помощи файлового менеджера Far, включив отображение коротких путей к файлу комбинацией клавиш CTRL+N);
* отредактировать файл dcc32.cfg в каталоге %DELPHI%\bin, а именно:
* в конец строки, начинающейся с «-u», добавить (через «;»)C:\HytTech\HTVCL\LIB***X***, где ***X*** – порядковый номер версии Delphi (для Delphi 7 – 7, Delphi 8 – 8, Delphi 2005 – 9, Delphi 2006 – 10, Delphi 2007 – 11);
* добавить строку -iC:\HytTech\HTVCL;
* в каталоге C:\HytTech\HTVCL в командной строке запустить файл MakeAll.cmd, передав ему в качестве единственного параметра номер версии Delphi (нумерация аналогична использующейся в предыдущем пункте). Помимо MakeAll.cmd можно запустить MakeExe.cmd с аналогичным параметром – тогда в каталоге C:\HytTech\HTVCL\RTL появятся программы для редактирования макросов и отладки SQL-запросов;
* установить компоненты в среду разработки – запустить среду разработки, в меню Component найти пункт «Install Packages», в появившемся меню нажать кнопку Add и выбрать каталог C:\HytTech\HTVCL\LIB*X* и добавить из него все файлы \*.bpl.

Проверим, что компоненты доступны для использования при разработке. Создадим новое приложение и новую форму, обратимся к панели инструментов среды разработки, удостоверимся, что на панели доступны две новые закладки – HtComp и HyTech (см. ).

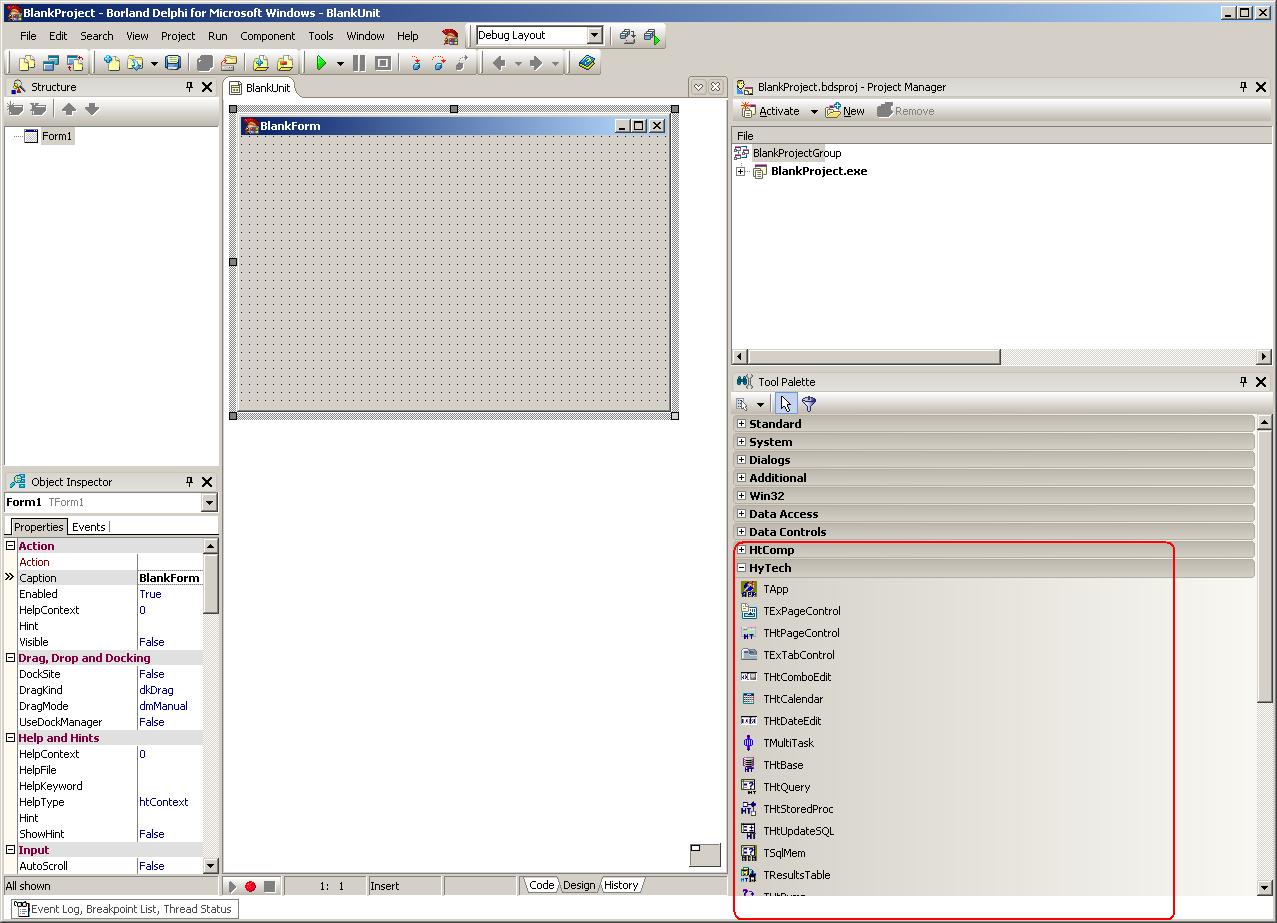


Рис. . Новые закладки на панели инструментов среды разработки

Список наиболее часто используемых компонентов, находящихся на закладках HtComp и HyTech приведен в .

Таб. . Компоненты для работы с HyTech

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Закладка | Название компонента | Иконка | Описание компонента |
|  | HyTech | THtBase |  | Предназначен для обеспечения связи с базой данных HyTech |
|  | HyTech | THtQuery |  | Обеспечивает доступ к данным в таблицах и представления БД HyTech, обладает кэш-буфером начитываемых записей |
|  | HyTech | THtStoredProc |  | Предназначен для выполнения произвольных запросов, не требующих чтения более одной строки результата |
|  | HyTech | THtUpdateSQL |  | Позволяет назначить произвольные модификационные SQL на THtDataSet |
|  | HyTech | TSqlMem |  | Предназначен для кэширования результатов запросов, при активизации создает THtQuery, выполняет в нем запрос, копирует в себя все данные и уничтожает Query, удобен для кэширования на клиенте мелких справочников и использования в качестве lookup DataSet позволяет назначить обработчики событий создаваемого компонентом THtQuery |
|  | HyTech | TMemTable |  | Представляет собой набор данных, который хранит в памяти записи, при этом вычисляемые поля хранятся в каждой записи. Компонентом поддерживается журналирование изменений |
|  | HyTec`h | TDBFMove |  | Предназначен выгрузки информации из БД Hytech в формат dbf на рабочей станции пользователя, при этом необходимо построить соответствия полей источника и полей пункта назначения, достоинством компонента является возможность использования выражений |
|  | HyTech | TTXTMove |  | Предназначен для выгрузки информации из DataSet в текстовый файл |
|  | HyTech | THTDBFTable |  | Является набором данных для БД в формате DBF, не использующим BDE |
|  | HyTech | TMacList |  | Предназначен для работы с макросами, хранит текст макросов в DFM файле |
|  | HyTech | TMacrosList |  | Предназначен для работы с макросами, реализует хранилище макросов как отдельный от DFM файл |
|  | HyTech | THtPump |  | Предназначен для буферизованного заполнения таблицы HyTech, подразумевает создание принимающей таблицы и запроса информации из неё |
|  | HyTech | THtDsPump |  | Является наследником THtPump, обладает несколькими дополнительными свойствами и методами для загрузки информации из различных наборов данных |
|  | HyTech | TSQLParser |  | Препроцессор SQL |
|  | HtComp | THtxDbFilter |  | Предназначен для обеспечения фильтрации набора данных |
|  | HtComp | TXTreeView |  | Предназначен для показа ветвящихся иерархических структур на основе информации из базы данных. |
|  | HtComp | THTree |  | Используется в паре с THTreeView, предназначен для хранения древовидной структуры узлов, кэширует информацию; при подключении нескольких THTreeView, обеспечивает их взаимодействие и идентичность отображаемой информации |
|  | HtComp | THTreeView |  | Используется в паре с THTree, предназначен для отображения узлов |
|  | HtComp | TExcel |  | Предназначен для выгрузки информации в Excel и загрузки её из Excel в БД |

## **Развертывание тестовой схемы БД**

При изучении приемов работы с HyTech будет использована тестовая схема БД. В качестве примера выбрана предметная область «Учет книг библиотечного фонда», ER-диаграмма которой представлена на - .



Рис. . Логическая схема БД



Рис. . Физическая схема БД

Для развертывания тестовой схемы необходимо:

1. скачать скрипт создания таблиц ([www.hytechdb.ru/testdb/scheme.sql](http://www.hytechdb.ru/testdb/scheme.sql)) и скрипт наполнения таблиц информацией ([www.hytechdb.ru/testdb/data.sql](http://www.hytechdb.ru/testdb/data.sql));
2. запустить htw.exe – приложение, используемое для выполнения пакетных и интерактивных запросов, а также административных команд БД HyTech;
3. установить соединение с настроенной ранее СУБД HyTech;
4. открыть и выполнить приведенные скрипты сначала создания таблиц, затем наполнения их данными. 7 новых таблиц должны появиться в дереве объектов слева.

Проверим, что указанная последовательность действий выполнена успешно: выберем информацию из таблицы AUTHOR посредством использования приложения htw. Результат работы запроса приведен на .

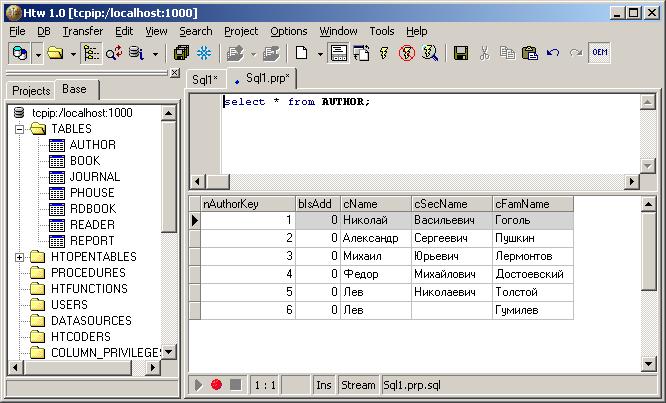


Рис. . Результат работы запроса

## **Запуск примера приложения**

Для запуска примера откроем в среде разработки тестовое приложение (исходные коды проекта располагаются по адресу [www.hytechdb.ru/sources/TestConnectionProj.RAR](http://www.hytechdb.ru/sources/TestConnectionProj.RAR)). Внешний вид главной формы при открытии её в среде разработки представлен на .

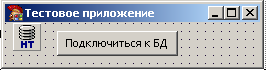


Рис. . Тестовое приложение (среда разработки)

Скомпилируем и запустим приложение. В случае, если всё установлено и настроено верно, при нажатии на кнопку возникнет диалог для внесения реквизитов подключения к HyTech. После внесения реквизитов и установки соединения с БД, заголовок формы изменится. Внешний вид приложения после успешного соединения с БД представлен на

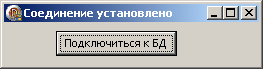


Рис. . Тестовое приложение (после успешного соединения с БД)

Соединение с базой данных будет разорвано автоматически при закрытии приложения. Более подробно вопросы соединения с БД и отключения от неё рассмотрены в следующем разделе.

# Типовые приемы работы с СУБД

## **Введение в технологию работы с БД в Delphi**

В основе технологии разработки приложений Delphi, работающих с СУБД, лежат следующие идеи:

* работа с СУБД из клиентских приложений в значительной части является типовой и может быть реализована вне специфики конкретной СУБД. Например, типичные клиентские приложения выполняют запросы к БД, получают результаты запросов, работают с результатами (например, визуализируют), вносят изменения, удаляют, добавляют данные. При этом сами запросы можно описать на языке SQL, а с результатами можно работать как с таблицами в памяти, вне зависимости от того, как они получены (строки таблицы – выбранные в результате запроса записи, колонки – значения полей в данной записи). Возможность типовой работы вне специфики СУБД в полной мере реализована в Delphi. Delphi предлагается набор базовых компонент которые эту типовую работу обеспечивают;
* должна быть обеспечена возможность учесть специфику конкретной СУБД, в тех случаях, когда это требуется, но без потери универсальности типовой работы с БД. В Delphi такие возможности реализуются путем использования наборов специальных компонент для каждой СУБД. Эти специальные компоненты либо являются расширением базовых компонент (механизм наследования классов), либо создаются как независимые. Заметим, что набор компонент для СУБД HyTech содержит преимущественно компоненты, расширяющие базовые, что позволяет для разработчиков, имеющих опыт создания приложений Delphi для других СУБД использовать при работе с HyTech в приложениях Delphi сходные приемы работы.

Базовые компоненты для реализации технологии работы с БД представлены следующими группами:

* компоненты для доступа к данным;
* визуальные компоненты, реализующие интерфейс пользователя;
* компоненты для связи источников данных с визуальными компонентами, предоставляющими интерфейс пользователя;

Основным базовым компонентом, предназначенным для обеспечения доступа к данным из таблиц БД, является ***TDataSet*** - это базовый компонент для всех специальных компонентов доступа к данным БД.

Компонент TDataSet  обеспечивает доступ к последовательности записей, которые читаются из некоторого источника данных, хранятся во внутреннем буфере (для повышения производительности) и, возможно, модифицируются пользователем (в этом случае изменения записываются обратно в место постоянного хранения данных). Компонент посредством одного из своих свойств связан с компонентом, обеспечивающим доступ к БД.

Основными функциями компонента являются:

* выполнение запроса к БД. Для выполнения запроса, его текст необходимо разместить в одном из полей компонента, кроме того в запросе могут быть использованы параметры, способ передачи параметров является унифицированным для всех СУБД;
* осуществление навигации по набору данных - набор данных, получаемый в результате выполнения запроса, представляет собой таблицу, строками которой являются записи таблиц БД, столбцами которого являются атрибуты таблиц, компонент предоставляет методы для перемещения на строчку вперед, на строчку назад, на первую строку, на последнюю, поддерживает признаки начала и окончания набора данных;
* редактирование информации в памяти приложения с последующим применением изменений в памяти базы данных, поддерживаются операции вставки, редактирования, удаления; в общем случае запросы на соответствующее действие (например, редактирование) создаются автоматически, однако такое поведение компонента становится неприемлемым, если в запросе используется соединение двух и более таблиц, в такой ситуации для редактирования набора данных необходимо использовать специальный компонент TUpdateSql, который также может применяться для реализации возможности редактирования набора данных, созданного с доступом "только для чтения". Компонент TUpdateSql в соответствующих полях хранит скрипты удаления, редактирования и вставки, которые выполняются при инициировании соответствующего действия в TDataSet.

Также компонент TDataSet поддерживает и некоторые другие функции, например, возможность поиска по результирующему набору или фильтрация.

Компонент ***TDataSource*** реализует связь между компонентами - наборами данных и элементами управления, используемыми для отображения данных. При построении отношений между таблицами "родительская – дочерняя" компонент "источник данных" служит для связывания наборов данных, указывая родительский набор данных, обеспечивая при этом возможность сложной навигации между компонентами. Компонент посредством одного из своих свойств связан с компонентом TDataSet.

При конструировании формы невизуальные компоненты, используемые для доступа к данным, такие как DataSource, DataSet, размещаются на форме, но при выполнении приложения эти компоненты не видны, поэтому их можно размещать в любом удобном месте формы.

Визуальные компоненты, реализующие интерфейс пользователя, представлены на закладке Data Controls палитры компонентов. Многие из этих компонентов аналогичны элементам управления закладке Standard с тем лишь отличием, что связаны через источник данных (компонент типа TDataSource) с определенным полем (или полями) из набора данных (компонентов типа  TDataSet или их наследников). Библиотека VCL [предоставляет](javascript://) ряд компонентов для управления данными, наиболее часто используемые представлены в .

Таб. Компоненты для управления данными

|  |  |  |
| --- | --- | --- |
| № | Название компонента | Описание компонента |
|  | TDBGrid | Осуществляет отображение записей набора данных в виде таблицы и управление ими |
|  | TDBNavigator | Предоставляет средства навигации по набору данных, возможности добавления новых записей, включения режима редактирования, присвоения и отмены сделанных изменений |
|  | TDBEdit | Реализует работу с однострочным полем редактирования |
|  | TDBMemo | Реализует работу с многострочным полем редактирования, в котором можно отображать и изменять значение поля набора данных |
|  | TDBCheckBox | Реализует работу с логическим компонентом ("флажок"), который связан с полем базы данных. |
|  | TDBListBox | Реализует работу с компонентом "список", который используется для отображения значений поля набора данных |
|  | TDBComboBox | Реализует работу с компонентом "раскрывающийся список", который используется для отображения значений поля набора данных |
|  | TDBLookupListBox | Позволяет выполнять просмотр списка, заполненного значениями полей из другого набора данных |

Компоненты, предназначенные для работы с БД HyTech, являются, в большинстве случаев, расширением стандартных компонентов и во многом повторяют их поведение, однако при этом имеют ряд особенностей, которые мы рассмотрим в дальнейшем.

## **Соединение и завершение работы с БД HyTech**

Первым шагом при работе с БД HyTech из приложения на Delphi является установка соединения с базой данных. Для доступа к базе данных HyTech можно воспользоваться стандартными ODBC компонентами, однако, чтобы использовать в полной мере возможности СУБД HyTech следует работать с набором компонент, специально разработанных для СУБД с HyTech и учитывающих ее специфические особенности. В дальнейшем типовые приемы работы с СУБД HyTech будут описаны именно для этого набора компонент.

Итак, для установки связи с базой данных используется компонент типа THTBase, который располагается на закладке HyTech.

Компонент THTBase предоставляет несколько методов для установки соединения:

* Connect – не принимает параметров. Используется в случае, когда в реквизиты подключения уже заданы ранее или будут заданы в окне регистрации;
* ConnectAs – принимает три параметра – путь к базе данных, имя пользователя, пароль;
* Login – также принимает три параметра – имя пользователя, пароль и необязательный параметр ShowDialog, по умолчанию равный false, влияющий на показ окна регистрации. Реквизиты подключения к базе данных процедура считывает из ini-файла sql64.ini. располагающегося в каталоге, в который установлена СУБД (в данном случае это с:\hytech).

Компонент THTBase, таким образом, предоставляет возможность указать программно реквизиты подключения (адрес БД, имя пользователя и пароль) или внести реквизиты в момент инициирования подключения посредством диалога.

Соединение с БД рассмотрим на следующем примере. Создадим новое приложение, на главную форму поместим компонент THTBase (присвоим ему имя htb), находящийся на закладке HyTech, компонент типа TLabel (присвоим ему имя lbStatus),находящийся на закладке Standard и кнопку также с закладки Standard (присвоим имя bConnect) «Подключиться к БД». Пусть при нажатии на кнопку будет устанавливаться связь с базой данных, а в метку помещаться информация о версии СУБД, с которой установлено соединение. Для установки соединения с БД воспользуемся методом Connect, позволив пользователю ввести реквизиты БД при подключении. В данном случае необходимо свойству ConnectUser компонента ТHTBase необходимо присвоить значение “true”. Процесс подключения пользователя к БД HyTech состоит из двух этапов – установка соединения с БД и авторизация. Свойство htb.ConnectUser показывает, необходимо ли запускать процедуру авторизации после процедуры установки соединения.

Отключение от базы данных осуществляется методом Disconnect. Разместим на форме ещё одну кнопку (присвоим ей имя bDisconnect) «Отключиться от БД». Внешний вид формы в процессе разработки представлен на .

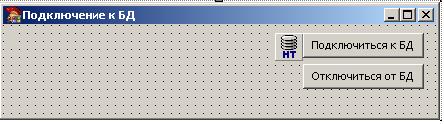


Рис. . Внешний вид формы (среда разработки)

Код обработчиков нажатия на кнопки «Подключиться к БД» и «Отключиться от БД» приведен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bConnectClick(Sender: TObject);  begin  htb.Connect;  if htb.Active then  lbStatus.Caption := 'Соединение установлено. Версия сервера '+  htb.StrVersion;  end;  procedure TConnectForm.bDisconnectClick(Sender: TObject);  begin  htb.Disconnect;  if not Htb.Active then  lbStatus.Caption := 'База данных не доступна';  end; |

После запуска приложения и нажатия на кнопку «Подключиться к БД» инициируется процесс установки соединения, внешний вид приложения приведен на - .

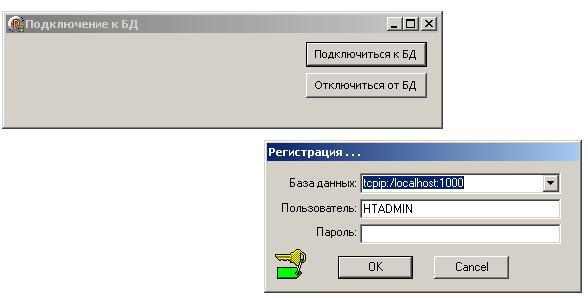


Рис. . Диалог подключения к БД HyTech

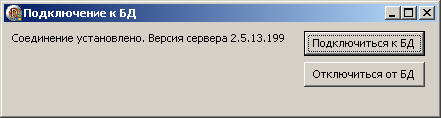


Рис. . Внешний вид приложения после успешного соединения с БД

Результат работы приложения после нажатия на кнопку «Отключиться от БД» приведен на .



Рис. . Внешний вид приложения после отключения от БД

Заметим, что компонент THtBase используется не только для подключения/отключения БД, другие его возможности будут приведены по мере необходимости при рассмотрении других типовых приемов работы с БД. Подробная информация о методах компонента, его свойствах и событиях приведена в разделе 5.

## **Выполнение запросов на выборку данных в БД и работа с результатами запросов**

Для выполнения запросов на выборку данных в БД и работы с результатами усложним пример из предыдущего пункта следующим образом.

Поместим на главную форму компоненты THtQuery с закладки HyTech (присвоим компоненту имя htq), TDataSource с закладки DataAccess (присвоим имя ds), TDbGrid с закладки DataControls (присвоим имя grd). Внешний вид формы в процессе разработки представлен на .



grd (TDbGrid)

ds (TDataSource)

htq (THtQuery)

htb (THTBase)

Рис. Внешний вид формы (среда разработки)

Компонент htq будет использован для выполнения запроса к БД и работы с результатами, компонент grd для визуализации результатов запроса в виде таблицы, а компонент ds будет действовать как посредник между компонентами TDataSet (THtQuery является его наследником) и компонентами Data Controls (здесь – THtDbGrid). Свойства компонента htq представлены на .

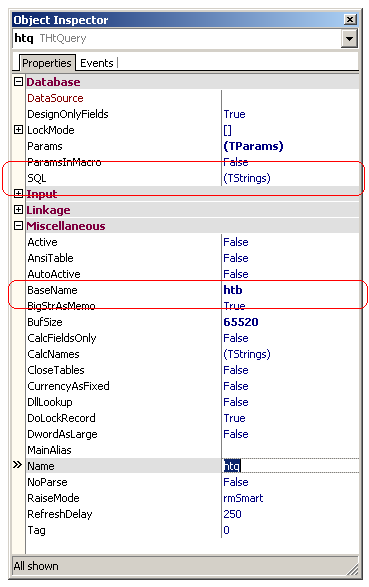


Рис. . Свойства компонента htq

Итак, для выполнения запроса в свойстве BaseName компонента htq указываем ссылку на созданный ранее компонент htb, текст запроса разместим в свойстве SQL того же компонента. Текст запроса приведен в .

Листинг

|  |
| --- |
| **select** b.cName cName,  a.cFamName cAuthorFam,  substr(a.cName,0,1) cAuthorName  **from** book b, author a  **where** b.nAuthorKey=a.nAuthorKey |

В компоненте ds указываем ссылку на htq, в grd – на ds. Свойства компонентов ds и grd представлены на -.

|  |  |
| --- | --- |
|  |  |
| Рис. . Свойства компонента ds | Рис. . Свойства компонента grd |

Поместим на форму ещё одну кнопку с закладки Standard (назовем её bSelect), при нажатии на которую будет выполняться запрос к базе данных. Обработчик события нажатия на кнопку представлен в

Листинг

|  |
| --- |
| procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  htq.Active := true;  end; |

Приложение готово для запуска. После компиляции и запуска приложения необходимо нажать на кнопку «Подключиться к БД» для установки соединения с базой данных, а затем кнопку «Выполнить запрос». Результаты работы приложения после нажатия на кнопку «Выполнить запрос» продемонстрированы на .

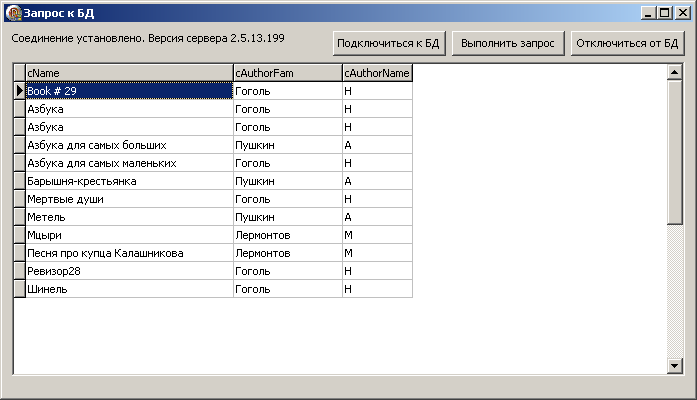


Рис. . Внешний вид приложения после нажатия на кнопку «Выполнить запрос»

Рассмотрим типовые приемы работы с набором данных.

## Перемещение по набору данных и получение значений полей текущей записи набора данных

Как и стандартный компонент TDataSet, компонент THtQuery поддерживает методы навигации по результирующему набору строк, а именно:

* переместиться на первую запись – метод First;
* переместиться на последнюю запись – метод Last;
* переместиться на следующую, относительно текущей, запись – метод Next;
* переместиться на предыдущую, относительно текущей, запись – метод Prior;

Номер текущей строки компонент хранит в свойстве RecNo.

Рассмотрим пример работы с методами навигации . На форму, рассмотренную в предыдущем разделе, поместим ряд компонентов: панель (назовем её pNavigate) с закладки Standard, на панель поместим четыре кнопки с той же закладки, назвовем их- bFirst (метка «Перв»), bPrior (метка «Пред»), bNex (метка «След»), bLast( метка «Последн»), добавим на форму также компонент TDbEdit (назовем его dbeBookName) с закладки Data Controls. Внешний вид формы представлен на .

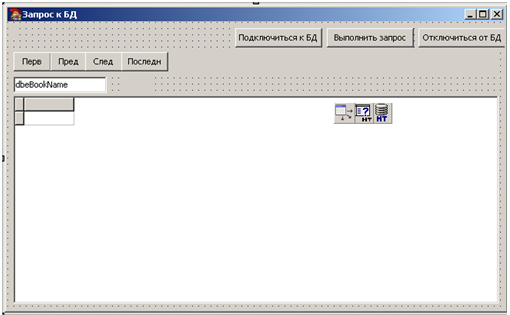


Рис. . Внешний вид формы (среда разработки)

При нажатии на кнопки будет осуществляться перемещение на первую, предыдущую относительно текущей, следующую относительно текущей, последнюю строки соответственно. При обработке нажатия необходимо проверять, находится ли компонент в активном состоянии, то есть выполнен ли запрос (если компонент находится в активном состоянии, свойству htq.Active присвоено значение true). Очевидно, что навигация по набору данных не имеет смысла, если запрос не вернул ни одной записи, проверять количество записей будем путем обращения к свойству htq.RecordCount. При перемещении на следующую или предыдущую запись также необходимо проверять, не достигнут ли конец или начало списка. Признак того, что текущая строка является первой (и перемещение на предыдущую не имеет смысла), содержится в свойстве htq.BoF, признак того, что текущая строка является последней (и перемещение на следующую строку не имеет смысла), содержится в свойстве htq.EoF. Код обработчиков нажатия на кнопки и процедуры проверки представлен в .

Листинг 4

|  |
| --- |
| function TConnectForm.NavigationCheck : boolean;  begin  if not htq.Active then  begin  Application.MessageBox('Необходмио выполнить запрос','Внимание');  result := false;  exit;  end;  if htq.RecordCount = 0 then  begin  Application.MessageBox('Результирующий набор содержит 0 строк',  'Внимание');  result := false;  exit;  end;  result := true;  end;  procedure TConnectForm.bFirstClick(Sender: TObject);  begin  if not NavigationCheck then  exit;  htq.First;  end;  procedure TConnectForm.bLastClick(Sender: TObject);  begin  if not NavigationCheck then  exit;  htq.Last;  end;  procedure TConnectForm.bNextClick(Sender: TObject);  begin  if not NavigationCheck then  exit;  if not htq.Eof then  htq.Next;  end;  procedure TConnectForm.bPriorClick(Sender: TObject);  begin  if not NavigationCheck then  exit;  if not htq.Bof then  htq.Prior;  end; |

Помимо кнопок для навигации набора данных мы добавили также компонент dbeBookName, который будет использоваться для отображения значения поля «названия книги», которое содержится в текущей строке набора данных, свяжем его(dbeBookName) с компонентом ds (TDataSource), указав ссылку на ds в свойстве dbeBookName.DataSource. Однако для отображения именно названия книги в свойстве компонента DataField необходимо указать соответствующее поле компонента htq. В нашем примере поля компонента htq создаются в момент его активизации, поэтому проставлять свойство dbeBookName.DataField необходимо программно при выполнении запроса, в связи с чем в конце обработчика нажатия на кнопку «Выполнить запрос» допишем строку, приведенную в .

Листинг

|  |
| --- |
| dbeBookName.DataField := htq.FindField('cName').FieldName; |

Метод компонента htq.FindField используется для поиска ссылки на объект «поле» по указанному имени. Имя найденного поля необходимо присвоить свойству dbeBookName.DataField.

Добавим на форму ещё один компонент типа TLabel (назовем его lbAuthorFam) с закладки Standard, будем заполнять его значением поля «Фамилия автора» по событию изменения позиции указателя текущей строки компонента htq (событие AfterScroll). Код обработчика события представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.htqAfterScroll(DataSet: TDataSet);  begin  lbAuthorFam.Caption := htq.FindField('cAuthorFam').Value;  end; |

Внешний вид приложения после соединения с базой данных и выполнения запроса представлен на . Не трудно заметить, что в компоненте dbeBookName содержится название книги, находящееся в текущей строке компонента набора данных, являющегося результатом запроса.

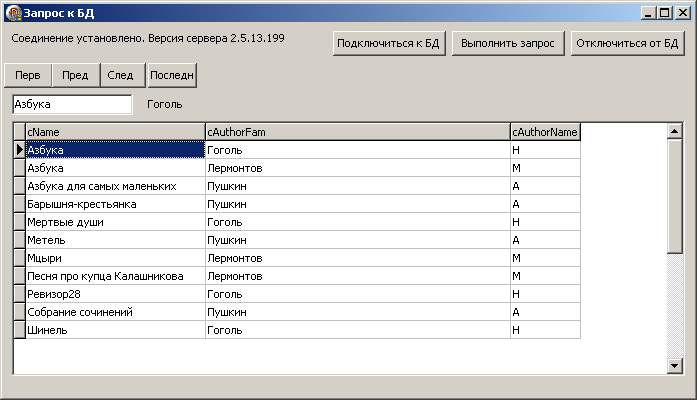


Рис. . Внешний вид приложения после выполнения запроса

Внешний вид приложения после нажатия на кнопку «След», «Послед» приведен на - .

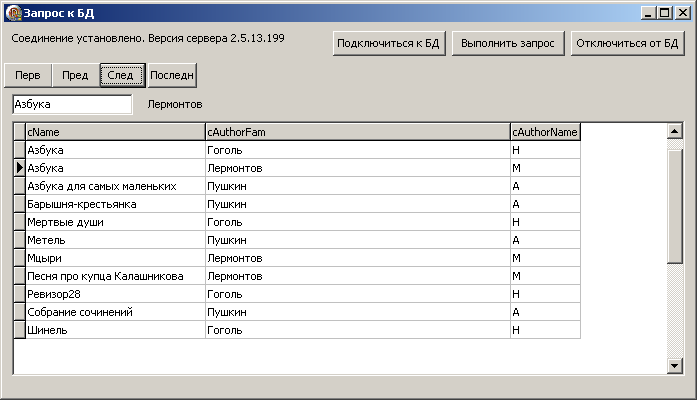


Рис. Внешний вид приложения после нажатия на кнопку «След»

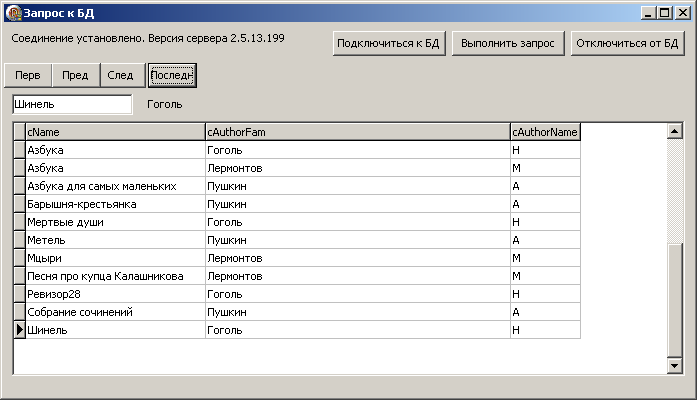


Рис. . Внешний вид приложения после нажатия на кнопку «Последн»

## Работа с полями компонента THtQuery

В предыдущем примере компонент htq создавал поля автоматически при открытии, однако список полей можно указать вручную путем редактирования свойства Fields, компоненту при этом можно указать, надо ли создавать автоматически недостающие поля в момент активизации, если таковые будут присутствовать в запросе, в стандартном компоненте TDataSet необходимость создавать поля автоматически отражается в свойстве DefaultFields. Компонент THtQuery имеет похожее свойство THtQuery.DesignOnlyFields - если THtQuery в неактивном состоянии имеет какие-либо поля (TField) и DesignOnlyFields истинно, то никакие больше поля при активизации не создадутся. Если изменить значение свойства DesignOnlyFields  на false, то после выполнения запроса у DataSet будут созданы TField для всех отобранных столбцов. Приведем пример. Создадим одно поле (TField) у имеющегося компонента htq для атрибута «название книги», текст запроса при этом изменять не будем. Создать новое поле можно нажав правой кнопкой мыши на узел htq->Fields в окне дерева объектов и выбрав пункт меню «New Field». Процесс добавления поля представлен на - .

|  |  |
| --- | --- |
|  |  |
| Рис. Добавление нового поля в дереве объектов | Рис. Добавление нового поля |

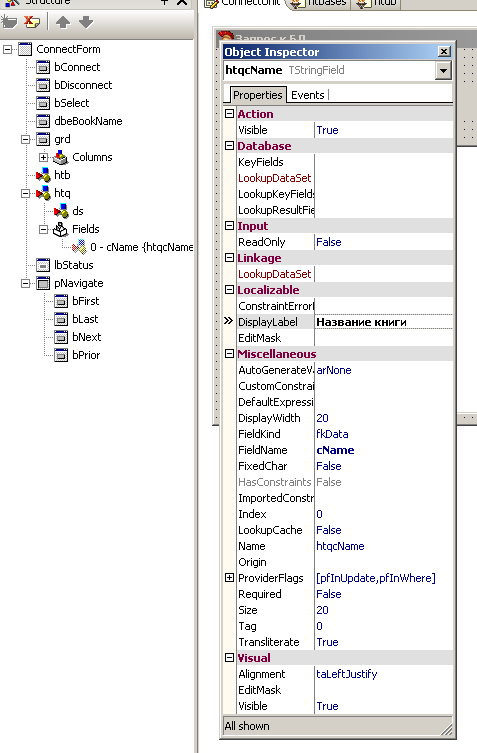


Рис. Свойства добавленного поля

Так как DesignOnlyFields компонента htq истинно, никакие другие поля при активизации компонента htq созданы не будут. Вследствие того, что относительно предыдущего примера состав полей компонента htq был изменен, значение поля cAuthorFam временно не нужно присваивать метке lbAuthorFam, поэтому ссылку на процедуру htqAfterScroll из обработчика события AfterScroll компонента необходимо удалить.

Запрос выполняется при нажатии на кнопку «Выполнить запрос», внешний вид формы после выполнения запроса приведен на .

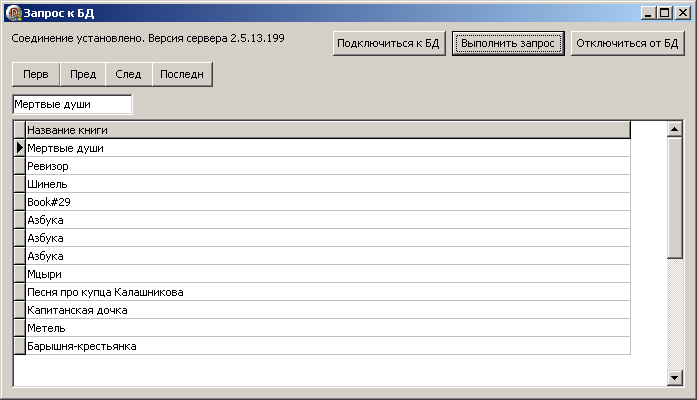


Рис. . Внешний вид приложения после нажатия на кнопку «Выполнить запрос»

Присвоим DesignOnlyFields значение false, вновь назначим обработчик событию htq.AfterScroll и выполним запрос ещё раз, внешний вид формы после выполнения запроса приведен на .



Рис. . Внешний вид приложения после нажатия на кнопку «Выполнить запрос»

## Использование параметров в запросе

Рассмотрим возможность использования параметров в запросе.

В тексте SQL запроса параметры задаются с двоеточием, предшествующим имени параметра. В процессе разработки при создании параметра можно задавать такие его свойства как Name, ParamType (тип параметра), Value.

Программно присвоить значения параметрам можно воспользовавшись такими свойствами, как TParam. Value, TParam. AsString, TParam. AsInteger. Получить доступ к параметрам в процессе выполнения приложения можно либо по номеру параметра в свойстве Params соответствующего компонента (например, TQuery.Params), при этом последовательность, в которой располагаются параметры в свойстве Params, определяется последовательностью их упоминания в запросе SQL, либо воспользовавшись методом ParamByName.

Аналогично стандартному компоненту TQuery, компонент THtQuery также позволяет работать с параметризованными запросами, однако имеет ряд особенностей, а именно:

* параметры подставляются непосредственно в текст SQL (свойство htq.SQL.Text) перед передачей его на исполнение, поэтому параметром может быть любая часть запроса, в том числе список таблиц, из которых производится выборка, часть условия ‘where’ итд;
* параметры типа string имеют два режима подстановки. Режим определяется типом параметра (ParamType): если ParamType = ptInput, то значение подставляется непосредственно в текст запроса, при этом значение параметра не заключается в кавычки (таким образом, параметром может быть часть SQL выражения). Если ParamType <> ptInput, то значение параметра перед подстановкой приводится к виду, понятному SQL интерпретатору, для чего :
  + все символы младше char(32) (пробел) преобразуются к виду "\xHH", где HH - шестнадцатиричный код символа;
  + все символы ' и " заменяются парой \' и \";
  + длинная строка разделяется на несколько подстрок по 64..70 символов каждая (интерпретатор не имеет возможности работать со строковыми константами, длина которых больше 80 символов), каждая подстрока заключается в двойные кавычки и обрамляется символами CR/LF.

Для хранения параметров в компоненте THtQuery используется свойство Params стандартного типа TParams. Также компонент обладает свойством ParamCount, хранящим количество параметров, и рядом методов для работы с параметрами.

Параметры запроса могут быть созданы в процессе разработки приложения ( а значения им присвоены в процессе работы приложения), либо созданы в процессе работы приложения. Первый способ рассмотрен в разделе , второй - .

## Использование параметров, созданных в процессе разработки приложения

На форму, рассмотренную в разделе , поместим компонент TEdit (назовем его eAuthorName) с закладки. Внешний вид формы представлен на . Это поле будет использоваться для указания значения параметра запроса.

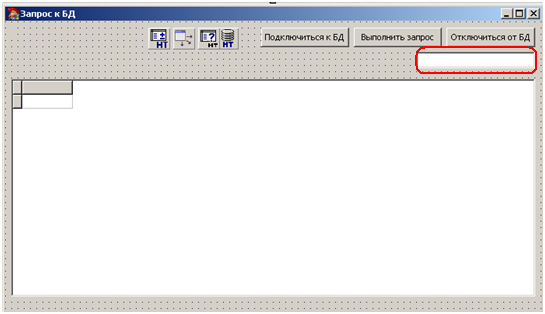


Рис. . Внешний вид формы (среда разработки)

Разместим в свойстве htq.SQL текст запроса, представленный в .

Листинг

|  |
| --- |
| select b.cName cName,  a.cFamName cAuthorFam , substr(a.cName,0,1) cAuthorName  from book b, author a  where b.nAuthorKey = a.nAuthorKey  and a.cFamName = :pFamily  order by a.cFamName |

Обратите внимание на то, что в запросе использован параметр pFamily, в запросе ему обязательно предшествует двоеточие (не отделяемое пробелом от имени параметра).

Изменим также код обработчика события нажатия на кнопку bSelect на код, приведенный в .

Листинг

|  |
| --- |
| procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if eAuthorName.Text <> '' then  htq.ParamByName('pFamily').AsString := eAuthorName.Text  else  htq.ParamByName('pFamily').AsString := '';  htq.Requery;  end; |

В данном случае значение параметра задается установкой свойства AsString – параметр предназначен для хранения строкового значения.

Метод htq.Requery – аналог пары методов Close/Open, он закрывает курсор, затем выполняет SQL запрос, при этом считается, что результат в курсоре имеет тот же тип и структуру: информация о курсоре не запрашивается, и поля не пересоздаются (т.е. запрос должен изменяться только в части WHERE, HAVING или ORDER BY), что дает некоторую экономию времени.

В случае, если в текстовое поле не внесено какое-либо значение, запрос не вернет ни одной строки. Результат работы приложения представлен на .



Рис. . Внешний вид приложения после выполнения запроса с пустым значением параметра

Внесем в поле eAuthorName значение «Пушкин» и нажмем на кнопку «Выполнить запрос». Результат работы приложения представлен на .

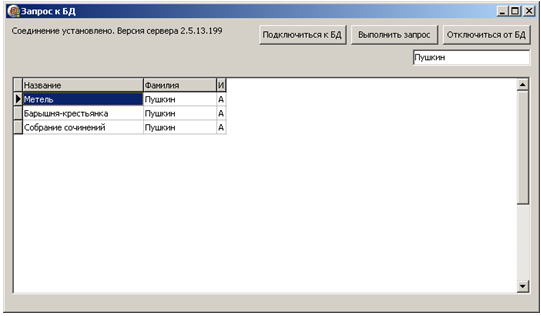


Рис. . Внешний вид приложения после выполнения запроса с непустым значением параметра

## Использование создаваемых в процессе работы приложения параметров

Поместим на форму, рассмотренную в предыдущем примере, компонент TLabel (назовем его lQueryText ) с закладки Standard. Присвоим свойству lQueryText.Caption значение “select b.сName. b.nIssYear from book b :pQueryTail” – этот текст в нашем примере будет использоваться как неизменяемая часть запроса, в котором будет использоваться параметр :pQueryTail. Также немного увеличим длину компонента eAuthorName, в который будем помещать значение параметра.

Внешний вид формы представлен на .

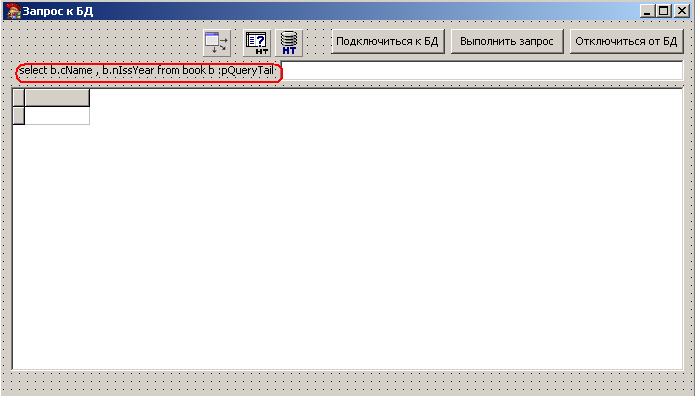


Рис. . Внешний вид формы (среда разработки)

Изменим код обработчика нажатия на кнопку bSelect на код, приведенный в .

Листинг

|  |
| --- |
| procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if htq.Active then  begin  htq.Params.Clear;  htq.Close;  end;  htq.SQL.Text := lQueryText.Caption;  if eAuthorName.Text <> '' then  begin  htq.ParamByName('pQueryTail').ParamType := ptInput;  htq.ParamByName('pQueryTail').AsString :=' '+ eAuthorName.Text;  end;  htq.Open;  end; |

Перед изменением текста запроса компонент htq необходимо сделать неактивным и удалить все ранее созданные параметры (это делается методом Clear). После присвоения нового текста запроса свойству htq.SQL.Text, компонент автоматически создает новый параметр pQueryTail, однако по умолчанию его типом становится ptUnknown, поэтому если в компонент eAuthorName вносится какой-то текст (после запуска приложения) для использования его в качестве значения параметра, тип параметра необходимо изменить вручную на ptInput для того, чтобы значение параметра подставлялось в текст запроса «как есть», не заключенное при этом в кавычки. Значение параметра, также как и в предыдущем примере, записывается в свойство AsString.

В данном случае мы не можем использовать метод htq.Requery, так как заранее известно только количество возвращаемых атрибутов, тогда как список таблиц, к которым идёт обращение, может изменяться.

Внешний вид приложения после соединения с базой данных, внесения текста в компонент eAuthorName и выполнения запроса представлен на - .

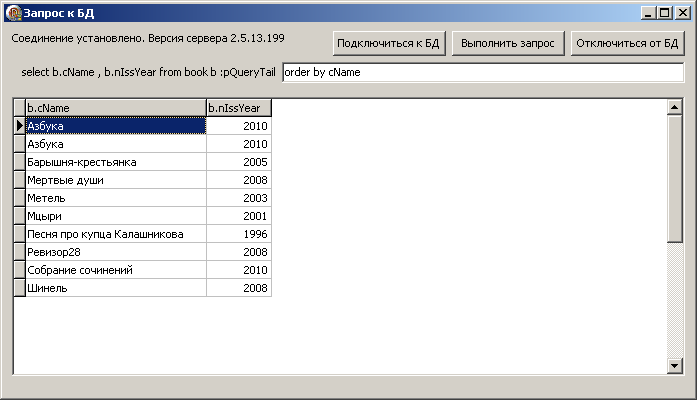


Рис. . Внешний вид приложения после изменения eAuthorName и нажатия на кнопку «Выполнить запрос»

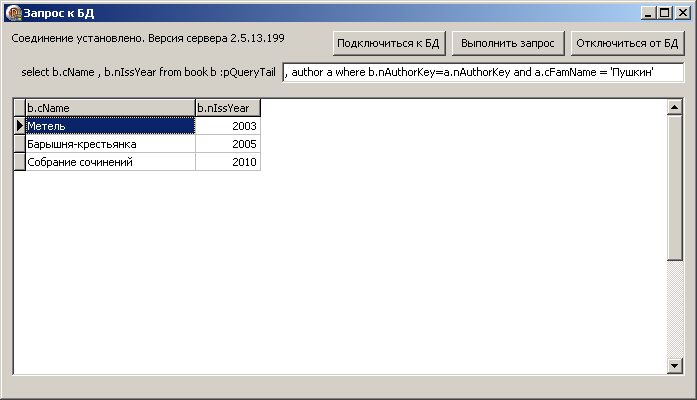


Рис. . Внешний вид приложения после изменения eAuthorName и нажатия на кнопку «Выполнить запрос»

## Редактирование набора данных

THtQuery допускает редактирование записей, если выполнены три условия:

1. свойству THtQuery.ReadOnly присвоено значение false;
2. результат сортирован или результат был выбран для изменения (for update);
3. результат был получен по одной таблице или назначены MainAlias и UpdateObject.

Расширим пример из раздела возможностями редактирования. Для удовлетворения требований 1-3 изменим ряд свойств компонента htq.

Добавим на форму компонент THtUpdateSQL (назовем его htupd) с закладки HyTech (подробное описание компонента см ниже). В свойстве htq.UpdateObject укажем ссылку на компонент htupd, также изменим код запроса, находящийся в свойстве htq.SQL, для того, чтобы набор данных был доступен для изменения.

Код, размещенный в свойстве htq.SQL , приведен в .

Листинг

|  |
| --- |
| fix all;  select b.cName cName, b.nBookKey nBookKey, b.recno recno,  a.cFamName cAuthorFam,  substr(a.cName,0,1) cAuthorName, a.nAuthorKey nAuthorKey  from book b, author a  where b.nAuthorKey=a.nAuthorKey  order by b.cName |

Создадим поля в компоненте htq, соответствующие атрибутам, использованным в запросе, и изменим свойство Visible на false для поляей recno, nAuthorKey и nBookKey, в противном случае в компоненте grd будут также отображены их значения, которые являются малоинформативными для пользователя. Свойству DataField компонента dbeBookName присвоим значение «cName».

В случае, если в запросе используется соединение нескольких таблиц, алиас таблицы, запись которой захватывается в начале редактирования должен быть обязательно указан в свойстве THtQuery.MainAlias(под алиасом таблицы понимается то, что пишется после имени таблицы в sql запросе в конструкции from, если алиас не упомянут, то он совпадает с именем таблицы. Так для таблицы book в коде запроса, приведенного в , алисом является «b», для таблицы author алиасом является «а». Таким образом свойству htq.MainAlias необходимо присвоить значение «b».

Вернемся к компоненту htupd. Его предназначение - модификация информации в БД с помощью заранее подготовленных SQL-предложений. Он позволяет определить различные операторы SQL для удаления, вставки и модификации записи, в том числе отличные от простых операторов delete, insert, update. Эти операторы SQL содержатся в свойствах DeleteSQL, InsertSQL, ModifySQL соответственно. Компонент выполняет SQL выражения при операциях THtQuery.Post и THtQuery.Delete, возвращая результат выполнения компоненту THtQuery. Напрямую методы THtUpdateSQL выполнять не следует, так как все необходимые действия производятся соответствующим компонентом THtQuery.

В свойстве компонента htupd.ModifySQL необходимо разместить код редактирования основной таблицы набора данных. Код, размещенный в свойстве htupd.ModifySQL, приведен в .

Листинг

|  |
| --- |
| var err;  fix book; // обновим текущий снимок  begin work b1 table book; // объявим начало транзакции по таблице book  /\*  В запросе используется две внешние, по отношению к скрипту, переменные – :recno и :cName.  recno – встроенная переменная, содержащая номер текущей строки в наборе данных, она позволяет однозначно идентифицировать соответствующую запись в БД HyTech в таблице book.  cName соответствует текущему значению поля с таким именем в набору данных.  Связь между переменной скрипта :cName и значением поля в наборе данных обеспечивается автоматически.  Обновим значение cName в таблице в соответствии с текущем значением поля cName набора данных  \*/  update book b  set cName = :cName  where recno =:recno;  // проверим наличие ошибки модификации  if(@err = lastsqlerr()){  usersend(-1 , 'ошибка модификации ['+@err+'] '+ sqlermsg(@err));  rollback work b1; // откатим транзакцию в случае ошибки  retcode(@err);  quit;  } ;  commit work b1;  // проверим наличие ошибки завершения транзакции  if(@err = lastsqlerr()){  usersend(-1 , 'ошибка завершения транзакции ['+@err+'] '+ sqlermsg(@err));  rollback work b1; // откатим транзакцию в случае ошибки  retcode(@err);  quit;  } ;  // установим retcode, равный нулю в случае отсутствия ошибок  retcode(0);  fix book;  select b.cName cName, b.nBookKey nBookKey, b.recno recno,  a.cFamName cAuthorFam,  substr(a.cName,0,1) cAuthorName, a.nAuthorKey nAuthorKey  from book b, author a  where b.recno= :b.recno and  b.nAuthorKey=a.nAuthorKey; |

Комментарии к листингу даны в коде, однако внесем несколько пояснений: оператор select в конце выражения обязателен и должен осуществлять то же соединение таблиц и с теми же алиасами, что и основной оператор select, по сути, он должен быть копией основного оператора select за исключением того, что в выражении where должно быть дополнительное условие для того, чтобы была отобрана одна и только одна строка. Это условие (b.recno= :b.recno).лучше ставить первым в выражении where: при выполнении SQL множество записей сразу же ограничится только одной записью из каждой таблицы, что кардинально сократит время на их связывание (одна из самых длительных операций). Для успешного выполнения изменения данных свойство htupd.SelRecSQL должно содержать оператор select, отбирающий одну и только одну строку для того, чтобы не выполнять исходный запрос целиком при изменении строки в таблице БД (это может занять большое количество времени) и не позиционировать в наборе данных указатель на текущую строку. Код, размещенный в свойстве htupd. SelRecSQL , приведен в .

Листинг

|  |
| --- |
| select b.cName cName, b.nBookKey nBookKey, b.recno recno,  a.cFamName cAuthorFam,  substr(a.cName,0,1) cAuthorName, a.nAuthorKey nAuthorKey  from book b, author a  where b.recno= :b.recno  and b.nAuthorKey=a.nAuthorKey; |

Так как в качестве основной таблицы мы выбрали таблицу «book», то в результирующем наборе данных мы сможем изменять только поле «Название книги».

Добавим в панель навигации ещё одну кнопку (назовем её bPost ) с палитры Standard. При нажатии на кнопку будем осуществлять применение изменений в БД. Внешний вид формы приведен на .

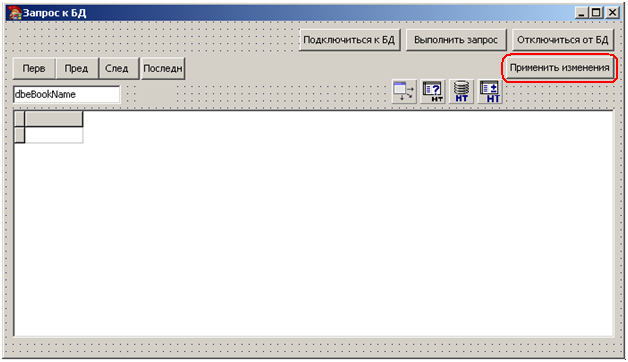


Рис. . Внешний вид формы (среда разработки)

Код обработчика события нажатия на кнопку bPost приведен в . Там же приведен код обработчика нажатия на кнопку bSelect.

Листинг

|  |
| --- |
| procedure TConnectForm.bPostClick(Sender: TObject);  begin  if not htq.Active then  exit;  htq.Post;  end;  procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if htq.Active then  htq.Close;  htq.Open;  end; |

Вместе с применением изменений идёт автоматическое обновление результирующего набора данных.

Скомпилируем и запустим приложение, установим связь с БД и выполним запрос. Изменим название книги с «Азбука» на «Азбука для самых маленьких» в компоненте dbeBookName.

Нажмем кнопку «Применить изменения». Результат работы приложения приведен на –.

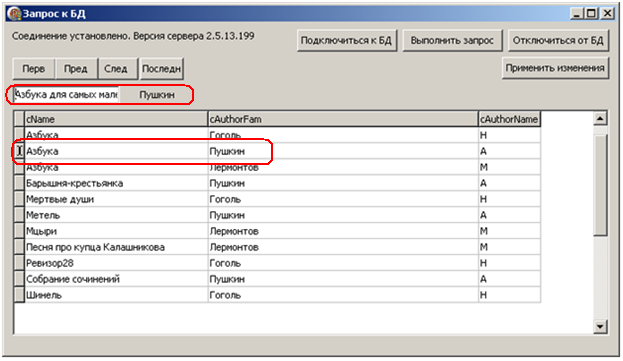


Рис. . Внешний вид приложения в процессе редактирования записи

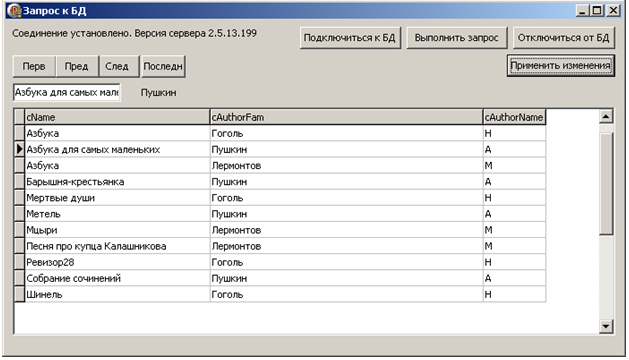


Рис. . Внешний вид приложения после нажатия на кнопку «Применить изменения»

## Добавление новой строки в набор данных

Добавим на форму кнопку «Добавить» (назовем её bAdd) с закладки Standard. Кнопка будет предназначена для добавления информации в таблицу. Внешний вид формы приведен на .

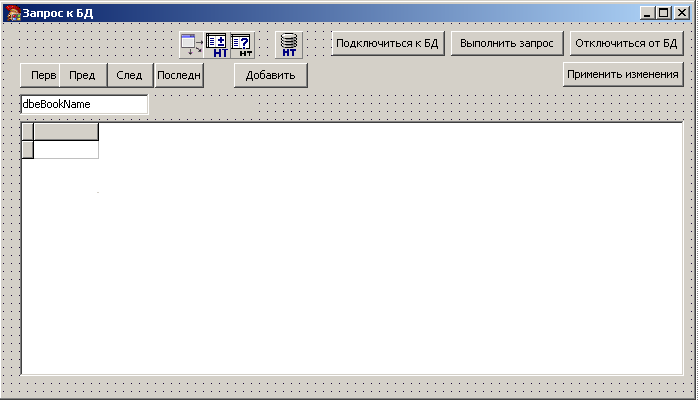


Рис. . Внешний вид формы (среда разработки)

В качестве основной таблицы выбрана таблица book, единственным атрибутом, возвращаемым из этой таблицы является «Название книги», то есть cName, поэтому при добавлении информации в компоненте grd мы будем заполнять только колонку с названием книги. Код обработчика нажатия на кнопку представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bAddClick(Sender: TObject);  begin  htq.Insert;  end; |

Метод Insert компонента htq используется для добавления информации в таблицу.

Для выполнения операции вставки информации необходимо изменить свойство компонента htupdt.InsertSQL. Код, который необходимо поместить в это свойство, приведен в .

Листинг

|  |
| --- |
| var err, rn;  begin work a\_book table book;  insert into book(cName, nAuthorKey)  values(:cName, 6);  @rn = recnu(); // recno добавленной записи  if(@err = lastsqlerr()){  usersend(-1 , 'ошибка добавления ['+@err+'] '+ sqlermsg(@err));  rollback work a\_book;  retcode(@err);  quit;  }  commit work a\_book;  if(@err = lastsqlerr()){  usersend(-1 , 'ошибка завершения транзакции ['+@err+'] '+ sqlermsg(@err));  rollback work a\_book;  retcode(@err);  quit;  }  retcode(@rn);  fix all;  select b.recno recno  from book b, author a  where b.recno = @rn and a.nAuthorKey = b.nAuthorKey;; |

Значения для полей таблицы берутся из полей htq. Запрос обязательно должен вернуть recno вновь добавленной записи (функция SQL recnu()) или код ошибки, по аналогии с запросом на изменение, оператор select в конце выражения обязателен и должен осуществлять то же соединение таблиц и с теми же алиасами, что и основной оператор select.

При написании текста запроса полезно иметь ввиду следующее:

* перед каждым "select" необходимо ставить "fix <имя\_таблицы>" (или fix all), иначе изменения могут быть не видны;
* в операторе select лучше указывать физические номера записей таблиц (recno), это необязательно, но если recno не указан, то при операциях модификации будут выполняться дополнительные обращения к серверу для определения этих номеров, что будет замедлять работу;
* если в SQL запросе нужно использовать переменные SQL ( например, как в , их лучше записывать как @VAR1, а не :VAR1, иначе компонент-наследник HtDataSet будет считать, что :VAR1 -это параметр запроса.

Внешний вид приложения после соединения с базой данных и нажатия на кнопку «Добавить» представлен на , внешний вид приложения после нажатия на кнопку «Применить изменения» представлен на .

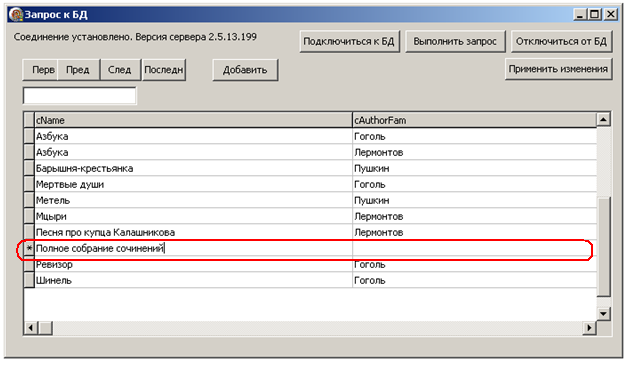


Рис. . Внешний вид приложения после нажатия на кнопку «Добавить»

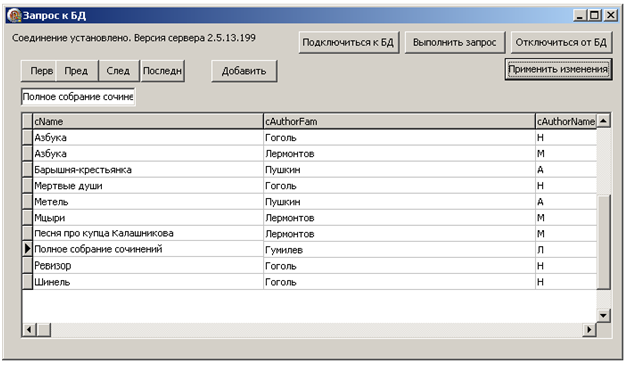


Рис. . Внешний вид приложения после нажатия на кнопку «Применить изменения»

## Удаление строки из набора данных

Добавим на форму кнопку «Удалить» (назовем её bDelete) с закладки Standard. Кнопка будет предназначена для удаления информации из таблицы book. Внешний вид формы приведен на .

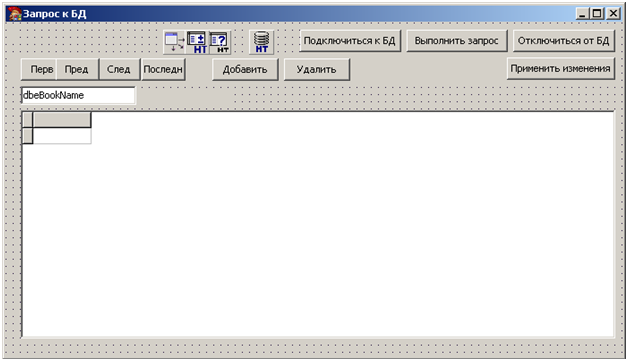


Рис. . Внешний вид формы (среда разработки)

Код обработчика нажатия на кнопку представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bDeleteClick(Sender: TObject);  begin  htq.Delete;  end; |

Метод Delete компонента htq используется для удаления информации из таблиц базы данных.

Для выполнения операции удаления необходимо изменить свойство компонента htupdt.DeleteSQL. Код, который необходимо поместить в это свойство, приведен в .

Листинг

|  |
| --- |
| var err;  delete from book  where recno = :recno;  if(@err = lastsqlerr())  usersend(-1 , ’Ошибка удаления [’+@err+’] ’+ sqlermsg(@err));  retcode(@err); |

В запросе выполняется удаление записи с номером :recno, который берется из основного курсора, затем проверяются ошибки. Запрос обязательно должен вернуть код ошибки или нуль.

Внешний вид приложения после соединения с базой данных, до и после нажатия на кнопку «Удалить» представлен на - .

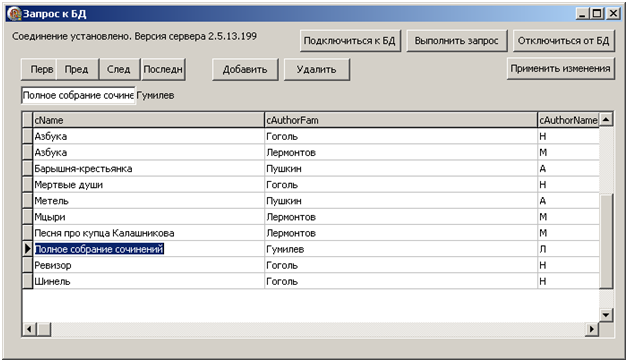


Рис. . Внешний вид приложения до нажатия на кнопку «Удалить»

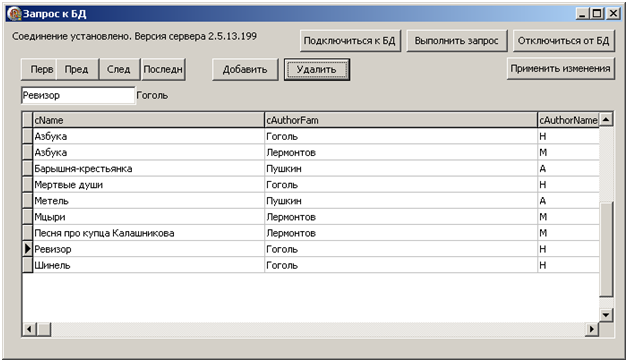
**

Рис. . Внешний вид приложения после нажатия на кнопку «Удалить»

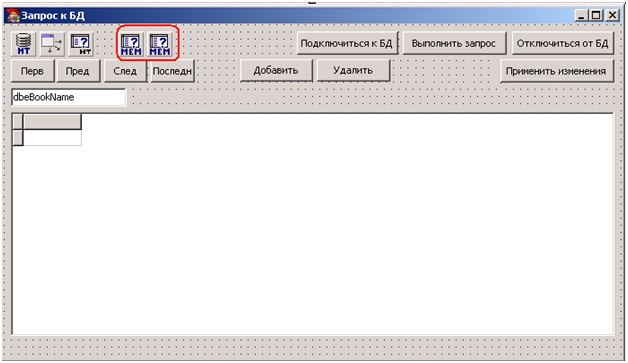
## Использование справочников для ввода данных

Обычно справочник представляет собой таблицу, первичный ключ которой является внешним ключом в одной или нескольких других таблицах базы данных. Для ввода значений справочника обычно используется выпадающий список (LookUp), из которого выбирается элемент (отображаемое поле), а значение элемента (ключевое поле) присваивается полю основного набора данных. Для редактирования/добавления строк таблицы, ссылающейся на справочник, обычно используется два варианта форм ввода: табличная форма ввода и форма ввода единственной строки. Технику работы со справочниками рассмотрим на примере ввода через табличную форму ввода, для этого модифицируем пример из раздела . Добавим на главную форму ряд компонентов, ряд компонентов удалим и некоторым изменим поведение. Список всех необходимых компонентов формы представлен в таблице .

Таб. . Компоненты главной формы примера работы со справочниками

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Название | Тип | Закладка | Статус | Назначение |
|  | htb | THtBase | HyTech | Без изменений | Осуществление связи с БД |
|  | htq | THtQuery | HyTech | Изменены значения ряда свойств | Формирование основного набора данных |
|  | htsqlAuthor | TSQLMem | HyTech | Добавлен | Формирование справочного набора данных для наполнения LookUp- поля набора данных htq информацией об авторах |
|  | htsqlPHouse | TSQLMem | HyTech | Добавлен | Формирование справочного набора данных для наполнения LookUp - поля набора данных htq информацией об издательствах |
|  | ds | TDataSource | DataAccess | Без изменений | Установка связи компонента htq с grd |
|  | grd | TDbGrid | DataControls | Без изменений | Отображение набора данных htq |
|  | bConnect | TButton | Standard | Без изменений | Установка связи с БД |
|  | bDisconnect | TButton | Standard | Без изменений | Отключение от БД |
|  | bSelect | TButton | Standard | Изменен обработчик | Активация компонента htq. Первичное наполнение компонента данными |
|  | bAdd | TButton | Standard | Без изменений | Перевод компонента htq в режим добавления информации |
|  | bDelete | TButton | Standard | Без изменений | Удаление текущей строки набора данных htq |
|  | bPost | TButton | Standard | Без изменений | Применение изменений |
|  | dbeBookName | TDBEdit | DataControls | Без изменений | Отображение названия книги |
|  | lbAuthorFam | TButton | Standard | Без изменений | Отображение фамилии автора |
| 15. | lbStatus | TLabel | Standard | Без изменений | Отображение информации о СУБД в случае успешного подключения |

Внешний вид формы в среде проектирования приведен на . Красным цветом на рисунке отмечены добавленные компоненты.

****

**Рис. 40. Внешний вид формы (среда разработки)**

Для организации справочников будем использовать два компонента – htsqlAuthor(TSQLMem) и htsqlPHouse (TSQLMem). Особенностью компонента TSQLMem является следующее: при активизации компонент выполняет запрос к БД, сохраняет результат запроса (набор данных) в свой буфер, после чего ему более не требуются ресурсы базы данных. Вместо TSQLMem можно воспользоваться компонентом-наследником TDataSet (например, THtQuery), но при этом следует позаботиться об освобождении ресурсов БД самостоятельно.

Изменим код обработчика нажатия на кнопку bSelect и код обработчика события htq.AfterScroll. Исходный код приведен в . Комментарии к коду приведены в разделе .

Листинг

|  |
| --- |
| // Активизация основного и справочных наборов данных  procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if htq.Active then  htq.Close;  htsqlPHouse.Open;  htsqlAuthor.Open;  htq.Open;  dbeBookName.DataField := htq.FindField('cName').FieldName;  end;  // присвоение метке значения поля «Автор» по событию изменения текущей строки  procedure TConnectForm.htqAfterScroll(DataSet: TDataSet);  begin  lbAuthorFam.Caption := vartostr(htq.FindField('cFamName').Value);  end; |

## Создание полей в наборе данных

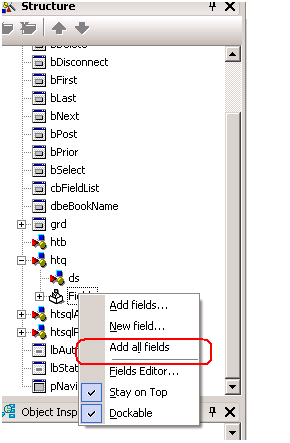
Изменим текст запроса в свойстве htq.SQL в соответствии с .

Листинг

|  |
| --- |
| **fix all;**  **select** nBookKey, nAuthorKey , nPHouseKey, cName , nIssYear  **from**  book  **order by** cName |

В данном запросе внешними ключами, ссылающимися на справочники, являются атрибуты nPHouseKey и nAuthorKey, соответственно справочниками будут являться таблицы phouse и author.

Найдем в дереве объектов компонент htq, нажмем правой кнопкой мыши на элемент дерева Fields и в возникшем контекстном меню выберем пункт «Add all fields» (см. **)**. На экране возникнет диалог подключения к базе данных HyTech, и в случае внесения правильных реквизитов, все четыре поля, приведенные в запросе, находящемся в htq.SQL, будут добавлены автоматически (). Присвоим свойству Visible значение false для полей nAuthorKey и nPHoseKey, иначе в таблице будут отображаться числовые значения, содержащиеся в указанных полях. Изменим значение свойства DisplayLabel на «Название книги» и «Год издания» для полей cName и nIssYear соответственно.



**Рис. 41. Добавление полей в набор данных htq**



**Рис. 42. Список полей компонента htq**

Перед тем, как добавлять LookUp поля в компоненте htq, необходимо изменить ряд свойств компонентов htsqlAuthor и htsqlPHouse для того, чтобы эти два компонента обращались к таблицам author и phouse. Обратимся к компоненту htsqlAuthor. Свойству htqsqlAuthor.BaseName присвоим значение htb. В свойство htqsqlAuthor.SQL внесем код запроса, приведенный в .

Листинг

|  |
| --- |
| **fix all;**  **select** cFamName, nAuthorKey **from** author |

Выбор данных осуществляется из справочника author, при этом cFamName будет использоваться как отображаемое поле, а nAuthorKey как ключевое поле.

Аналогичным образом необходимо изменить поля компонента htqsqlPHouse. Свойству htqsqlPHouse.BaseName присвоим значение htb, в свойство htqsqlPHouse.SQL внесем код, приведенный в .

Листинг

|  |
| --- |
| **fix all;**  **select** nPHouseKey, cName **from** phouse |

Выбор данных осуществляется из справочника phouse, при этом cNameбудет использоваться как отображаемое поле, а nPHouseKey как ключевое поле.

После выполнения всех указанных действий можно создавать LookUp поля в компоненте htq. Снова нажмем правой кнопкой мыши на элемент дерева Fields компонента htq и в возникшем контекстном меню выберем пункт «New field» (см. ), после чего на экране появится форма добавления нового поля. Значения, которые необходимо присвоить полям формы добавления поля значения, приведены в таблице .

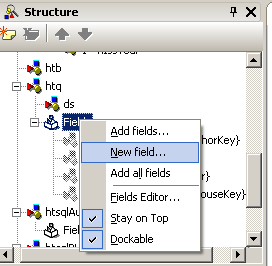


Рис. . Добавление нового поля в набор данных htq

Таб. . Значение полей для формы добавления нового поля cFamName

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название поля | Значение | Комментарий |
|  | Name | cFamName | Название поля в наборе данных htq |
|  | Component | htqcFamName | Название компонента |
|  | Type | string | Тип данных поля |
|  | FieldType | Lookup | Тип поля |
|  | KeyField | nAuthorKey | Поле из списка полей htq, которому будет присваиваться значение |
|  | DataSet | htsqlAuthor | Ссылка на набор данных, формирующий справочник |
|  | LookupKeys | nAuthorKey | Название поля из справочного набора данных (htsqlAuthor), значение которого будет присваиваться |
|  | ResultField | cFamName | Название поля из справочного набора данных (htsqlAuthor), значение которого будет отображаться в компоненте grd |

После выбора значения типа поля Lookup становятся доступными для редактирования свойства, относящиеся к получению значений из справочного набора данных. Обратите внимание, в поле KeyField необходимо указать название ранее созданного поля nAuthorKey компонента htq. В поле DataSet необходимо указать ссылку на набор данных, формирующий справочник (htsqlAuthor), а в полях LookupKeys и ResultField необходимо указать ссылки на поля справочного набора данных (htsqlAuthor), значение которых будет присваиваться (LookupKeys) полю основного набора данных (nAuthorKey) и отображаться в компоненте grd(ResultField). Внешний вид формы добавления нового поля с заполненными значениями представлен на .

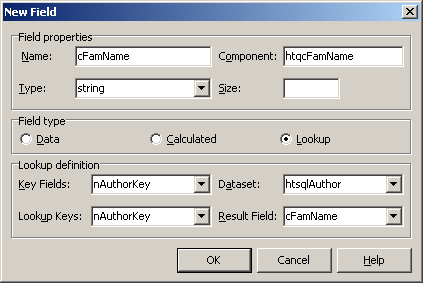


Рис. . Внешний вид формы добавления нового поля с заполненными значениями

Пользуясь той же методикой, создадим ещё одно Lookup поле cPHouseName. Значения, которые необходимо присвоить полям формы добавления поля значения, приведены в таблице .

Таб. . Значение полей для формы добавления нового cPHouseName

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название поля | Значение | Комментарий |
|  | Name | cPHouseName | Название поля в наборе данных htq |
|  | Component | htqcPHouseName | Название компонента |
|  | Type | string | Тип данных поля |
|  | FieldType | Lookup | Тип поля |
|  | KeyField | nPHouseKey | Поле из списка полей htq, которому будет присваиваться значение |
|  | DataSet | htsqlPHouse | Ссылка на набор данных, формирующий справочник |
|  | LookupKeys | nPHouseKey | Название поля из справочного набора данных htsqlPHouse), значение которого будет присваиваться |
|  | ResultField | cName | Название поля из справочного набора данных (htsqlPHouse), значение которого будет отображаться в компоненте grd |

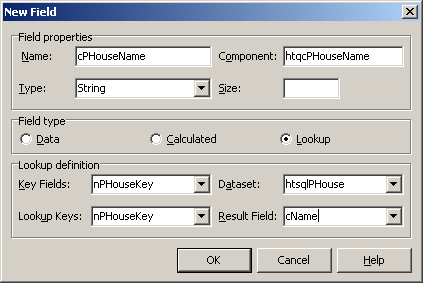


Рис. 45. Внешний вид формы добавления нового поля с заполненными значениями

Изменим значение свойства DisplayLabel на «Автор» и «Издательство» для добавленных полей cFamName и cPHouseName соответственно.

Для того чтобы LookUp поля содержали информацию, при активизации основного набора данных htq также необходимо выполнять активизацию справочных наборов данных htqsqlAuthor и htqsqlPHouse, что необходимо осуществить в обработчике события нажатия на кнопку bSelect. Обработчик представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if htq.Active then  htq.Close;  // активизация справочного набора данных htsqlPHouse  htsqlPHouse.Open;  // активизация справочного набора данных htsqlAuthor  htsqlAuthor.Open;  // активизация основного набора данных htq  htq.Open;  dbeBookName.DataField := htq.FindField('cName').FieldName;  end; |

Также, как и в предыдущих разделах, по событию AfterScroll компонента htq будем заполнять поле Caption метки lbAuthorFam фамилией автора, однако название поля, содержащего фамилию, изменилось, что необходимо отразить в коде обработчика. Код представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.htqAfterScroll(DataSet: TDataSet);  begin  lbAuthorFam.Caption := htq.FindField('cFamName').Value;  end; |

В случае, если поля создаются вручную в процессе разработки приложения, а не автоматически в процессе выполнения приложения, при внесении каких-либо изменений в состав или название полей необходимо проверять, что произведенные изменения не повлияют на работоспособность программы.

## Изменение набора данных – добавление и редактирование строк таблицы

Скомпилируем и запустим приложение, установим связь с базой данных, нажмем на кнопку «Выполнить запрос». Внешний вид приложения представлен на .

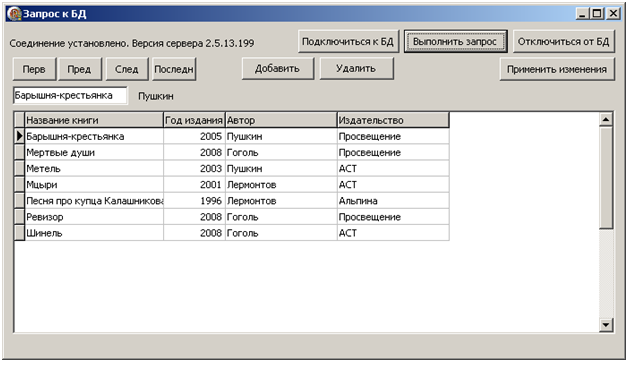


Рис. . Внешний вид приложения после нажатия на кнопку «Выполнить запрос»

Отредактируем строку набора данных - изменим значение поля «Издательство» для книги «Метель» и нажмем на кнопку «Применить изменения» (см. , 48- ).

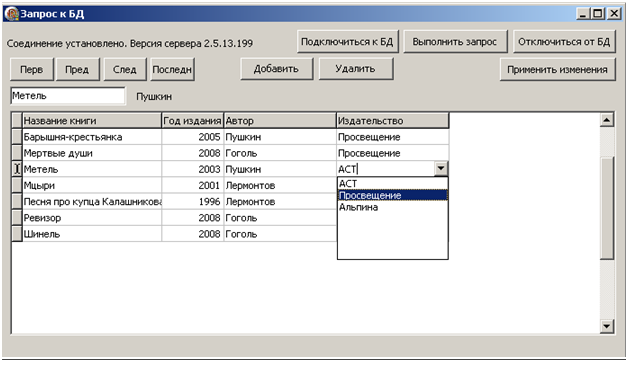


Рис. . Внешний вид приложения в процессе редактирования строки

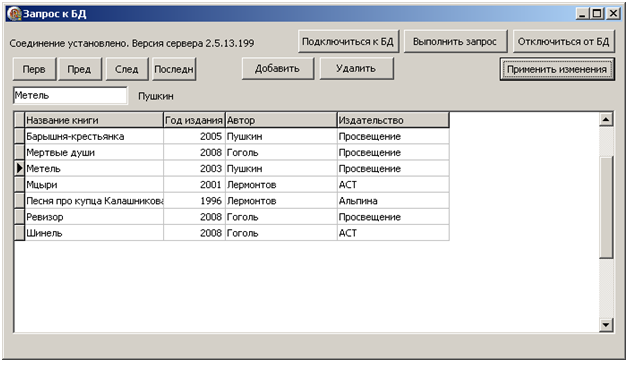


Рис. . Внешний вид приложения, строка отредактирована

Добавим строку, нажав на кнопку «Добавить», и внесем значения во все 4 поля, нажмем на кнопку «Применить изменения»( см. – ).

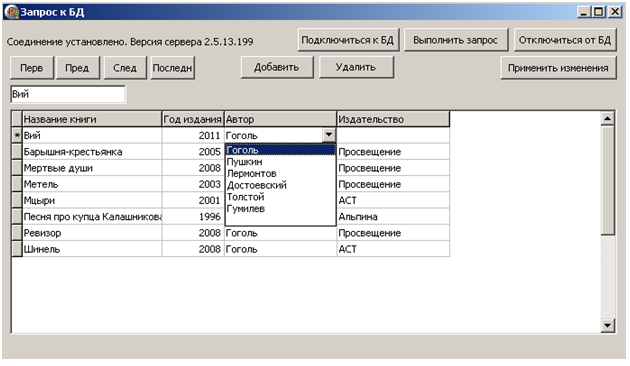


Рис. . Внешний вид приложения в процессе добавления строки

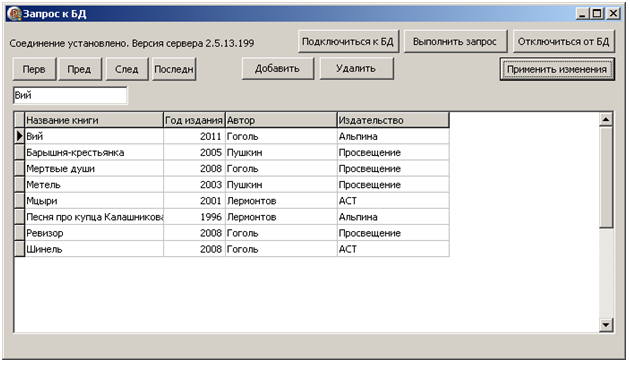


Рис. . Внешний вид приложения, строка добавлена

Удаление строки из таблицы осуществляется при нажатии на кнопку «Удалить». Поведение приложения при этом аналогично описанному в разделе поведению.

## Поиск строки по значению поля

Для демонстрации поиска воспользуемся приложением из раздела .

Достаточно часто при использовании компонента, возвращающего набор данных, встает задача поиска номера строки по значению поля. В стандартном компоненте TDataSet поиск осуществляется посредством метода Locate. Аналогом метода в компоненте THtQuery является RowbyValue. Поиск выполняется сначала во внутреннем буфере,если поиск не дал результата вызывается метод предка. Функция возвращает номер строки, в случае, если строка не найдена, будет возвращено значение «-1». Добавим на форму кнопку с закладки Standard , назовем её bSearch, компонент TComboBox с той же закладки, назовем его cbFieldList, и компонент TEdit, закладка Standard, назовем его eSearchValue. Будем заполнять cbFieldList названиями полей при выполнении запроса к БД, а при нажатии на кнопку будем выполнять поиск строки, в которой выбранному полю из списка cbFieldList будет соответствовать значение, указанное в eSearchValue.

Внешний вид формы в процессе разработки представлен на .

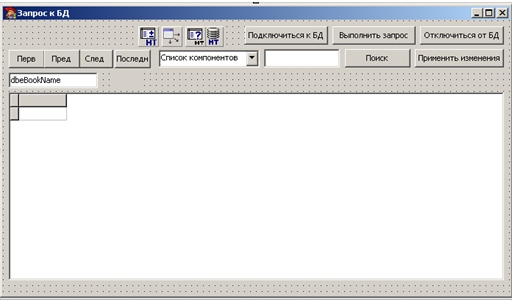


Рис. . Внешний вид формы (среда разработки)

Новый код обработчика нажатия на кнопку «Выполнить запрос» и код обработчика нажатия на кнопку «Поиск» представлены в .

Листинг

|  |
| --- |
| procedure TConnectForm.bSelectClick(Sender: TObject);  var  i : integer;  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  htq.Active := true;  for i:= 0 to htq.FieldCount - 1 do  cbFieldList.AddItem(htq.Fields[i].FieldName,nil);  end;  procedure TConnectForm.bSearchClick(Sender: TObject);  begin  htq.RecNo :=  htq.RowByValue(cbFieldList.Text,eSearchValue.Text);  grd.SetFocus;  end; |

Внешний вид приложения после нажатия на кнопку «Поиск» представлен на .

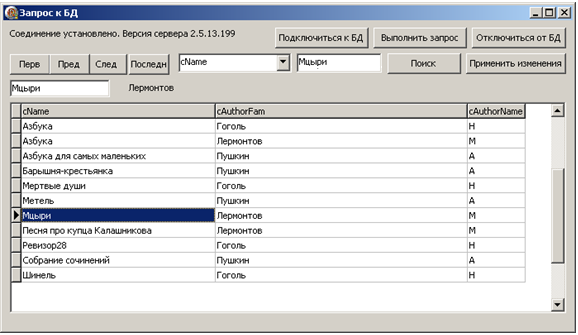
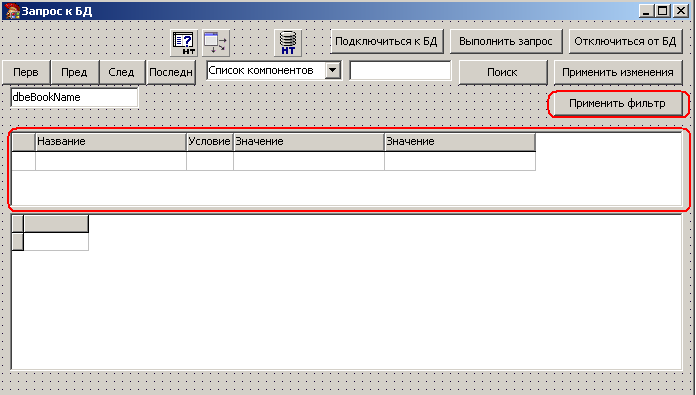


Рис. . Внешний вид приложения после нажатия на кнопку «Поиск»

Исходный код получившегося приложения можно скачать по адресу [www.hytechdb.ru/sources/QueryApp.rar](http://www.hytechdb.ru/sources/QueryApp.rar).

## **Фильтрация информации в наборе данных**

Фильтрация информации позволяет пользователю отображать на экране только нужный ему в настоящий момент информацию. Для демонстрации поиска разметим на форме из раздела компонент dbFilter типа THtxDBFilter с закладки HtComp и кнопку bFilter с закладки Standard, по нажатию на которую будем осуществлять фильтрацию информации. Внешний вид формы в среде разработки представлен на .



**Рис. 53. Внешний вид формы (среда разработки)**

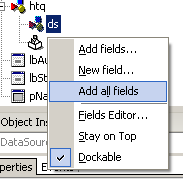
Свойству SQL компонента htq присвоим код, приведенный в .

Листинг

|  |
| --- |
| fix all;  select \*  from book |

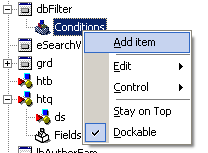
В компоненте htq удалим обработчик события htqAfterScroll, оставшийся от предыдущего примера.

Создадим поля в компоненте htq, нажав на компонент правой клавишей мыши и выбрав из конекстного меню пункт «Add all fields».



**Рис. 54. Добавление полей**

Добавим условие в компоненте dbFilter, нажав на него правой клавишей мыши и выбрав пункт «Add item» (см. ) . В созданном условии полю LookUpDataSet присвоим значение htq, полю FieldName присвоим значение cName, DisplayLabel – «Название книги».

****

**Рис. 55. Добавление условия**

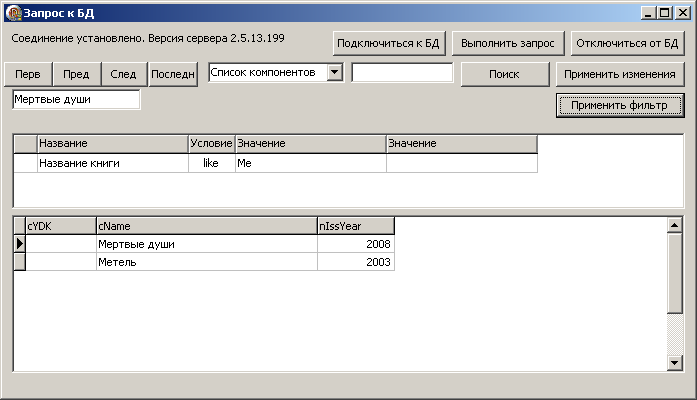
По нажатию на кнопку будем применять фильтр. Код обработчика события нажатия на кнопку приведен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bFilterClick(Sender: TObject);  begin  htq.SQL.Text := htq.SQL.Text + ' and '+dbFilter.MakeText();  htq.Requery;  end; |

Скомпилируем и запустим приложение. Установим связь с БД, выполним запрос. Сформируем условие так, как показано на и нажмем кнопку «Применить фильтр». Результат применения фильтра приведен на .

**Рис. 56. Формирование условия в ходе работы с приложением**

****

**Рис. 57. Результат применения фильтра**

## **Журналирование изменений набора данных в памяти**

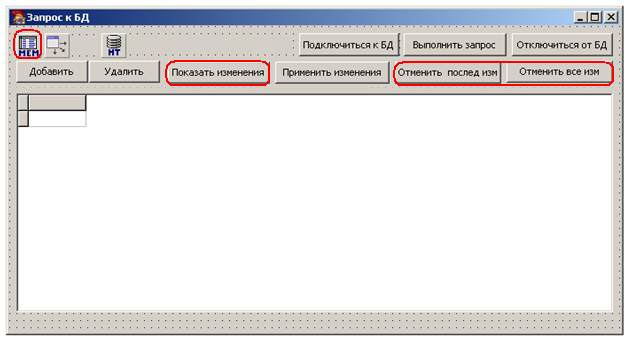
Журналирование изменений набора данных в памяти позволяет более гибко и эффективно работать с изменяемой пользователем информацией. Компоненты, поддерживающие журналирование изменений, реализуют такие возможности, как построение набора данных из измененных строк, получение статуса строки (изменена, добавлена), отмены изменения, возврата набора данных в состояние, в котором он находился в момент открытия.

Технику работы с журналированием изменений в памяти рассмотрим на примере. Для этого модифицируем форму из раздела . Добавим на форму ряд компонентов, ряд компонентов удалим и некоторым изменим поведение. Список всех необходимых компонентов главной формы представлен в таблице .

Таб. . Компоненты главной формы примера журналирования изменений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Название | Тип | Закладка | Статус | Назначение |
| 1. | htb | THtBase | HyTech | Без изменений | Осуществление связи с БД |
| 2. | htq | THtQuery | HyTech | Изменен текст запроса | Первичное получение информации из БД |
| 3. | htmem | TMemTable | HyTech | Добавлен | Работа с результатами запросов в памяти с журналированием изменений |
| 4. | ds | TDataSource | DataAccess | Изменено свойство | Установка связи компонента htmem с grd |
| 5. | grd | TDbGrid | DataControls | Без изменений | Отображение набора данных htmem |
| 6. | bConnect | TButton | Standard | Без изменений | Установка связи с БД |
| 7. | bDisconnect | TButton | Standard | Без изменений | Отключение от БД |
| 8. | bSelect | TButton | Standard | Изменен обработчик | Активация компонента htmem. Первичное наполнение компонента данными |
| 9. | bAdd | TButton | Standard | Изменен обработчик | Перевод компонента htmem в режим добавления информации |
| 10. | bDelete | TButton | Standard | Изменен обработчик | Удаление текущей строки набора данных htmem |
| 11. | bShowChanges | TButton | Standard | Добавлен | Создание и отображение набора измененных данных htmem |
| 12. | bPost | TButton | Standard | Изменен обработчик | Применение изменений |
| 13. | bCancelLastChange | TButton | Standard | Добавлен | Отмена последнего изменения в наборе данных htmem |
| 14. | bCancelAll | TButton | Standard | Добавлен | Отмена всех изменений набора данных htmem |
| 15. | lbStatus | TLabel | Standard | Без изменений | Отображение информации о СУБД в случае успешного подключения |

Внешний вид формы в среде проектирования приведен на . Красным цветом на рисунке отмечены добавленные компоненты.

****

**Рис. 58. Внешний вид формы (среда разработки)**

Коды обработчиков нажатия на кнопки приведены в . Комментарии к коду приведены в разделах 2.5.1 - .

Листинг

|  |
| --- |
| // добавление строки в набор данных  procedure TConnectForm.bAddClick(Sender: TObject);  begin  if not htMem.Active then  exit;  htMem.Insert;  end;  // отмена всех изменений набора данных  procedure TConnectForm.bCancelAllClick(Sender: TObject);  begin  if not htMem.Active then  exit;  htMem.CancelUpdates;  end;  // отмена последнего изменения набора данных  procedure TConnectForm.bCancelLastChangeClick(Sender: TObject);  begin  if not htMem.Active then  exit;  htMem.UndoLastChange(true);  end;  // установка соединения с базой данных  procedure TConnectForm.bConnectClick(Sender: TObject);  begin  htb.Connect;  if htb.Active then  lbStatus.Caption := 'Соединение установлено. Версия сервера'+  htb.StrVersion;  end;  // удаление текущей строки из набора данных  procedure TConnectForm.bDeleteClick(Sender: TObject);  begin  if not htMem.Active then  exit;  htMem.Delete;  end;  // отключение от базы данных  procedure TConnectForm.bDisconnectClick(Sender: TObject);  begin  htb.Disconnect;  lbStatus.Caption := 'База данных не доступна';  end;  // применение изменений, очистка журнала изменений htmem  procedure TConnectForm.bPostClick(Sender: TObject);  begin  if not htMem.Active then  exit;  htMem.Reconcile;  end;  // активация компонента htMem, первичное наполнение его данными  procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if htMem.Active then  htMem.Close;  htMem.Open;  htq.Open;  htMem.CopyData(htq);  htq.Close;  end;  //формирование и отображение набора измененных данных компонента htmem  procedure TConnectForm.bShowChangesClick(Sender: TObject);  var  mdf : TModifiedDataForm;  begin  mdf := TModifiedDataForm.Create(self);  mdf.dsModified.DataSet := htMem.MakeUpdatesDataSet([usModified, usInserted, usDeleted],'stat');  mdf.dsModified.DataSet.Open;  mdf.grdModified.DataSource := mdf.dsModified;  mdf.ShowModal;  mdf.dsModified.DataSet.Close;  FreeAndNil(mdf);  end; |

Компонент htb оставлен без изменений и по-прежнему будет использоваться для установки соединения с базой данных (при нажатии на кнопку bConnect). В случае успешной установки соединения информация о сервере будет помещаться в компонент lbStatus. Отключение от БД будет также осуществляться при нажатии на кнопку bDisconnect.

Журналирование изменений в примере включает следующие основные шаги ( в реальных приложениях рекомендуется использовать аналогичную технику):

1. обращение к базе данных для выполнения запроса на получение информации (посредством компонента htq);
2. копирование данных из компонента htq в htmem;
3. работа с набором данных в памяти (добавление, редактирование, удаление строк);
4. работа с журналом изменений.

## **Извлечение данных из БД и копирование их в таблицу в памяти**

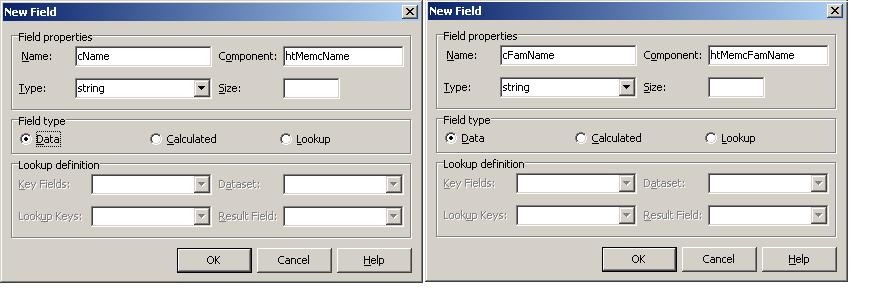
Для извлечения данных из БД воспользуемся компонентом htq. В свойстве компонента htq.SQL разместим запрос, приведенный в .

Листинг

|  |
| --- |
| **fix all;**  **select** cName, cFamName **from** author  **order by** cName |

Для организации таблицы в памяти используется компонент htmem(TMemTable), поддерживающий журналирование изменений, совершаемых в наборе данных, хранящемся непосредственно в этом компоненте. TMemTable является наследником стандартного компонента TDataSet, поэтому он обладает теми же стандартными методами работы, что и компонент TDataSet, например: открытие/закрытие набора данных (open, close), добавление, удаление, редактирование строк набора данных (Insert, Delete, Edit), создание полей, поиск полей по названию, методы навигации по набору данных (Next, Prior, First, Last) и некоторые другие.

Создадим в компоненте htmem два строковых поля – cName и cFamName, DisplayLabel Имя и Фамилия соответственно, см. .



**Рис. 59. Добавление полей в среде разработки**

После открытия компонента htq становится доступным копирование элементов результата запроса в htmem посредством метода CopyData, которому необходимо передать ссылку на компонент-наследник TDataSet (htq). В нашем примере извлечение данных будет происходить при нажатии кнопки bSelect («Выполнить запрос»). Код обработчика нажатия на кнопку представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bSelectClick(Sender: TObject);  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if htMem.Active then  htMem.Close;  htMem.Open; // активация компонента htMem  htq.Open; // активация компонента htq, выполнение запроса к БД  htMem.CopyData(htq); // копирование набора данных, полученного в ходе  // запроса компонентом htq в компонент htMem  htq.Close; // закрытие компонента htq  end; |

## **Работа с набором данных в памяти (добавление, удаление, редактирование строк)**

Методы работы со строками набора данных компонента TMemTable не отличаются от методов работы со строками набора данных THtQuery.

Для связи компонента htMem с визуальными компонентами необходимо использовать компонент ds (TDataSource). Связь ds с компонентом htmem создается также, как и с компонентом htq, таким образом, свойству ds.DataSet необходимо присвоить значение htMem. Ссылка на компонент ds (как и в предыдущих примерах) проставлена в свойстве grd.DataSource.

Добавление строк в набор данных и удаление строк из набора данных будем осуществлять при нажатии на кнопку bAdd и bDelete соответственно. Код обработчиков нажатия на кнопки представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bAddClick(Sender: TObject);  begin  htMem.Insert;  end;  procedure TConnectForm.bDeleteClick(Sender: TObject);  begin  htMem.Delete;  end; |

## **Работа с журналом изменений**

Компонент htmem позволяет выгрузить в отдельный набор данных измененные строки, при этом можно указать, какого типа изменение интересует – добавление, редактирование или удаление. Для того, чтобы компонент htmem осуществлял журналирование изменений, свойству htmem.LogChanges необходимо присвоить значение true. Для отображения измененного набора данных создадим ещё одну форму (модуль ModifiedDataUnit, форма ModifiedDataForm). Для того чтобы форма ModifiedDataForm была доступна из главного модуля, необходимо в раздел uses главного модуля добавить ссылку на модуль ModifiedDataUnit.

На форму ModifiedForm поместим компонент dsModified (типа TDataSource)с закладки DataAccess и компонент grdModified (типа TDBGrid) с закладки DataControls. Внешний вид формы представлен на .

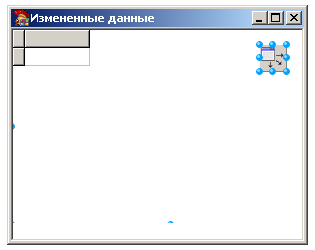


Рис. . Внешний вид формы для отображения измененных данных в среде разработки

Для выгрузки отредактированных записей в отдельный набор данных используется кнопка bShowChanges, при нажатии на которую будет открываться вспомогательная форма, отображающая журнал с измененными строками. Код обработчика нажатия на кнопку представлен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bShowChangesClick(Sender: TObject);  var  mdf : TModifiedDataForm;  begin  mdf := TModifiedDataForm.Create(self);  mdf.dsModified.DataSet := htMem.MakeUpdatesDataSet([usModified],’stat’);  mdf.dsModified.DataSet.Open;  mdf.grdModified.DataSource := mdf.dsModified;  mdf.ShowModal;  mdf.dsModified.DataSet.Close;  FreeAndNil(mdf);  end; |

Для создания набора данных с измененными записями был использован метод htMem.MakeUpdatesDataSet, которому были переданы два параметра. Значением первого параметра является набор, элементами которого являются стандартные значения типа TUpdateStatus, такие как usModified, usInserted, usDeleted. В нашем примере мы выгружали только отредактированные строки, поэтому набор состоял из одного элемента [usModified].

Во втором параметре указано название поля для отображения вида изменений – он полезен, если набор данных создается не для одного вида изменений, а для нескольких, например [usInserted, usModified], второй параметр не обязателен и, если он не указан, то поле просто не создается. После окончания работы с измененным набором данных, его необходимо явно закрыть методом Close - в этом случае внутренний счетчик измененных наборов данных компонента htmem станет равным 0, до этого момента редактировать информацию, содержащуюся в htmem невозможно.

Запустим приложение. Установим связь с базой данных, нажмем на кнопку «Выполнить запрос». Отредактируем строку и нажмем на кнопку «Показать изменения». Внешний вид приложения представлен на - .

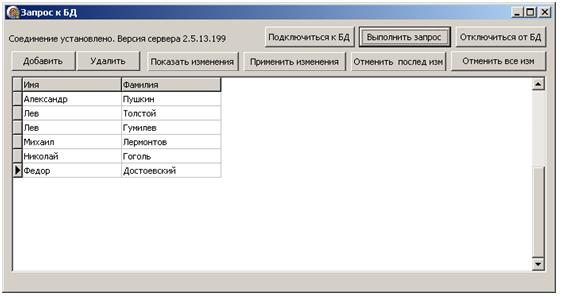


Рис. . Внешний вид приложения после нажатия на кнопку «Выполнить запрос»

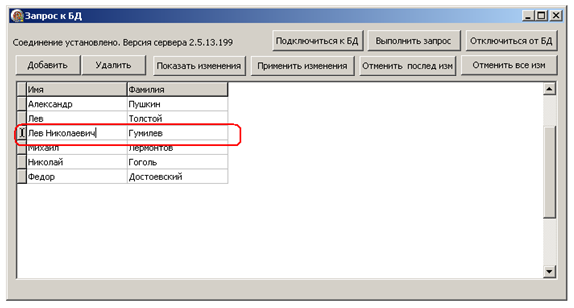


Рис. . Внешний вид приложения при редактировании записи

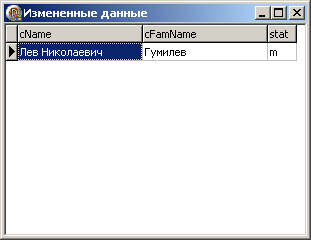


Рис. . Внешний вид приложения (форма «Измененные данные») после нажатия на кнопку «Показать изменения»

## **Применение и отмена изменений**

Компонент htMem позволяет отменить изменения по одному или все сделанные изменения единовременно, а также сделать легитимными все изменения, очистив при этом журнал изменений. Для отмены всех изменений будем использовать кнопку «Отменить все изм», для применения изменений будем использовать кнопку «Применить изменения». Также изменим обработчик нажатия на кнопку «Показать изменения» так, чтобы в измененный набор данных попадали не только измененные строки, но и добавленные и удаленные. Кнопка bPost в данном примере будет использоваться для реализации возможности применения изменений, то есть очистки журнала изменений компонента htMem. При нажатии на кнопку bAdd будем добавлять строки, при нажатии на кнопку bDelete будем удалять строки. Код обработчиков нажатий на кнопки представлен в листинге .

Листинг

|  |
| --- |
| // применить изменения, очистить журнал изменений компонента htmem  procedure TConnectForm.bPostClick(Sender: TObject);  begin  if not htMem.Active then  exit;  htMem.Reconcile;  end;  // отменить все изменения  procedure TConnectForm.bCancelAllClick(Sender: TObject);  begin  htMem.CancelUpdates;  end;  //сформировать и отобразить набор измененных данных компонента htmem  procedure TConnectForm.bShowChangesClick(Sender: TObject);  var  mdf : TModifiedDataForm;  begin  mdf := TModifiedDataForm.Create(self);  mdf.dsModified.DataSet := htMem.MakeUpdatesDataSet([usModified, usInserted, usDeleted],'stat');  mdf.dsModified.DataSet.Open;  mdf.grdModified.DataSource := mdf.dsModified;  mdf.ShowModal;  mdf.dsModified.DataSet.Close;  FreeAndNil(mdf);  end;  // отменить последнее изменение  procedure TConnectForm.bCancelLastChangeClick(Sender: TObject);  begin  htMem.UndoLastChange;  end; |

Запустим приложение, установим связь с базой данных, наполним htMem информацией и осуществим следующие действия: добавим автора «Ольга Громыко», отредактируем «Имя» Михаила Лермонтова и удалим строку «Александр Пушкин». Внешний вид приложения после проведения всех манипуляций с данными приведен на рис. - .

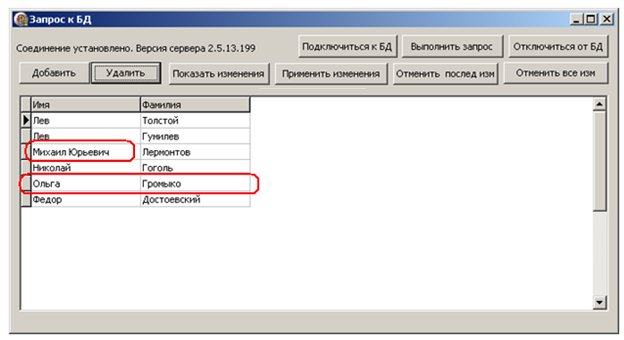


Рис. . Внешний вид приложения после манипуляций с данными

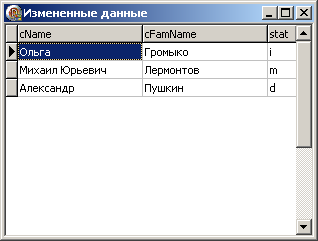


Рис. . Внешний вид приложения (форма «Измененные данные») после нажатия на кнопку «Показать изменения»

Отменим все три операции, три раза нажав на кнопку «Отменить посл изм». Внешний вид приложения приведен на - .

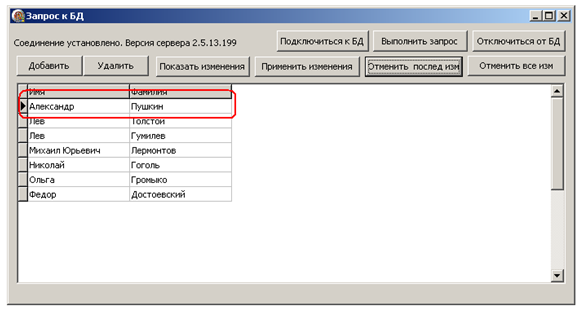


Рис. . Внешний вид приложения – строка «Александр Пушкин» восстановлена

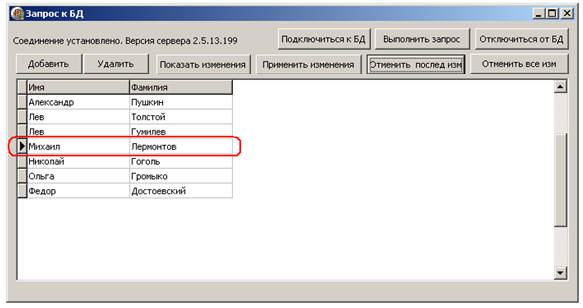


Рис. . Внешний вид приложения - строка «Михаил Лермонтов» возвращена в исходное состояние

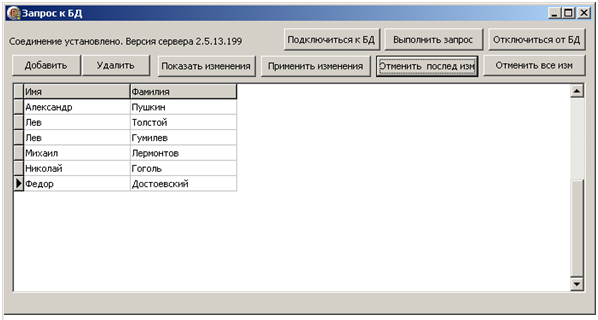


Рис. . Внешний вид приложения – строка «Ольга Громыко» удалена

Ещё раз добавим, отредактируем и удалим по одной строке, а затем отменим все изменения, нажав на кнопку «Отменить все изм». Внешний вид приложения представлен на -.

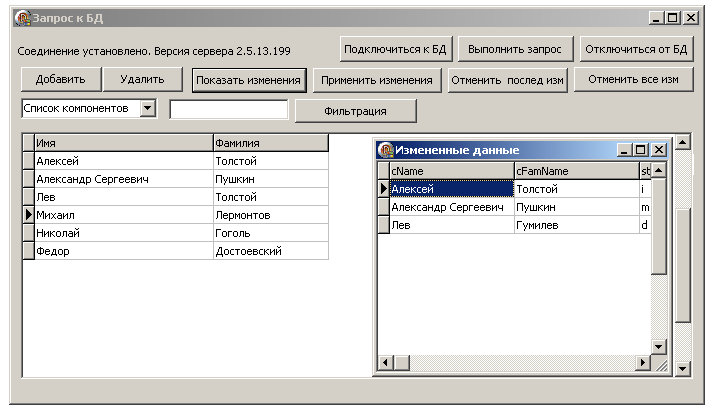


Рис. . Внешний вид приложения - набор данных изменен

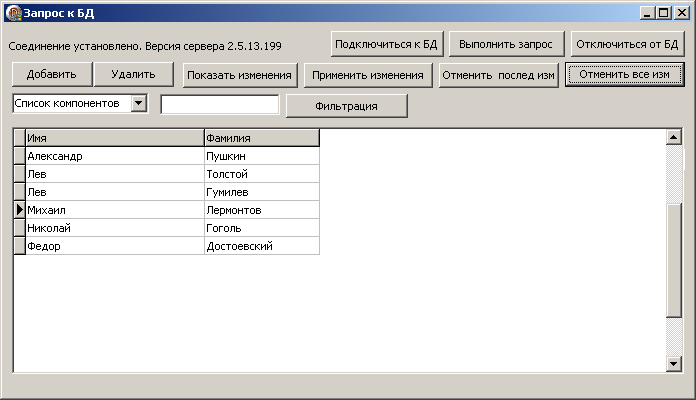


Рис. . Внешний вид приложения – все изменения отменены

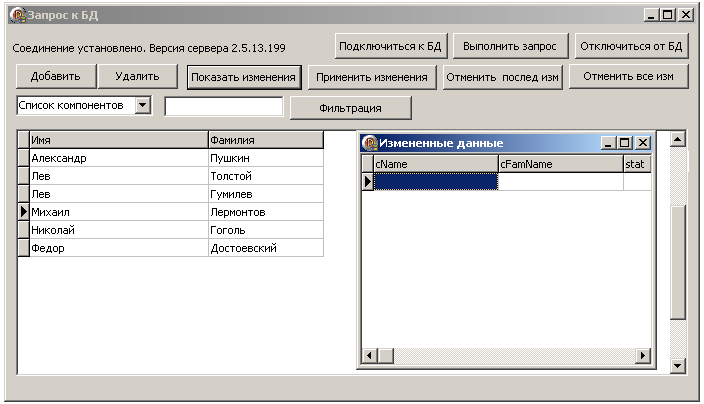


Рис. . Внешний вид приложения – набор измененных данных не содержит строк

Добавим две строки, а затем применим изменения, нажав на кнопку «Применить изменения». Внешний вид приложения представлен на - .

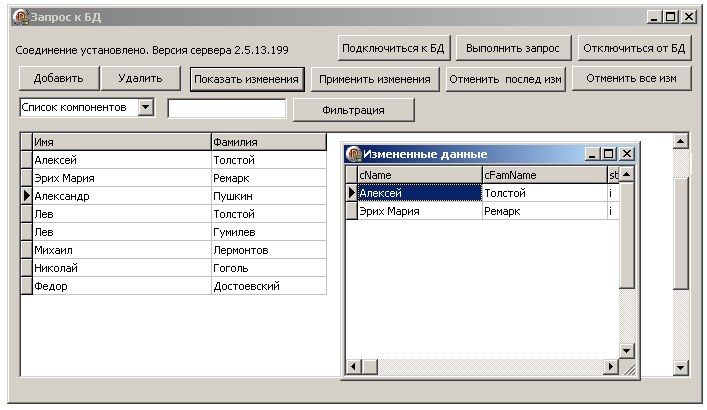


Рис. . Внешний вид приложения - строки добавлены

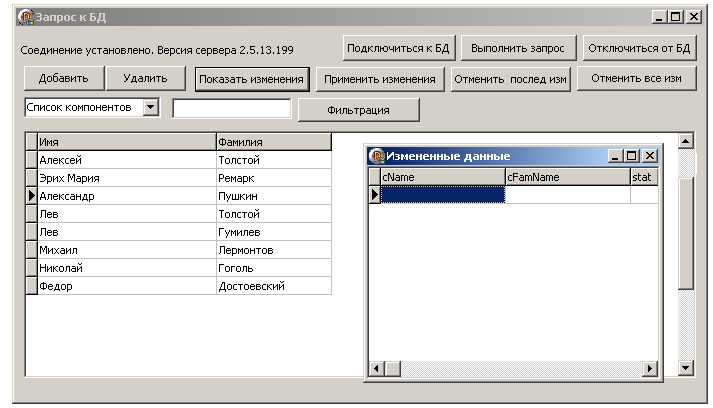


Рис. . Внешний вид приложения - набор измененных данных не содержит строк

Обратите внимание, что применение изменений очищает журнал изменений и делает изменения легитимными только для компонента. На информацию в базе данных сделанные изменения не оказывают никакого влияния.

## **Пакетная загрузка данных в таблицу HyTech**

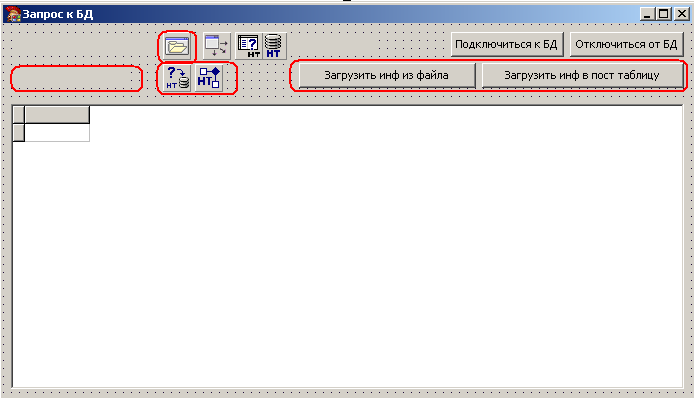
Для осуществления пакетной загрузки в таблицу базы данных могут быть использованы компоненты THtPump или THtDSPump. Выбор типа компонента осуществляется в зависимости от типа источника данных для загрузки. Если источником данных является компонент – наследник класса TDataSet, то для загрузки необходимо использовать THtDsPump, во всех остальных случаях необходимо использовать THtPump.

Пакетную загрузку рассмотрим на примере. Для этого модифицируем форму из раздела . Добавим на форму ряд компонентов, ряд компонентов удалим и некоторым изменим поведение. Список всех необходимых компонентов главной формы представлен в .

Таб. . Компоненты главной формы примера пакетной загрузки изменений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Название | Тип | Закладка | Статус | Назначение |
| 1. | htb | THtBase | HyTech | Без изменений | Осуществление связи с БД |
| 2. | htpump | THtPump | HyTech | Добавлен | Осуществление пакетной загрузки |
| 3. | htsp | THtStoredProc | HyTech | Добавлен | Формирование набора данных |
| 3. | htq | THtQuery | HyTech | Изменен текст запроса | Формирование набора данных |
| 4. | ds | TdataSource | DataAccess | Изменено свойство | Установка связи компонента htq с grd |
| 5. | grd | TdbGrid | DataControls | Без изменений | Отображение набора данных htq |
| 6. | bConnect | TButton | Standard | Без изменений | Установка связи с БД |
| 7. | bDisconnect | TButton | Standard | Без изменений | Отключение от БД |
| 8. | bLoadFromFile | TButton | Standard | Добавлен | Выбор файла для загрузки во временную таблицу и загрузка во временную таблицу |
| 9. | bSendToTable | TButton | Standard | Добавлен | Загрузка информации из временной в постоянную таблицу |
| 10. | opDialog | TOpenDialog | Dialogs | Добавлен | Выбор файла – источника данных |
| 11. | lbStatus | TLabel | Standard | Без изменений | Отображение информации о СУБД в случае успешного подключения |
| 12. | lBufContCount | TLabel | Standard | Добавлен | Отображение количества записей в буфере htpump |

Внешний вид формы в среде проектирования приведен на . Красным цветом на рисунке отмечены добавленные компоненты.



lbBufContCount (TLabel)

Рис. . Внешний вид формы (среда разработки)

Для демонстрации работы пакетной загрузки выбран следующий метод: информация из текстового файла, выбираемого посредством компонента opDialog при нажатии на кнопку «Загрузить инф из файла», загружается во временную таблицу, содержимое которой после загрузки отображается в компоненте grd и при нажатии на кнопку «Загрузить информациюв пост таблицу» переписывается из временной таблицы в постоянную. Обработчики событий нажатия на кнопки bLoadFromFile и bSendToTable представлены в .

Листинг

|  |
| --- |
| // загрузка информации из файла во временную таблицу  procedure TConnectForm.bLoadFromFileClick (Sender: TObject);  var  strLst : TStringList;  i : integer;  strtmp, strName, strChief: string;  begin  // проверка, произведено ли подключение к БД  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  // отображение диалога выбора файла-источника данных  if opDialog.Execute then  begin  strLst := TStringList.Create();  // копирование информации из файла в переменную типа TStringList  strLst.LoadFromFile(opDialog.FileName);  if htPump.Active then  htPump.Close;  htPump.Open();  for i := 0 to strLst.Count - 1 do  begin  strName := copy(strLst[i], pos('"',strLst[i])+1, pos(' "',strLst[i])-3);  strtmp := copy(strLst[i], pos(' "',strLst[i]));  strChief := copy(strtmp, 3, length(strtmp)-3);  // присвоение полям компонента htpump значений  // из текущей строки переменной типа TStringList  htPump.FieldByName('cName').AsString := strName;  htPump.FieldByName('cChiefEdit').AsString := strChief;  // сброс значений полей компонента htpump в буфер компонента htpump  htPump.Post;  end;  // закрытие компонента и перенос  // информации из буфера компонента во временную таблицу  htPump.Close;  // отображение содержания временной таблицы  htq.SQL.Text := 'select \* from jourtmp';  htq.Open;  FreeAndNil(strLst);  end;  end;  // загрузка информации из временной таблицы в постоянную  var v : variant;  begin  // выполнение запроса, не создающего результата  v := htsp.ExecNoSelect;  htsp.Close;  if htq.Active then  htq.Close;  htq.SQL.Text := 'fix all; select cName, cChiefEdit from journal';  htq.Open;  end; |

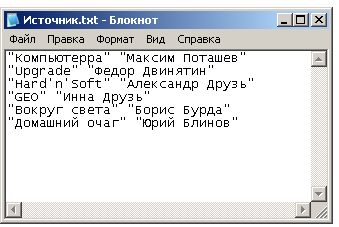
Комментарии к коду приведены в следующих разделах.

## **Загрузка информации из файла во временную таблицу**

Присвоим значение htb свойству htPump.BaseName. Для иллюстрации пакетной загрузки будем создавать временную таблицу jourtmp, а затем перегружать из неё информацию в таблицу journal, для чего свойству htPump.TableName присвоим значение jourtmp. При открытии компонент будет создавать временную таблицу с именем, указанном в этом свойстве. Нажмем правой кнопкой мыши на элемент Fields компонента htpump в дереве объектов и в контекстном меню выберем пункт «Add Item», при этом в список полей будет добавлено одно поле. Назовем его cName, проставим свойству HtType значение etChar, а свойству Size 100 таким же способом создадим поле cChiefEdit (см. - ). Атрибуты создаваемой временной таблицы будут созданы в соответствии с полями компонента htpump.

|  |  |
| --- | --- |
|  |  |
| Рис. . Добавление нового поля | Рис. . Компонент htpump c двумя добавленными полями |

В нашем примере в качестве источника информации используется текстовый файл. В файле значения, предназначенные для различных полей, заключены в двойные кавычки и отделены друг от друга пробелом. Содержание файла представлено на .



**Рис. 77. Содержание текстового файла**

При нажатии на кнопку bLoadFromFile осуществляются следующие действия:

* проверка, произведено ли подключение к БД;
* отображение диалога выбора файла-источника данных;
* копирование информации из файла в переменную типа TStringList;
* присвоение полям компонента htpump значений из текущей строки переменной типа TStringList;
* сброс значений полей компонента htpump в буфер компонента htpump ;
* закрытие компонента htpump, при этом происходит перенос информации из буфера компонента во временную таблицу;
* отображение содержания временной таблицы.

Код обработчика события нажатия на кнопку bLoadFromFile приведен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bSelectFileClick(Sender: TObject);  var  strLst : TStringList;  i : integer;  strtmp, strName, strChief: string;  begin  // проверка, произведено ли подключение к БД  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  // отображение диалога выбора файла-источника данных  if opDialog.Execute then  begin  strLst := TStringList.Create();  // копирование информации из файла в переменную типа TStringList  strLst.LoadFromFile(opDialog.FileName);  if htPump.Active then  htPump.Close;  htPump.Open();  for i := 0 to strLst.Count - 1 do  begin  strName := copy(strLst[i], pos('"',strLst[i])+1, pos(' "',strLst[i])-3);  strtmp := copy(strLst[i], pos(' "',strLst[i]));  strChief := copy(strtmp, 3, length(strtmp)-3);  // присвоение полям компонента htpump значений  // из текущей строки переменной типа TStringList  htPump.FieldByName('cName').AsString := strName;  htPump.FieldByName('cChiefEdit').AsString := strChief;  // сброс значений полей компонента htpump в буфер компонента htpump  htPump.Post;  end;  // закрытие компонента и перенос  // информации из буфера компонента во временную таблицу  htPump.Close;  // отображение содержания временной таблицы  htq.SQL.Text := 'select \* from jourtmp';  htq.Open;  FreeAndNil(strLst);  end;  end; |

Перенос информации из буфера компонента htpump в таблицу происходит в случае, если:

* инициировано закрытие компонента;
* размер буфера достиг предельно допустимого (предельно допустимое значение указывается в свойстве htpump.BufSize)
* инициирован принудительный перенос информации из буфера компонента в таблицу.

## **Перенос информации из временной в постоянную таблицу**

Перенос информации из временной таблицы в постоянную осуществляется при нажатии на кнопку bSendToTable при помощи компонента htsp, в свойстве BaseName которого укажем значение htb, а в свойство SQL поместим sql код, приведенный в .

Листинг

|  |
| --- |
| **insert into** journal (cName, cChiefEdit)  **select** char(rtrim(cName),100), char(rtrim(cChiefEdit),100) **from** jourtmp |

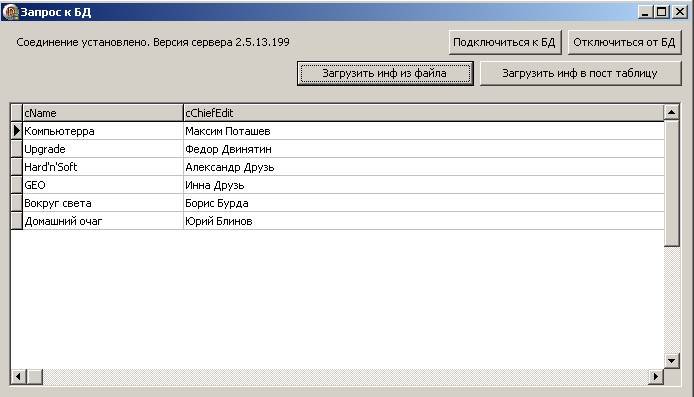
Компонент THtStoredProc удобно использовать тогда, когда необходимо выполнить запрос к БД, не создающий результат, например, запрос на добавление строк в таблицу, для этого используется метод ExecNoSelect.

Код обработчика нажатия на кнопку bSendToTable приведен в .

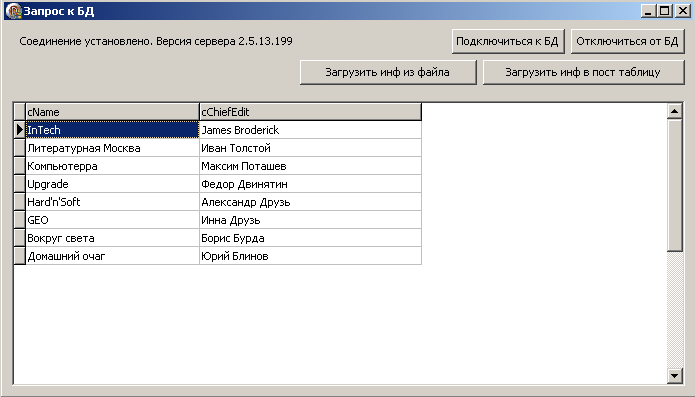
Листинг

|  |
| --- |
| procedure TConnectForm.bSendToTableClick(Sender: TObject);  var v : variant;  begin  // выполнение запроса, не создающего результата  v := htsp.ExecNoSelect;  htsp.Close;  if htq.Active then  htq.Close;  htq.SQL.Text := 'fix all; select cName, cChiefEdit from journal';  htq.Open;  end; |

Скомпилируем и запустим приложение. Установим связь с базой данных, нажмем на кнопку «Загрузить инф из файла» и выберем файл-источник данных (). Нажмем на кнопку «Загрузить инф в пост таблицу» ().



**Рис. 78. Внешний вид приложения, информация из файла загружена во временную таблицу**



**Рис. 79. Внешний вид приложения, информация из временной таблицы загружена в постоянную**

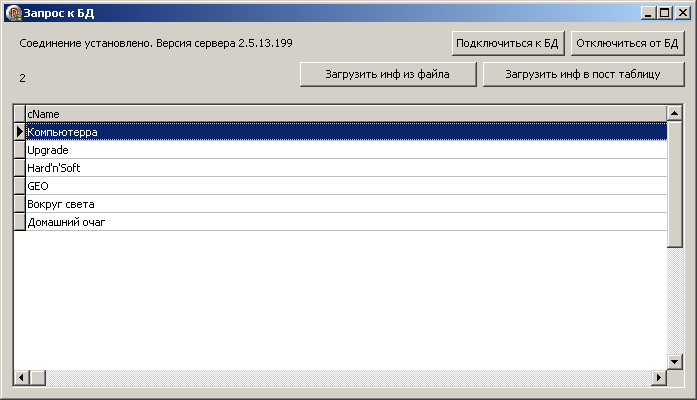
## **Принудительный перенос информации из буфера компонента htpump во временную таблицу**

Как уже было отмечено в разделе , сброс информации из буфера компонента htpump можно инициировать программно, для чего служит метод htPump.Flush. Также сброс информации из буфера осуществляется при достижении размера буфера предельно допустимого значения. Рассмотрим пример. Изменим размер буфера на минимально допустимый, то есть свойству htPump.BufSize присвоим значение 4096. Размер полей cName и cChiefEdit компонента htPump установим равным 500, таким образом одна строка будет занимать 1000 б, следовательно сброс содержимого буфера в таблицу будет осуществлен два раза – после 4ого добавления строки в буфер и после 6го. Изменим обработчик события нажатия на кнопку bLoadFromFile. В ходе цикла по строкам переменной типа TStringList будем присваивать метке lBufContCount значение, отражающее количество строк, находящихся в буфере, то есть разность двух значений свойств компонента htPump.Posted (показывает количество записей, которое поместили в компонент с момента его активизации) и htPump.Transferred (показывает количество записей, переданных в таблицу с момента активизации), также введем секундную задержку и перерисовку формы для того, чтобы в ходе выполнения загрузки в буфер компонента можно было визуально наблюдать количество записей в буфере компонента htpump. Код обработчика нажатия на кнопку bLoadFromFileClick приведен в .

Листинг

|  |
| --- |
| procedure TConnectForm.bLoadFromFileClick(Sender: TObject);  var  strLst : TStringList;  i : integer;  strtmp, strName, strChief: string;  begin  if not htb.Active then  begin  Application.MessageBox('Необходимо установить связь с БД', 'Внимание');  exit;  end;  if opDialog.Execute then  begin  strLst := TStringList.Create();  strLst.LoadFromFile(opDialog.FileName);  if htPump.Active then  htPump.Close;  htPump.Open();  for i := 0 to strLst.Count - 1 do  begin  strName := copy(strLst[i], pos('"',strLst[i])+1, pos(' "',strLst[i])-3);  strtmp := copy(strLst[i], pos(' "',strLst[i]));  strChief := copy(strtmp, 3, length(strtmp)-3);  htPump.FieldByName('cName').AsString := strName;  htPump.FieldByName('cChiefEdit').AsString := strChief;  htPump.Post;  // присвоим метке значение, отражающее количество строк, находящихся  // в буфере компонента htpump  lBufContCount.Caption := inttostr( htPump.Posted - htPump.Transferred );  sleep(1000);  Repaint;  end;  htPump.Close;  htq.SQL.Text := 'select \* from jourtmp';  htq.Open;  FreeAndNil(strLst);  end;  end; |

Скомпилируем и запустим приложение. Установим связь с базой данных, нажмем на кнопку «Загрузить инф из файла» и выберем файл-источник данных ().



**Рис. 80. Внешний вид приложения, информация из файла загружена во временную таблицу**

Нетрудно заметить, что в процессе загрузки в метке присваиваются последовательно значения «1», «2», «3», «0», «1», «2», таким образом, перенос информации из буфера осуществляется на 4ой итерации цикла и в этот момент буфер очищается. На рисунке выше в компоненте grd видно только одно поле cName, это связано с тем, что мы увеличили размер каждого из полей компонента htpump до 500. Увеличение размера поля не повлияет на возможность переноса данных из временной таблицы в постоянную, так как при переносе содержимое поля усекается до размера 100 символов посредством применения функции rtrim (см. код sql запроса компонента htsp в ).

## **Использование компонента – наследника TDataSet как источника информации для пакетной загрузки**

В случае если источником информации для пакетной загрузки является один или несколько компонентов – наследников TDataSet вместо компонента THtPump, целесообразно использовать компонент THtDsPump, свойству которого Src необходимо присвоить ссылку на компонент-наследник TDataSet. В остальном методы работы с компонентом THtDsPump практически повторяют методы работы с компонентом THtPump. Убедиться в этом читателю предлагается самостоятельно.

## **Экспорт данных**

Набор компонент Delphi для работы с HyTech позволяет экспортировать данные из БД во внешний файл. Экспорт может быть выполнен в формате текстового файла и в формате DBF. Компоненты экспорта используют в качестве источника данных открытый набор данных (результат запроса на выборку данных), что существенно повышает гибкость операций экспорта по сравнению с прямым экспортом отдельно взятых таблиц из БД.

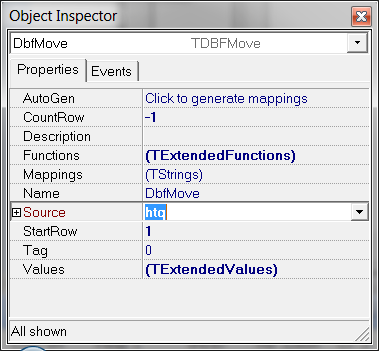
Для демонстрации возможностей HyTech по экспорту данных из клиентских приложений Delphi рассмотрим следующий пример. Усовершенствуем пример из пункта 2.3. добавив на главную форму приложения компоненты dbfMove (тип TDBFMove) и txtMove (тип TTXTMove) с закладки HyTech. Компонент dbfMove будет использован для экспорта данных в формате dbf, а компонент txtMove для экспорта данных в текстовом формате. Помимо указанных компонентов добавим на главную форму две кнопки bExportTxt («Экспорт TXT») и bExportDbf («Экспорт Dbf»). При нажатии на кнопку bExportTxt будет выполнен экспорт данных в текстовом формате, при нажатии кнопки bExportDbf будет выполнен экспорт данных в формате dbf соответственно.

Внешний вид формы в среде проектирования приведен на . Красным цветом на рисунке отмечены добавленные компоненты.



**Рис. 81. Внешний вид формы (среда разработки)**

Для выполнения экспорта требуется связать компоненты dbfMove и txtMove с источником данных. Для этого следует свойствам компонент dbfMove.Source и txtMove.Source присвоить значение htq (ссылка на источник данных). Htq имеет тип THtQuery и используется для формирования набора данных для экспорта. Пример присвоения свойства Source для компонента приведен на .



**Рис. 82. Редактирование свойства компонента (среда разработки)**

Коды обработчиков нажатия на кнопки bExportTxt («Экспорт TXT») и bExportDbf («Экспорт Dbf») приведены в .

Листинг

|  |
| --- |
| // экспорт в формате TXT  procedure TConnectForm.bExportTxtClick(Sender: TObject);  begin  txtMove.MoveTo('example.txt');  end;  // экспорт в формате DBF  procedure TConnectForm.bExportDbfClick(Sender: TObject);  begin  dbfMove.MoveTo('example.dbf');  end; |

Как можно видеть из исходного текста обработчиков для экспорта данных, в обеих компонентах используется метод MoveTo. В качестве параметра метода должен указываться файл экспорта. Для компонента TTXTMove при вызове метода MoveTo должен указываться файл с расширением txt, а для TDBFMove файл с расширением dbf. В нашем примере будем экспортировать данные в файлы example.dbf и example.txt соответственно.

Скомпилируем и запустим приложение. Установим связь с базой данных, нажмем на кнопку «Выполнить запрос» и выберем файл-источник данных ().



**Рис. 83. Внешний вид приложения, данные выбраны из БД и готовы для экспорта**

После нажатия кнопок «Экспорт TXT» и «Экспорт Dbf» в каталоге, откуда запущено приложение, будут созданы файлы example.txt и example.dbf. Эти файлы будут содержать экспортированные данные.

Следует иметь в виду, что при вызове метода MoveTo для компонент TDBFMove и TTXTMove в случае, если имя файла указано неправильно, не выполняется генерация исключения. Также при экспорте не генерируются исключения в случаях, когда отсутствует соединение с БД и запрос для выборки связанного с компонентом экспорта набора данных (htq в примере) не выполнен, в этой ситуации при экспорте будет создан пустой файл.

Имена полей и форматы в структуре создаваемого файла dbf совпадают с соответствующими полями связанного с компонентом экспорта набора данных (htq в примере). Кодировка символов в файле dbf соответствует кодировке DOS (кодовая страница 866).

При экспорте в формат txt в качестве символа-разделителя используется запятая. Строки разделяются парой символов «перевод каретки» и «возврат строки» (символы ASCII 13 и ASCII 10). В качестве кодировки символов по умолчанию используется кодировка Windows – кодовая страница 1251. Кодировку можно изменить, если установить свойство txtMove.OemText = true. В этом случае будет использоваться кодировка DOS (кодовая страница 866).

Если файл, в который производится экспорт, уже существует, то существующий файл удаляется, а вместо него создается новый с тем же именем.

Компоненты TDBFMove и TTXTMove позволяют экспортировать данные результирующего запроса не целиком, а частично. Для указания количества экспортируемых строк используется свойство CountRow (по умолчанию CountRow установлено в -1, что подразумевает экспорт всех строк, начиная с указанной). Для указания строки, начиная с которой выполняется экспорт, используется свойство StartRow (по умолчанию StartRow = 1).

## **Работа с макросами и препроцессором программного кода**

Препроцессор программного кода используется в клиенте Delphi для упрощения программирования запросов к БД в случаях, когда клиентскому приложению требуется частое использование «похожих» фрагментов программного кода в различных запросах. Первой такой возможностью препроцессора являются макросы. Выгода от применения макросов состоит в том, что похожие фрагменты программного кода могут быть заменены одним общим фрагментом кода с параметрами (программным кодом макроса). Подстановка в момент вызова макроса кода макроса и замена параметров на значения позволяет получить в тексте конкретного запроса нужный фрагмент кода. Использование макросов помогает сократить объем программного кода запросов, уменьшить количество ошибок, обеспечить более высокий уровень повторного использования программного кода.

Помимо обработки макросов препроцессор имеет дополнительные возможности, реализуемые при помощи так называемых прагм. Прагмы представляет собой идентификаторы, начинающиеся символом решетки (#). В зависимости от назначения прагмы препроцессор преобразует исходный код в соответствии с заданным набором правил. С этой точки зрения прагмы очень похожи на команды препроцессора других языков программирования (например, прагма #def аналогична конструкции #def языка C). Применение прагм, также как и макросов, позволяет сократить объем программного кода запросов, уменьшить количество ошибок, обеспечить более высокий уровень повторного использования программного кода.

Следует обратить внимание на то, что препроцессор предлагает исключительно клиентскую технологию. Сервер СУБД HyTech получает запрос, в котором вызовы макроса и прагмы уже заменены на соответствующий программный код.

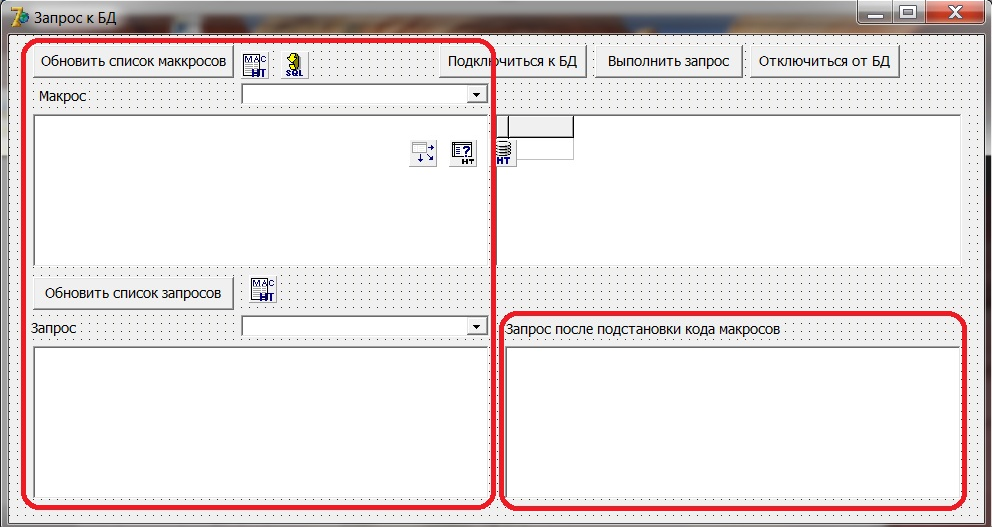
## **Работа с макросами**

Рассмотрим технику работы с макросами на примере. Для этого модифицируем приложение из раздела - добавим на главную форму ряд компонентов. Список всех необходимых компонентов главной формы представлен в таблице .(в список не добавлены компоненты типа TLabel с поясняющими подписями, они носят исключительно иллюстративный характер).

Таб. . Компоненты главной формы примера работы с макросами

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Название | Тип | Закладка | Статус | Назначение |
| 1. | htb | THtBase | HyTech | Без изменений | Осуществление связи с БД |
| 2. | htq | THtQuery | HyTech | Изменен текст запроса | Выполнение запроса к БД. Текст запроса содержит макрос |
| 3. | ds | TDataSource | DataAccess | Без изменений | Установка связи компонента htq с grd |
| 4. | grd | TDbGrid | DataControls | Без изменений | Отображение набора данных htq |
| 5. | bConnect | TButton | Standard | Без изменений | Установка связи с БД |
| 6. | bDisconnect | TButton | Standard | Без изменений | Отключение от БД |
| 7. | bSelect | TButton | Standard | Без изменений | Активация компонента htq. Выполнение запроса к БД |
| 8. | bReloadMacro | TButton | Standard | Добавлен | Загрузка библиотеки макросов (библиотека организуется компонентом MacList) |
| 9. | MacList | TMacrosList | HyTech | Добавлен | Организация библиотеки макросов |
| 10. | MacListCombo | TComboBox | Standard | Добавлен | Визуальное отображение списка макросов. Выбор текущего макроса. |
| 11. | MacroText | TMemo | Standard | Добавлен | Визуальное отображение программного кода текущего выбранного макроса |
| 12. | bReloadQuery | TButton | Standard | Добавлен | Загрузка списка тестовых запросов |
| 13. | QueryList | TMacrosList | HyTech | Добавлен | Организация списка тестовых запросов |
| 14. | QueryListCombo | TComboBox | Standard | Добавлен | Визуальное отображение списка тестовых запросов. Выбор текущего запроса |
| 15. | QueryText | TMemo | Standard | Добавлен | Визуальное отображение программного кода текущего выбранного тестового запроса |
| 16. | SqlParser | TSqlParser | HyTech | Добавлен | Компонент для преобразования кода исходного тестового запроса в код с подстановкой макросов и значений параметров |
| 17. | ResultText | TMemo | Standard | Добавлен | Визуальное отображение программного кода текущего выбранного тестового запроса после подстановки в текст кода макросов и значений параметров |

Внешний вид формы в среде проектирования приведен на . Красным цветом на рисунке отмечены добавленные компоненты.

****

ResultText

QueryText

MacroText

MacroListCombo

QueryListCombo

**Рис. 84. Внешний вид формы (среда разработки)**

Программный код обработчиков нажатия на кнопки и обработки событий выбора элементов в выпадающих списках приведен в .

Листинг

|  |
| --- |
| // Загрузка библиотеки макросов  procedure TConnectForm.bReloadMacroClick(Sender: TObject);  var i : integer;  begin  MacList.Active := false;  MacList.Active := true;  MacListCombo.clear();  for i := 0 to MacList.Names.Count - 1 do begin  MacListCombo.AddItem( MacList.Names[i], nil );  end;  MacListCombo.ItemIndex := 0;  MacListComboChange( self );  end;  // Выбор текущего макроса  procedure TConnectForm.MacListComboChange(Sender: TObject);  begin  // Отображение программного кода текущего выбранного макроса  MacroText.Text := MacList.Macro[ MacListCombo.Text ];  end;  // Загрузка списка тестовых запросов  procedure TConnectForm.bReloadQueryClick(Sender: TObject);  var i : integer;  begin  QueryList.Active := false;  QueryList.Active := true;  QueryListCombo.clear();  for i := 0 to QueryList.Names.Count - 1 do begin  QueryListCombo.AddItem( QueryList.Names[i], nil );  end;  QueryListCombo.ItemIndex := 0;  QueryListComboChange( self );  end;  // Выбор текущего тестового запроса  procedure TConnectForm.QueryListComboChange(Sender: TObject);  begin  // Отображение программного кода текущего тестового запроса  QueryText.Text := QueryList.Macro[ QueryListCombo.Text ];  // Применение к программному коду текущего тестового запроса макросов из  // библиотеки макросов  ResultText.Text := SqlParser.Parse('Parse example', QueryText.Text );  End; |

Работа с библиотекой макросов организуется при помощи компонента типа TMacrosList. В нашем примере этот компонент называется MacList. Библиотека макросов загружается из внешнего текстового файла специальной структуры. Имя внешнего файла задается в свойстве FileName (в примере свойство MacList.FileName = macro.sql). Загрузка содержимого файла выполняется при изменении свойства MacList.Active с false на true.

Файл библиотеки макросов имеет следующую структуру:

|  |
| --- |
| [Имя макроса 1]  тело макроса  [Имя макроса 2]  тело макроса  … |

Имя макроса в файле указывается в отдельной строке в квадратных скобках. Имя макроса должно быть «правильным» идентификатором, т.е. может содержать символы латинского алфавита, цифры и символ подчеркивания и не может начинаться с цифры. Идентификаторы - имена макросов могут содержать также символы кирилицы.

Тело макроса может содержать произвольное количество строк программного кода макроса (не менее одной), открывающая квадратная скобка первым символом строки не допускается.

Пример содержимого файла macro.sql приведен на .

|  |
| --- |
| [SQL\_MACROS\_1]  select %1 from %2;  [SQL\_MACROS\_2]  select field1 from test;  [SQL\_MACROS\_3]  field1, field2  [SQL\_MACROS\_4]  PROC1( %1 ); PROC2( %2 ); PROC3( %3 );  [AUTOR\_FIELD\_LIST]  b.cName cName, a.cFamName cAuthorFam |

**Рис. 85. Пример содержимого файла macro.sql**

В теле макроса могут быть указаны формальные параметры макроса. Для задания формального параметра используется символ процента и порядковый номер параметра вызова (от единицы до девяти) – см. на , например, тело макроса SQL\_MACROS\_1. Помимо формальных параметров в теле макроса могут использоваться другие инструкции препроцессора (см. далее).

После загрузки библиотеки макросов имена загруженных макросов становятся доступны через свойство Names компонента MacList. В примере список имен макросов визуализируется компонентом MacListCombo. Фрагмент кода по заполнению выпадающего списка MacListCombo приведен в .

Листинг

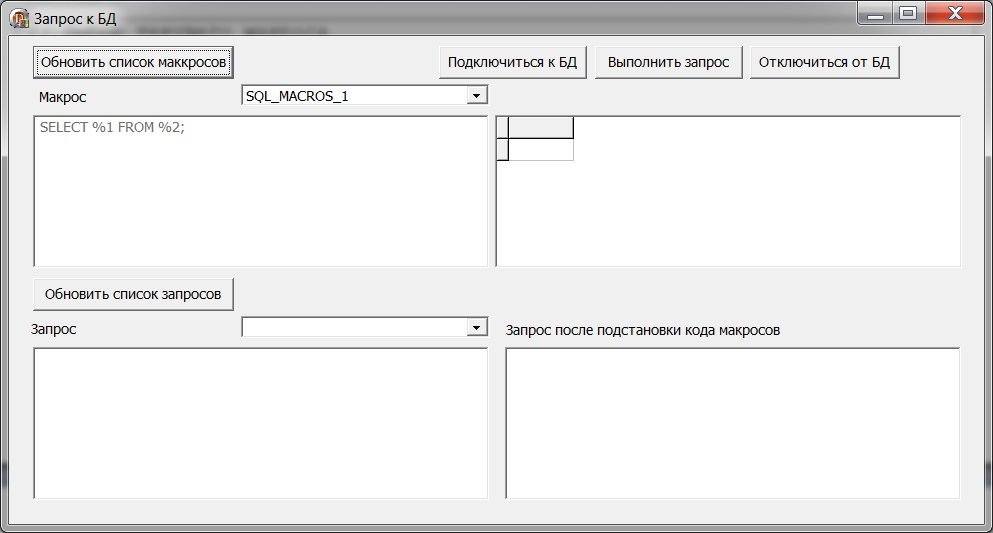
|  |
| --- |
| MacListCombo.clear();  for i := 0 to MacList.Names.Count - 1 do begin  MacListCombo.AddItem( MacList.Names[i], nil );  end;  MacListCombo.ItemIndex := 0; |

Тело макроса доступно через свойство MacList.Macro (ассоциативный массив, индексом которого для извлечения тела макроса должно являться переданое имя макроса). При изменении текущего элемента MacListCombo компонент MacroText визуализирует тело соответствующего макроса. Фрагмент кода по визуализации тела текущего макроса приведен в . Тело макроса также можно получить при вызове метода GetMacro компонента MacList. В качестве параметра в GetMacro должно быть передано имя макроса.

Листинг

|  |
| --- |
| // Выбор текущего макроса  procedure TConnectForm.MacListComboChange(Sender: TObject);  begin  // Отображение программного кода текущего выбранного макроса  MacroText.Text := MacList.Macro[ MacListCombo.Text ];  end; |

Скомпилируем и запустим приложение. Нажмем на кнопку «Обновить список макросов» ().



**Рис. 86. Внешний вид приложения, загружена библиотека макросов**

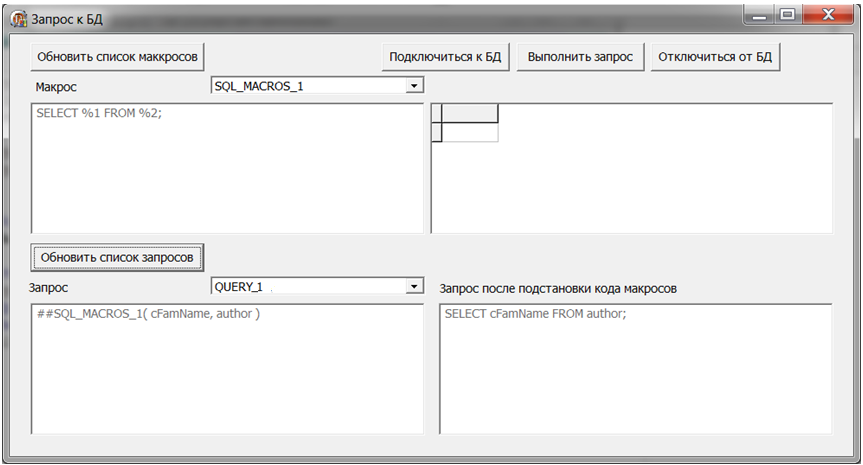
Для демонстрации использования макросов создадим примеры тестовых запросов и поместим их в файл query.sql (некоторые из них приведены ниже в таблице). Структуру файла сделаем полностью аналогичной macro.sql и воспользуемся для визуализации программного кода запросов теми же техническими приемами, что и для макросов. Выпадающий список QueryListCombo в итоге будет содержать названия тестовых запросов, а компонент QueryText отображать программный код текущего выбранного запроса.

Пример содержимого файла macro.sql приведен на .

|  |
| --- |
| [QUERY 1]  ##SQL\_MACROS\_1( cFamName, author )  [QUERY 2]  ##SQL\_MACROS\_2()  [QUERY 3]  SELECT ##SQL\_MACROS\_3() FROM TEST2  [QUERY 4]  SELECT##SQL\_MACROS\_3()FROM TEST2  [QUERY 5]  ##SQL\_MACROS\_4(, 2, 3 )  ##SQL\_MACROS\_4( 1,,3 )  ##SQL\_MACROS\_4( 1,2 )  [QUERY 6]  ##SQL\_MACROS\_1( (cFamName, substr(cFamName,1,1)), author )  [QUERY 7]  #def call( x1, x2 ) y = #x1 + #x2  #call( a, b );  [QUERY 8]  #def call c = a + b  #call;  [QUERY 9]  #def call( x1, x2 ) y = #x1 + #x2  #def call c = a + b  #call( a, b );  #call;  [QUERY 10]  #def call( x1, x2 ) y = #x1 #x2  #call( (a+c/), b );  [QUERY 11]  #def call( x1, x2 ) y = #x1#x2  #call( a+c, +b );  [QUERY 12]  #def call( x1, ... ) y = #x1 #0 #1 #2  #call( a+, b );  #call( a+, b+, c );  [QUERY 13]  #def A(p1, p2) select #p1 from #p2  #def B(p1, p2) where #p1 = #p2  #def AB(p1, p2, p3, p4) #A(#p1,#p2) #B(#p3,#p4)  #AB(ID, EIP, I\_TYPE, 2);  [QUERY 14]  #call( a+b, c );  [QUERY 15]  #!call( a+b, c );  [QUERY 16]  #def call y = x - z  z = "begin str "#"call"" end str";  [QUERY 17]  #def A(x) EIP#x  "gen"#"A(X)"\_EMP"  [QUERY 18]  #def call( x1, x2 ) y = #x1 + #x2  #"call"( a, b )  [QUERY 19]  #def call( x1, x2 ) y = #x1 + #x2  "aaaa"#"call"( a, b )"fff" |

**Рис. 87. Пример содержимого файла query.sql**

Нажмем в приложении на кнопку «Обновить список запросов» ().



**Рис. 88. Внешний вид приложения, загружены тестовые запросы**

На можно также видеть не только текст запроса, но и результат применения макроса (поле с заголовком «Запрос после подстановки кода макроса»). Для получения такого результата мы воспользовались препроцессором (компонент SqlParser). В приведен код обработки текста запроса препроцессором.

Листинг

|  |
| --- |
| // Применение к программному коду текущего тестового запроса макросов из  // библиотеки макросов  ResultText.Text := SqlParser.Parse('Parse example', QueryText.Text ); |

Исходный текст запроса QueryText.Text передается в качестве второго параметра метода SqlParser.Parse. Первый параметр SqlParser.Parse используется в качестве составной части сообщения об ошибке работы препроцессора (для нас данный параметр при рассмотрении примера не существенен). Результатом работы SqlParser.Parse является текст запроса, обработанный препроцессором.

При обработке исходного текста запроса, если в запросе встречается идентификатор, начинающийся двумя «диезами» (##), то препроцессор считает, что за двумя «диезами» следует имя макроса. За именем макроса в скобках через запятую должны следовать значения параметров, которые подставляются на место формальных параметров (напомним, что формальные параметры задаются через символ процента и порядковый номер параметра). Неправильное имя макроса приводит к генерации исключения (ошибка препроцессора).

Выделив в тексте запроса имя макроса, препроцессор ищет макрос с таким именем в библиотеке макросов. Имя библиотеки макросов (macro.sql) устанавливается в свойстве SqlParser.MacrosList (в примере это компонент MacList).

При написании тела макроса нужно иметь ввиду, что между символом процента и цифрой не должно быть пробела, иначе символ процента трактуется как обычный оператор целочисленного деления (HyTech SQL). Символ процента имеет смысл для препроцессора только при обработке тела макроса, а вне тела макроса игнорируется.

Применение макросов имеет ряд особенностей. Некоторые из них продемонстрируем на примерах ().

Таб. . Особенности применения макросов

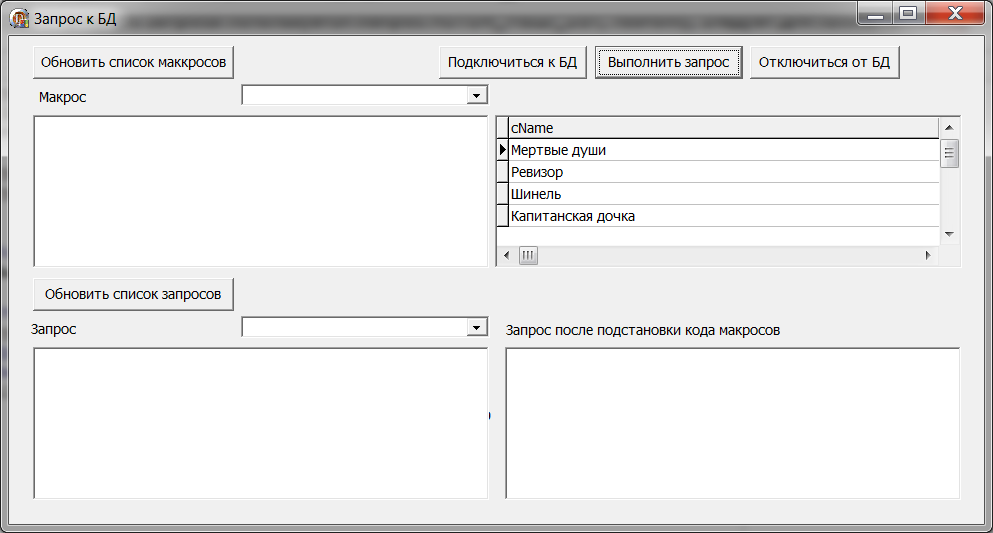
|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Особенности применения макросов | Исходный текст запроса | Текст запроса после обработки препроцессором |
| 1. | Макрос может не иметь параметров | ##SQL\_MACROS\_2() | select field1 from test; |
| 2. | Макрос может быть вставлен в произвольное место исходного текста | select ##SQL\_MACROS\_3() from TEST2 | select field1, field2 from TEST2; |
| 3. | В тексте запроса макрос не требуется выделять ни пробелами, ни каким либо другим способом. Препроцессор не проверяет синтаксис запроса с точки зрения HyTech SQL или его процедурного расширения | select##SQL\_MACROS\_3()from TEST2 | selectfield1, field2from TEST; |
| 4. | Значение параметра в тексте запроса может быть пропущено. В результирующем тексте оно заменяется на пробел | ##SQL\_MACROS\_4(, 2, 3 )  ##SQL\_MACROS\_4( 1,,3 )  ##SQL\_MACROS\_4( 1,2 ) | PROC1( ); PROC2( 2 ); PROC3( 3 );  PROC1( 1 ); PROC2( ); PROC3( 3 );  PROC1( 1 ); PROC2( 2 ); PROC3( ); |
| 5. | Если значение параметра указано в круглых скобках, оно может иметь внутри скобок любые символы, в том числе и запятые, при подстановке такого параметра круглые скобки убираются | ##SQL\_MACROS\_1( (cFamName, substr(cFamName,1,1)), author ) | select cFamName, substr(cFamName,1,1) from author; |

Препроцессор также может использоваться и без явного применения компонента SqlParser, например, в тексте SQL запроса в компоненте типа THtQuery. Для демонстрации такой возможности зададим в свойстве SQL компонента htq следующий запрос:

|  |
| --- |
| select ##AUTOR\_FIELD\_LIST()  from book b, author a  where b.nAuthorKey=a.nAuthorKey |

В тексте запроса используется макрос AUTOR\_FIELD\_LIST, поэтому следует для компонента htq указать библиотеку макросов для поиска макроса. Это можно сделать непосредственно для компонента htq, установив ему значение свойства MacrosList = macList, или для всех запросов по умолчанию, установив свойство MacrosList = macList для компонента htb.

Скомпилируем и запустим приложение. Установим соединение с БД и нажмем на кнопку «Выполнить запрос» ().



**Рис. 89. Внешний вид приложения, выполнен запрос с макросом**

Препроцессор подставил в текст запроса код макроса и выполнил следующий запрос:

|  |
| --- |
| select b.cName cName, a.cFamName cAuthorFam  from book b, author a  where b.nAuthorKey=a.nAuthorKey |

## **Использование прагм**

Помимо макросов при разработке программного кода запросов могут быть использованы прагмы. Прагма представляет собой идентификатор, начинающийся символом решетки (#). Прагма может иметь список параметров, обычно задаваемый в круглых скобках.

Определение прагмы должно начинаться с новой строки и заканчиваться символами окончания строки (обычно ASCII 13 и ASCII 10).

Основной и наиболее часто используемой прагмой является прагма #def.

Прагма #def имеет следующий синтаксис:

|  |
| --- |
| #def идентификатор [(список формальных параметров)] [выходной текст для замены] |

Прагма #def используется для задания конструкции подстановки. Если в исходном тексте встретится **#идентификатор** со списком фактических параметров в круглых скобках и **идентификатор** со списком формальных параметров ранее определен в прагме #def, то вместо такой конструкции препроцессор подставит **выходной текст для замены**. Вместо формальных параметров в **выходной текст запроса для замены** будут подставлены фактические параметры, переданные в круглых скобках после конструкции **#идентификатор**. Для того, чтобы препроцессор мог распознать формальные параметры **в выходном тексте для замены,** перед именем формального параметра должен использоваться символ решетки (#). В прагме #def последним формальным параметром может быть многоточие (…), указывающее, что фактических параметров на месте многоточия может быть несколько (или ни одного). Для обращения к фактическим параметрам «многоточия» используется символ решетки с последующим порядковым номером параметра в многоточии (#0, #1 и т.д.).

Идентификаторы могут содержать символы латинского алфавита, цифры и символ подчеркивания и не могут начинаться с цифры.

Применение прагмы #def имеет ряд особенностей. Проиллюстрируем их на примерах ().

Таб. . Особенности применения прагмы #def

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Особенности применения прагмы | Исходный текст запроса | Текст запроса после обработки препроцессором |
| 1. | Обычный #def с параметрами | #def call( x1, x2 ) y = #x1 + #x2  #call( a, b ); | y = a + b; |
| 2. | Описание прагмы может не иметь параметров | #def call c = a + b  #call; | c = a + b; |
| 3. | Нельзя использовать два #def с одним и тем же идентификатором, следующий #def с тем же идентификатором переопределяет предыдущий | #def call( x1, x2 ) y = #x1 + #x2  #def call c = a + b  #call( a, b );  #call; | Сообщение об ошибке |
| 4. | Если значение параметра указано в круглых скобках, оно может иметь внутри скобок любые символы, в том числе и запятые, при подстановке такого параметра круглые скобки убираются | #def call( x1, x2 ) y = #x1 #x2  #call( (a+c/), b ); | y = a+c/ b; |
| 5. | В тексте формальные параметры не требуется выделять ни пробелами, ни каким либо другим способом | #def call( x1, x2 ) y = #x1#x2  #call( a+c, +b ); | y = a+c+b; |
| 6. | Использование многоточия в списке формальных параметров | #def call( x1, ... ) y = #x1 #0 #1 #2  #call( a+, b );  #call( a+, b+, c ); | y = a+ b ;  y = a+ b+ c ; |
| 7. | В #def можно использовать идентификаторы, определенные в других #def | #def A(p1, p2) select #p1 from #p2  #def B(p1, p2) where #p1 = #p2  #def AB(p1, p2, p3, p4) #A(#p1,#p2) #B(#p3,#p4)  #AB(ID, EIP, I\_TYPE, 2); | select ID from EIP where I\_TYPE = 2; |
| 8. | Если в #def не определен идентификатор до момента обращения к нему (в примере #call), то вместо идентификатора с параметрами подставляется пустая строка | #call( a+b, c ); | Пустая строка |
| 9. | Использование модификатора обязательности.  Если в #def не определен идентификатор до момента обращения к нему (в примере #call) и в момент обращения использован модификатор (!) после (#), то возникает сообщение об ошибке. Если модификатор (!) используется, то он должен следовать сразу после (#) | #!call( a+b, c ); | Сообщение об ошибке |
| 10. | Идентификатор взят в кавычки: значение при подстановке тоже будет в тех же кавычках | #def call( x1, x2 ) y = #x1 + #x2  #«call»( a, b ) | «y = a + b» |
| 11. | Если значение прагмы нужно вставлять внутрь строковой константы, это записывается следующим образом:  "Часть строки до подстановки"#"pragma""часть строки после"  -- имя прагмы взято в те же кавычки и записано без пробелов между частями строки | #def call y = x - z  z = "begin str "#"call"" end str"; | z = "begin str y = x - z end str"; |

Применение других прагм препроцессора описано в справочнике по компонентам.

## **Работа с курсорами**

Ранее в курсе рассматривались типовые приемы работы с высокоуровневыми программными интерфейсами HyTech, ориентированными в первую очередь на традиционную идеологию работы с данными в Delphi. Эта идеология предполагает максимально возможное абстрагирование от специфики СУБД, использование единого подхода к работе с данными независимо от источника данных. В частности, базовым классом, предоставляющим интерфейсы для работы с данными, можно считать класс TDataSet, от которого наследуется широкий спектр специфических классов, ориентированных на конкретную СУБД. HyTech здесь не является исключением. Так от TDataSet наследуется класс THtDataSet, а от THtDataSet класс TCustomHtQuery, от которого, в свою очередь, подробно рассмотренный нами в предыдущих разделах THtQuery.

Хотя универсальных интерфейсов для работы с данными для большинства случаев достаточно, иногда может потребоваться умение использовать специальные программные интерфейсы СУБД HyTech. В частности, специальные интерфейсы СУБД HyTech могут потребоваться если:

* идеология TDataSet не может быть использована для доступа к данным по архитектурным соображениям, например, требуется использование ORM (Object Relational Mapping);
* для приложения установлены повышенные требования по производительности, реализуемые только посредством использования специальных интерфейсов СУБД;
* для приложений требуются возможности доступа к данным, не реализованные в существующих классах – наследниках TDataSet.

В данном разделе курса будут рассмотрены некоторые специальные интерфейсы СУБД HyTech, позволяющие организовать доступ к данным в ряде случаев более гибко, чем через использование классов, реализующих универсальные интерфейсы Delphi. Эти специальные интерфейсы основаны на использовании понятия курсора. Заметим, что рассматриваемые специальные интерфейсы являются достаточно низкоуровневыми и требуют от обучающегося понимания работы с указателями и управления памятью в Delphi.

Курсор – понятие, используемое для обозначения рабочей области памяти и связанных с ней ресурсов сервера. Доступ к данным выполняется через структуру данных курсора. В курсоре располагаются скомпилированный оператор SQL, коды завершения операций и различные системные переменные. Каждый курсор имеет номер, по которому к курсору ссылаются при вызове функций. По этому же номеру к курсору можно ссылаться в конструкции 'cursno #' в SQL-скриптах.

Для демонстрации возможностей по работе с курсорами из клиентских приложений Delphi рассмотрим следующий пример. Усовершенствуем пример из пункта 2.3., добавив на главную форму приложения компоненты, представленные в .

Таб. . Компоненты главной формы примера работы с курсорами

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Название | Тип | Закладка | Статус | Назначение |
|  | htb | THtBase | HyTech | Без изменений | Осуществление связи с БД, реализация интерфейсов работы с курсорами и других низкоуровневых интерфейсов HyTech |
|  | bConnect | TButton | Standard | Без изменений | Установка связи с БД |
|  | bDisconnect | TButton | Standard | Без изменений | Отключение от БД |
|  | SQLText | TMemo | Standard | Новый | Визуальное отображение текста SQL запроса для открытия курсора |
|  | ResultData | TMemo | Standard | Новый | Визуальное отображение результирующих данных, полученных из курсора, а также другой информации и диагностических сообщений |
|  | ListFieldsCombo | TComboBox | Standard | Новый | Список полей, соответствующих текущему открытому курсору |
|  | ListTablesCombo | TComboBox | Standard | Новый | Список таблиц, соответствующих текущему открытому курсору |
|  | eFind | TEdit | Standard | Новый | Визуальное отображение значения для поиска данных, добавления новой строки в курсор и изменения строки курсора |
|  | bQuery | TButton | Standard | Новый | Открытие курсора, выполнение выборки первичной порции данных и ее визуализация в ResultData |
|  | bShowInfo | TButton | Standard | Новый | Визуализация в ResultData информации об открытом курсоре |
|  | bShowTableInfo | TButton | Standard | Новый | Визуализация в ResultData информации о таблицах в открытом курсоре |
|  | bReadCursor | TButton | Standard | Новый | Чтение и визуализация в ResultData всего набора данных в курсоре |
|  | bFindRec | TButton | Standard | Новый | Поиск строки в наборе данных открытого курсора |
|  | bOrderCursor | TButton | Standard | Новый | Сортировка набора данных курсора по выбранному полю |
|  | bFreeCursor | TButton | Standard | Новый | Разрушение курсора. Освобождение ресурсов СУБД, связанных с курсором |
|  | bAddRec | TButton | Standard | Новый | Добавление строки в набор данных курсора и БД. Чтение и визуализация в ResultData всего набора данных в курсоре |
|  | bDelRec | TButton | Standard | Новый | Удаление строки из набора данных курсора и БД. Чтение и визуализация в ResultData всего набора данных в курсоре |
|  | bUpdRec | TButton | Standard | Новый | Модификация строки в наборе данных курсора и БД. Чтение и визуализация в ResultData всего набора данных в курсоре |

В не показаны компоненты, используемые для иллюстративных целей (поясняющие надписи).

В раздел private описания класса формы добавим текст, приведенный в .

Листинг

|  |
| --- |
| private  // ссылка на интерфейс курсора  htCursorBase: IHtCursor;  DataBuffer : PChar;  DataBufferLength : integer;  FieldInfo : array of PResultInfoRec;  TableInfo : array of PResultInfoRec;  procedure addQueryInfo( InfoRec: PResultInfoRec );  function execQuery(sql : string) : IHtCursor;  procedure freeResults;  function printCursorHeader : integer;  procedure printFirstResults;  procedure readCursorAllData;  function FindStr() : integer;  procedure changeStr(Buf : PChar; str : string); |

Внешний вид главной формы приложения представлен на .

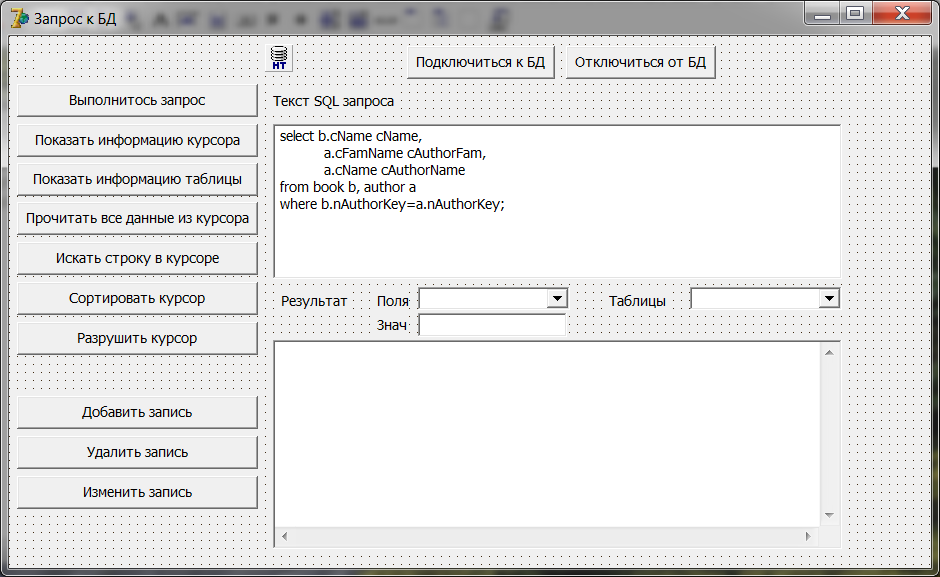


Рис. 90. Внешний вид главной формы приложения (среда разработки)

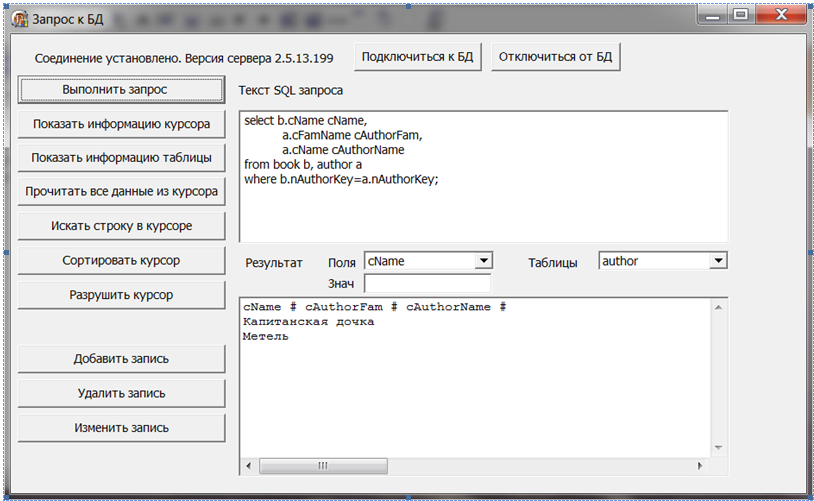
Кнопки «Подключится к БД» (bConnect) и «Отключится от БД» (bDisconnect) используются стандартным образом, также как и в других примерах данного курса.

Обратите внимание, что для реализации работы с курсорами в примере используется не только компонент из закладки HyTech – компонент htb (THtBase), но и структуры данных и интерфейсы, определенные в файлах HtApiDef.pas, HtDefs.pas. Эти файлы включены в конструкцию uses главной формы приложения.

Для удобства свойству Lines компонента SQLText присвоен текст запроса по умолчанию.

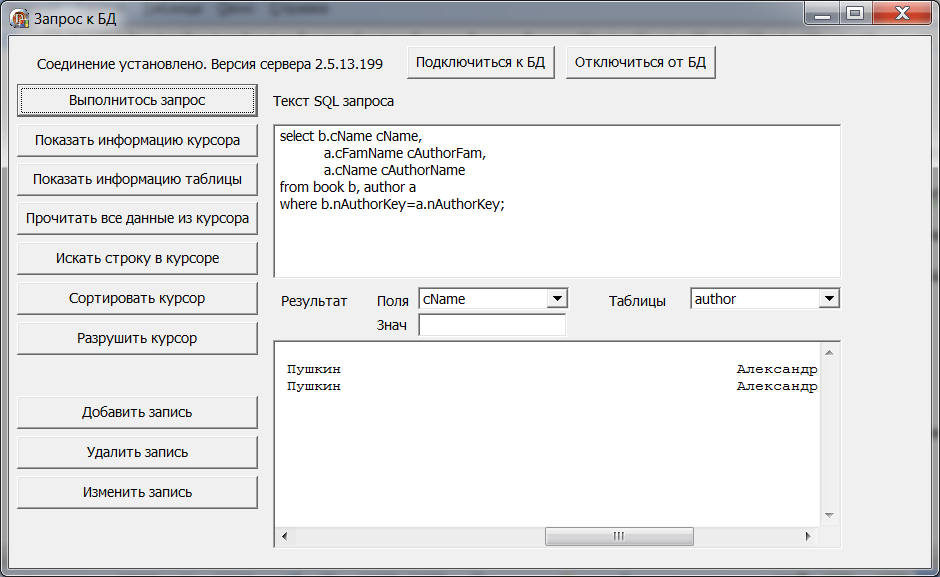
## **Создание курсора, выполнение запроса, открытие результата, получение первичной порции данных и разрушение курсора**

Скомпилируем и запустим приложение. Установим связь с базой данных, нажмем на кнопку «Выполнить запрос». Внешний вид приложения после нажатия кнопки «Выполнить запрос» представлен на .



**Рис. 91. Внешний вид приложения, нажата кнопка «Выполнен запрос»**

В результате нажатия кнопки «Выполнить запрос», открыт курсор, выполнен запрос, первая порция набора данных, связанного с курсором, прочитана и визуализирована в компоненте ResultData. Не должно смущать, что в наборе данных три поля, а на экране отображены данные только одного – это связано с шириной поля, если прокрутить ResultData вправо при помощи горизонтального скроллера, то можно увидеть и другие данные, см. .



**Рис. 92. Внешний вид приложения, нажата кнопка «Выполнен запрос», показаны дополнительные данные выборки**

Коды обработчика нажатия на кнопку bQuery («Выполнить запрос») приведен в .

Листинг

|  |
| --- |
| // открыть курсор, выполнить запрос  procedure TConnectForm.bQueryClick(Sender: TObject);  var i :integer;  begin  // очистить поле вывода результатов  ResultData.Text := '';  // закрыть ранее открытый курсор, освободить ресурсы  freeResults();  // выполнить запрос, вернуть ссылку на интерфейс доступа к курсору  htCursorBase := execQuery( SQLText.Text );  // визуализировать результаты execQuery  printFirstResults();  // заполнить выпадающие списки имен полей и имен таблиц информацией  // из курсора  ListFieldsCombo.Items.Clear;  ListTablesCombo.Items.Clear;  for i := 0 to Length(FieldInfo) - 1 do begin  ListFieldsCombo.Items.Add( FieldInfo[ i ].eName );  end;  for i := 0 to Length(TableInfo) - 1 do begin  ListTablesCombo.Items.Add( TableInfo[ i ].bName );  end;  // установить первую таблицу и первое поле в выпадающих списках  // по умолчанию  ListFieldsCombo.ItemIndex := 0;  ListTablesCombo.ItemIndex := 0;  end; |

В примере для удобства мы всегда будем работать с единственным курсором, соответственно создавая и разрушая при необходимости. В примере этот курсор доступен через переменную htCursorBase, ссылающуюся на интерфейс IHtCursor. Подробнее об использовании интерфейса IHtCursor далее.

Основным вызовом для выполнения запроса в примере является метод execQuery. Этот вызов возвращает ссылку на интерфейс IHtCursor открытого курсора (в случае успешного выполнения запроса). В качестве параметра в execQuery передается текст SQL запроса. Исходный текст метода execQuery приведен в .

Листинг

|  |
| --- |
| function TConnectForm.execQuery(sql: string) : IHtCursor;  var HtCurs: IHtCursor;  SqlToOEM, CalcOnly: Boolean;  InfoHandler: TInfoHandler;  CPR: TComplexParamRec; // определение типа находится в модуле HtBases  Param: PComplexParamRec; // определение типа находится в модуле HtBases  Statement: PChar;  str : string;  begin  // получить ссылку на интерфейс доступа к курсору  // будет создан класс доступа к курсору, реализующий IHtCursor  HtCurs := NewIHtCursor;  // через свойство HtCurs.CursorRec получаем  // ссылку на структуру THtCursor  // если нужно создать новый курсор при выполнении запроса, то  // Handle := -1  // после успешного выполнения запроса  // HtCurs.CursorRec.Handle будет содержать реальный номер открытого  // на сервере курсора  HtCurs.CursorRec.Handle := -1;  // Statement - указатель на текст SQL запроса  // Внимание! Statement должен быть определен как локальная переменная  // ссылка на параметр в htb.ComplexEval не сработает  str := sql;  Statement := PChar(str);  // дополнительные параметры вызова  CPR.DoCreateFields:= false;  CPR.Notifier:= nil;  CPR.CR.DataBytes:= 512; // сколько байт читать первой порцией  CPR.CR.ReadInfo:= true; // читать доп. информацию о курсоре  CPR.CR.CloseAfterRead:= false; // уничтожить курсор  // после чтения результатов  // ссылка на дополнительные параметры выполнения запроса  Param := @CPR;  // ссылка на функцию обработчик информации о вызове  InfoHandler := @CallBackSQLQuery;  // требуется преобразование текста SQL из кодировки Windows  // в кодировку OEM (DOS)  SqlToOEM := true;  // если CalcOnly = false помимо получения набора данных  // будет получена еще информация о  // таблицах и полях в запросе, по соображениям производительности  // можно установить CalcOnly = true, на запрос сервер будет тратить  // меньше времени  CalcOnly := false;  // открыть курсор, выполнить запрос, получить первую порцию данных  htb.ComplexEval(HtCurs,  SqlToOEM, CalcOnly,  Statement, InfoHandler, Param);  Result := HtCurs;  end; |

Создание курсора, выполнение в нем запроса, открытие результата, получение информации о результате и (если есть) первой порции данных выполняется при помощи вызова htb.ComplexEval. Весь остальной код метода - это практически задание параметров данного вызова. Рассмотрим, что представляют собой эти параметры. Для получения ссылки на интерфейс доступа к курсору используется вызов NewIHtCursor. Вызов NewIHtCursor – это вызов глобальной функции, определенной в HtApiDef.pas. Там же определен класс TIHtCursor, реализующий IHtCursor, NewIHtCursor создает экземпляр данного класса. Далее сведения о структуре класса TIHtCursor в примере никак не используются, поэтому структура класса TIHtCursor нами не описывается.

Через интерфейс IHtCursor (в примере получаем доступ через обращение к свойству HtCurs.CursorRec) доступна структура THtCursor с информацией о курсоре:

|  |
| --- |
| THtCursor = packed record  Handle : integer; {собственно курсор}  Records : integer; {записей в курсоре}  Aliases : integer; {хенделов в результате}  Columns : integer; {столбцов в результате}  RecLen : integer; {длина записи}  TypeRes : integer; {тип результата}  HtError : THtError;  SqlError: THtError;  RetCode : Variant;  end; |

После выполнения запроса эта структура будет содержать основные сведения о запросе. Дополнительные параметры вызова htb.ComplexEval задаются в переменной CPR, имеющей тип TComplexParamRec. Следует обратить внимание на параметр CPR.DataBytes, в котором задается максимальное число байт, читаемое в первой порции данных. Реально происходит чтение количества байт, кратное длине одной записи, возвращаемой курсором, но не больше заданного в CPR.CR.DataBytes (в структуре THtCursor после выполнения запроса длина одной записи будет записана в поле RecLen). Например, если CPR.CR.DataBytes = 512, а длина одной записи равна 200 байт, то будет считано 400 байт (2 записи).

Для получения информации в процессе обработки запроса htb.ComplexEval использует технику обратного вызова (CallBack). В htb.ComplexEval передается ссылка на функцию CallBackSQLQuery, которая вызывается по мере возникновения различных значимых событий в процессе обработки запроса. Исходный текст функции CallBackSQLQuery приведен в .

Листинг

|  |
| --- |
| // Функция обратного вызова при обработке запроса  function CallBackSQLQuery(InfoRec: PResultInfoRec; Param: Pointer): THtError; stdcall;  begin  ConnectForm.addQueryInfo( InfoRec );  Result := 0;  end; |

Функция CallBackSQLQuery не являтеся методом формы TConnectForm и соответствует типу, определенному в модуле HtDefs:

|  |
| --- |
| TInfoHandler = function(InfoRec: PResultInfoRec; Param: Pointer): THtError; stdcall; |

Функция CallBackSQLQuery передает для обработки ссылку InfoRec на структуру типа TResultInfoRec(также определенную в модуле HtDefs) в метод addQueryInfo. На структуре TResultInfoRec следует остановиться особо, поскольку она содержит ключевую информацию о полях, таблицах и данных, получаемых в результате запроса. TResultInfoRec представляет собой запись с вариантами и описывается следующим образом ():

Листинг

|  |
| --- |
| TResultInfoRec = packed record  DataType: TResultDataType;  case TResultDataType of  rdTable: (  bName : PChar; //Имя таблицы  bAlias : PChar; //Алиас таблицы (если инфо по курсору)  bRecLen : integer; //Длина записи таблицы  bFieldCnt : SmallInt; //Число полей  bGroupCnt : SmallInt; //Число групп  bSubstrCnt : SmallInt; //Число подстрок  bConstCnt : LongInt; //Число записей в постоянной части таблицы  bAddedCnt : LongInt; //Число добавленных записей  bChangedCnt: LongInt; //Число измененных записей  bDeletedCnt: LongInt; //Число удаленных записей  );  rdElement: (  eName : PChar; //Имя элемента (или Алиас - "as")  eAlias : PChar;  //Алиас таблицы элемента (если инфо по курсору)  eSize : integer; //Длина данных элемента  eOffset : integer; //Смещение элемента в записи  eType : integer; //Тип элемента  eClass : integer; //Класс элемента:  // EC\_FIELD = 0 - поле  // EC\_GROUP = 1 - группа  // EC\_SUBSTR = 2 - подстрока  eKey : integer; //Тип ключа  // EK\_NOTKEY = 0; не ключ  // EK\_COMMON = 1; ключ  // EK\_UNIQUE = 2; Уникальный ключ  // EK\_SURRGT = 3; Суррогатный ключ  // модификаторы ключа  eNull : boolean;  eAuto : boolean;  eFieldNo : integer;  //порядковый номер поля в результате (если инфо по курсору)  eOrder : integer; //порядковый номер поля в таблице  // доп. инфо по элементу  eFieldName: PChar;  //Имя поля (если элемент - поле и инфо по курсору)  );  rdData: (  Data : PChar; //указатель на буфер с данными  Bytes : integer; //длина буфера с данными  );  rdGetCalcName: (  cName : PChar;  // Имя для internal calc поля (заполняется в обработчике)  cCalcNo : integer; // номер п/п  cHandled: boolean; // признак, что имя назначено  );  end; |

Использование TResultInfoRec при выполнении запроса поясняет метод addQueryInfo. Исходный текст метода addQueryInfo приведен в .

Листинг

|  |
| --- |
| // обработка данных, поступивших при выполнении запроса  procedure TConnectForm.addQueryInfo( InfoRec: PResultInfoRec );  var FieldCount, TableCount : integer;  ptr : PResultInfoRec;  begin  if InfoRec^.DataType = rdElement then begin  // получена информация о поле  // добавляем новый элемент в массив FieldInfo  FieldCount := Length( FieldInfo );  SetLength( FieldInfo, FieldCount + 1 );  // выделяем память под новую структуру описания поля  GetMem( ptr, SizeOf( TResultInfoRec ) + 2 );  FieldInfo[ FieldCount ] := ptr;  // копируем поля из InfoRec в структуру описания поля  ptr.eName := StrNew( InfoRec^.eName );  ptr.eAlias := StrNew( InfoRec^.eAlias );  ptr.eSize := InfoRec^.eSize;  ptr.eOffset := InfoRec^.eOffset;  ptr.eType := InfoRec^.eType;  ptr.eClass := InfoRec^.eClass;  ptr.eKey := InfoRec^.eKey;  ptr.eNull := InfoRec^.eNull;  ptr.eAuto := InfoRec^.eAuto;  ptr.eFieldNo := InfoRec^.eFieldNo;  ptr.eOrder := InfoRec^.eOrder;  ptr.eFieldName := StrNew( InfoRec^.eFieldName );  end else if InfoRec^.DataType = rdData then begin  // получена порция данных  DataBufferLength := InfoRec^.Bytes;  if InfoRec^.Bytes > 0 then begin  // выделяем память под порцию первичных данных  GetMem( DataBuffer, InfoRec^.Bytes + 16 );  // преобразовываем кодировку данных + копируем данные в  // выходной буфер  // Внимание!  // преобразование кодировки используется исключительно для  // целей последующей визуализации буфера данных  OemToAnsi(InfoRec^.Data, DataBuffer );  end;  end else if InfoRec^.DataType = rdTable then begin  // получена информация о таблице  // добавляем новый элемент в массив TableInfo  TableCount := Length( TableInfo );  SetLength( TableInfo, TableCount + 1 );  // выделяем память под новую структуру описания таблицы  GetMem( ptr, SizeOf( TResultInfoRec ) + 2 );  TableInfo[ TableCount ] := ptr;  // копируем поля из InfoRec в структуру описания таблицы  ptr.bName := StrNew( InfoRec^.bName );  ptr.bAlias := StrNew( InfoRec^.bAlias );  ptr.bRecLen := InfoRec^.bRecLen;  ptr.bFieldCnt := InfoRec^.bFieldCnt;  ptr.bGroupCnt := InfoRec^.bGroupCnt;  ptr.bSubstrCnt := InfoRec^.bSubstrCnt;  ptr.bConstCnt := InfoRec^.bConstCnt;  ptr.bAddedCnt := InfoRec^.bAddedCnt;  ptr.bChangedCnt := InfoRec^.bChangedCnt;  ptr.bDeletedCnt := InfoRec^.bDeletedCnt;  end;  end; |

Метод addQueryInfo реагирует на три типа событий при выполнении запроса: поступление информации о таблице в курсоре, поступление информации о поле в курсоре и получение первичных данных.

Вернемся разделу private класса TConnectForm и поясним назначение ряда полей. Для того, чтобы этой информацией можно было воспользоваться по завершению выполнения запроса, в примере созданы структуры для хранения полученной информации:

|  |
| --- |
| DataBuffer : PChar;  DataBufferLength : integer;  FieldInfo : array of PResultInfoRec;  TableInfo : array of PResultInfoRec; |

Переменные DataBuffer и DataBufferLength предназначены для хранения порции первичных данных, считанных в результате запроса, и длины этих данных в байтах соответственно. Массив FieldInfo используется для хранения информации о полях текущего открытого курсора. Массив TableInfo используется для хранения информации о таблицах текущего открытого курсора.

Следует обратить внимание, что при добавлении в DataBuffer, FieldInfo и TableInfo данных мы вынуждены выполнять копирование поступившей в InfoRec информации, а не просто на нее ссылаться, поскольку разработчиком судьба InfoRec после передачи данных в функцию обратного вызова никак не специфицирована, и мы не можем полагаться на то, что эти данные не будут уничтожены после возврата из функции обратного вызова.

Таким образом, мы рассмотрели детали вызова метода execQuery. Вернемся к программному коду обработчика нажатия на кнопки bQuery. После вызова execQuery следует вызов метода printFirstResults. Исходный текст метода printFirstResults приведен в .

Листинг

|  |
| --- |
| // визуализировать результаты execQuery  procedure TConnectForm.printFirstResults;  var i, len :integer;  ptr : PChar;  begin  // визуализировать список имен полей в запросе в виде заголовка  len := printCursorHeader();  if DataBufferLength > 0 then begin  i := 0;  ptr := StrAlloc( len + 1 );  ptr[ len ] := chr(0);  while i < DataBufferLength do begin  // скопировать порцию данных, соответствующую очередной  // записи  StrLCopy( ptr, @DataBuffer[i], len );  ResultData.Lines.Add( ptr );  i := i + len;  end;  StrDispose( ptr );  end;  end;  // визуализировать список имен полей в запросе в виде заголовка  function TConnectForm.printCursorHeader : integer;  var i, len :integer;  begin  len := 0;  for i := 0 to Length(FieldInfo) - 1 do begin  // добавить в заголовок имя поля  ResultData.Text := ResultData.Text + FieldInfo[ i ].eName + ' # ';  len := len + FieldInfo[ i ].eSize;  end;  Result := len;  end; |

Метод printFirstResults считывает данные из DataBuffer порциями, соотвествующими длине записи, и отображает их в виде очередной строки ResultData. Метод printCursorHeader используется для визуализации списка имен полей в запросе в виде заголовка и возвращает длину записи. Можно видеть, что для получения списка имен полей printCursorHeader использует массив FieldInfo.

В коде обработчика нажатия на кнопки bQuery нами остался не рассмотрен только вызов метода freeResults. Исходный текст метода freeResults приведен в .

Листинг

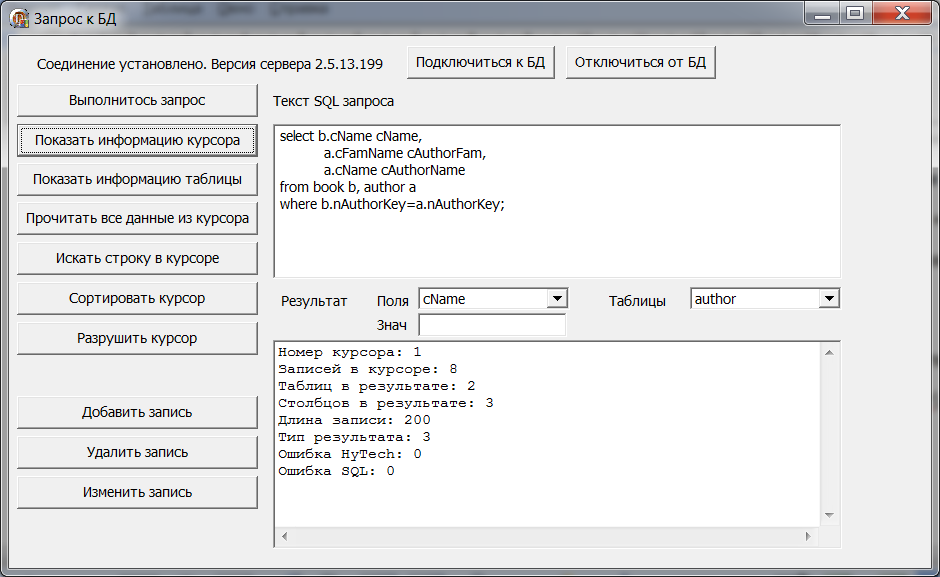
|  |
| --- |
| // закрыть ранее открытый курсор, освободить ресурсы  procedure TConnectForm.freeResults;  var i : integer;  begin  if Length(FieldInfo) > 0 then begin  for i := 0 to Length(FieldInfo) - 1 do begin  StrDispose( FieldInfo[ i ].eName );  StrDispose( FieldInfo[ i ].eAlias );  StrDispose( FieldInfo[ i ].eFieldName );  FreeMem( FieldInfo[ i ] );  end;  SetLength( FieldInfo, 0 );  end;  if Length(TableInfo) > 0 then begin  for i := 0 to Length(TableInfo) - 1 do begin  StrDispose( TableInfo[ i ].bName );  StrDispose( TableInfo[ i ].bAlias );  FreeMem( TableInfo[ i ] );  end;  SetLength( TableInfo, 0 );  end;  if DataBufferLength > 0 then begin  FreeMem( DataBuffer );  end;  DataBufferLength := 0;  // освободить (разрушить) курсор  htb.FreeCursor(htCursorBase);  end; |

Метод freeResults используется для закрытия ранее открытого курсора и освобождения связанных с курсором ресурсов. В тексте метода в основном мы удаляем данные, полученные в функции обратного вызова при выполнении запроса, но также следует обратить внимание на вызов htb.FreeCursor, который освобождает связанные с курсором ресурсы. После данного вызова для повторного использования курсор должен быть создан заново.

Заметим, что в нашем примере метод freeResults будет также вызван в качестве единственного действия в обработчике при нажатии на кнопку «Разрушить курсор» (bFreeCursor).

## **Получение информации о курсоре и информации о таблице**

После запуска приложения, соединения с БД, нажатия на кнопку «Выполнить запрос» информация по текущему открытому курсору доступна для чтения. Нажмем на кнопку «Показать информацию курсора» (bShowInfo). Внешний вид приложения будет следующий .



**Рис. 93. Внешний вид главной формы приложения после нажатия кнопки «Показать информацию**

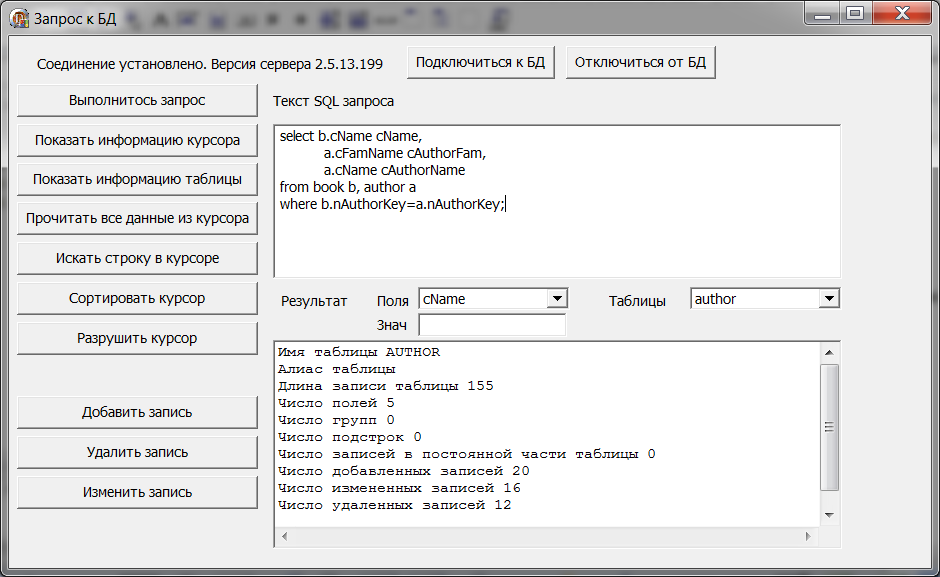
Исходный текст обработчика нажатия на кнопку «Показать информацию курсора» приведен в .

Листинг

|  |
| --- |
| // визуализировать информацию о текущем открытом курсоре  procedure TConnectForm.bShowInfoClick(Sender: TObject);  var  CursorPtr : PHtCursor;  begin  // получить ссылку на структуру данных с информацией о курсоре  CursorPtr := htCursorBase.GetCursorRec();  ResultData.Lines.Clear;  ResultData.Lines.Add( 'Номер курсора: ' + IntToStr(CursorPtr.Handle) );  ResultData.Lines.Add( 'Записей в курсоре: ' + IntToStr(CursorPtr.Records) );  ResultData.Lines.Add( 'Таблиц в результате: ' + IntToStr(CursorPtr.Aliases) );  ResultData.Lines.Add( 'Столбцов в результате: ' + IntToStr(CursorPtr.Columns) );  ResultData.Lines.Add( 'Длина записи: ' + IntToStr(CursorPtr.RecLen) );  ResultData.Lines.Add( 'Тип результата: ' + IntToStr(CursorPtr.TypeRes) );  ResultData.Lines.Add( 'Ошибка HyTech: ' + IntToStr(CursorPtr.HtError) );  ResultData.Lines.Add( 'Ошибка SQL: ' + IntToStr(CursorPtr.SqlError) );  end; |

Для получения ссылки на структуру данных с информацией о курсоре используется метод GetCursorRec интерфейса IHtCursor. В качестве альтернативы можно воспользоваться свойством CursorRec интерфейса IHtCursor – результат будет тот же.

Часто в приложениях требуется получить информацию о конкретной таблице. Это легко сделать для открытого курсора (в нашем примере информация обо всех таблицах, используемых в курсоре, содержится в массиве TableInfo, работу с которым мы уже ранее рассматривали), однако, не всегда удобно открывать курсор, особенно, если требуется только информация о таблице и никакие другие данные не нужны. В таких случаях удобнее воспользоваться специальным вызовом интерфейса HyTech. Продемонстрируем это на примере. Нажмем в нашем приложении кнопку «Показать информацию таблицы» (bShowTableInfo). Внешний вид приложения будет следующий ().



**Рис. 94. Внешний вид главной формы приложения после нажатия кнопки «Показать информацию таблицы»**

Исходный текст обработчика нажатия на кнопку «Показать информацию таблицы» приведен в .

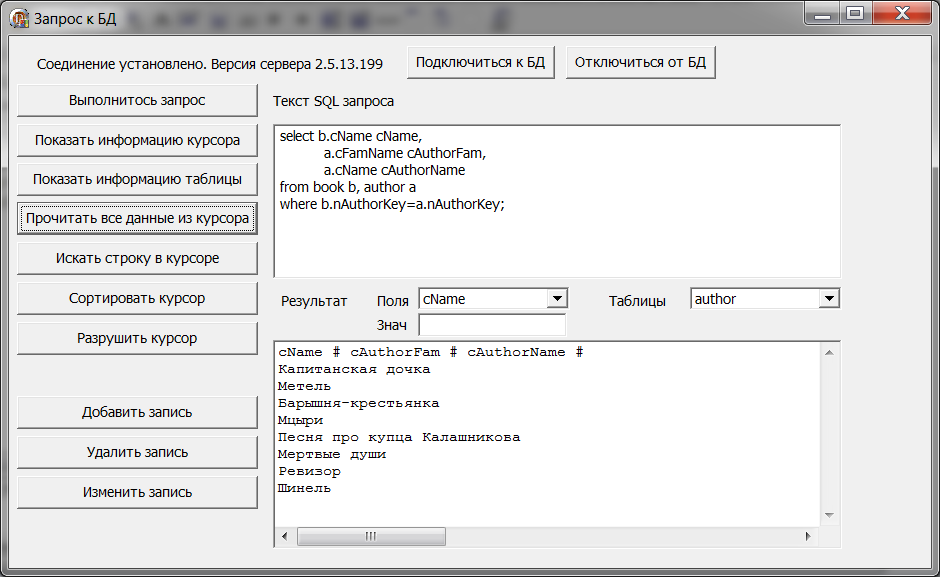
Листинг

|  |
| --- |
| // получить информацию таблицы  procedure TConnectForm.bShowTableInfoClick(Sender: TObject);  // функция обратного вызова для получения информации по таблице  // Внимание!  // эта функция должна быть определена как вложенная  procedure TableInfoProc(InfoRec: PResultInfoRec);  begin  case InfoRec^.DataType of  rdTable :  begin  ResultData.Lines.Add( 'Имя таблицы ' + InfoRec^.bName );  ResultData.Lines.Add( 'Алиас таблицы ' + InfoRec^.bAlias );  ResultData.Lines.Add( 'Длина записи таблицы ' +  IntToStr( InfoRec^.bRecLen ));  ResultData.Lines.Add( 'Число полей ' +  IntToStr( InfoRec^.bFieldCnt ));  ResultData.Lines.Add( 'Число групп ' +  IntToStr( InfoRec^.bGroupCnt ));  ResultData.Lines.Add( 'Число подстрок ' +  IntToStr( InfoRec^.bSubstrCnt ));  ResultData.Lines.Add( 'Число записей в постоянной части таблицы ' + IntToStr( InfoRec^.bConstCnt ));  ResultData.Lines.Add( 'Число добавленных записей ' +  IntToStr( InfoRec^.bAddedCnt ));  ResultData.Lines.Add( 'Число измененных записей ' +  IntToStr( InfoRec^.bChangedCnt ));  ResultData.Lines.Add( 'Число удаленных записей ' +  IntToStr( InfoRec^.bDeletedCnt ));  end;  rdElement: ;  end;  end;  begin  ResultData.Lines.Clear;  // получить информацию таблицы  htb.GetTablesInfo(ListTablesCombo.Items[ListTablesCombo.ItemIndex],  @TableInfoProc);  end; |

Обработчик нажатия кнопки использует вызов метода htb.GetTablesInfo. В качестве первого параметра в метод передается имя выбранной в выпадающем списке ListTablesCombo таблицы. Данный параметр может задавать несколько таблиц. В этом случае в символьной строке, передаваемой в качестве параметра, их имена следует перечислить через точку с запятой или другой разделитель (например, пробел). Вторым параметром вызова htb.GetTablesInfo является ссылка на функцию обратного вызова (TableInfoProc). Эта функция вызывается в момент получения информации о таблице или поле таблицы (в примере нас интересует информация только о таблице). Функция TableInfoProc имеет в качестве параметра ссылку на уже известную нам структуру данных TResultInfoRec.

## **Чтение данных, поиск данных в курсоре и сортировка курсора**

При нажатии на кнопку «Выполнить запрос» мы получили только первую порцию данных из набора данных открытого курсора. Эта порция данных может включать все запрашиваемые данные, но в общем случае это не так. В реальных приложениях может возникнуть необходимость чтения произвольно выбранных записей набора данных. Для демонстрации такой возможности прочитаем все записи из набора данных открытого курсора и визуализируем их. Для этого запустим приложение, соединимся с БД, выполним запрос нажатием на кнопку «Выполнить запрос», при этом откроем курсор и получим первую порцию данных и нажмем кнопку «Прочитать все данные из курсора» (bReadCursor). Внешний вид приложения будет следующий ().



**Рис. 95. Внешний вид главной формы приложения после нажатия кнопки «Прочитать все данные из курсора»**

Как можно видеть, на экран выведено значительно больше записей (все записи набора данных), чем при получении первой порции данных. Посмотрим, как реализована эта возможность в приложении. Исходный текст обработчика нажатия на кнопку «Прочитать все данные из курсора» приведен в .

Листинг

|  |
| --- |
| // обработчик нажатия на кнопку "Прочитать все данные из курсора"  procedure TConnectForm.bReadCursorClick(Sender: TObject);  begin  readCursorAllData();  end;  // Прочитать все данные из курсора  procedure TConnectForm.readCursorAllData;  var RowCount, i : Longint;  CursorPtr : PHtCursor;  RR: Pointer;  str : PChar;  begin  CursorPtr := htCursorBase.GetCursorRec();  // получаем количество строк в курсоре  RowCount := CursorPtr.Records;  // выделяем память для чтения строки  GetMem( RR, CursorPtr.RecLen + 1 );  ResultData.Lines.Clear;  // визуализировать список имен полей  // в запросе в виде заголовка  printCursorHeader();  // для каждой строки  for i := 0 to RowCount - 1 do begin  // Первая строка имеет 0-ой номер  // Читаем i-ую запись в буфер  htb.SeekRead( htCursorBase, i, RR, CursorPtr.RecLen );  // визуализируем запись в компоненте ResultData  str := PChar( RR );  str[ CursorPtr.RecLen ] := chr(0);  OemToAnsi( str, str );  ResultData.Lines.Add( str ); end; // освобождаем память, выделенную под чтение строки FreeMem( RR );end; |

Для получения общего числа записей и размера одной записи в курсоре воспользуемся уже известной нам структурой данных THtCursor. Для получения указателя на нее используется вызов htCursorBase.GetCursorRec(). Число записей содержится в поле Records структуры THtCursor, а длина записи в поле RecLen. Для чтения строки используется вызов метода htb.SeekRead. В качестве параметров в SeekRead передается указатель для доступа к интерфейсу курсора, номер читаемой записи, указатель на буфер для чтения записи и длина читаемой записи.

Следует обратить внимание, что в примере метод SeekRead читает записи по одной, однако, метод SeekRead имеет возможность чтения нескольких записей за один вызов. Для этого следует выделить память не под одну запись, а под несколько записей и задать в качестве последнего параметра длину выделенного буфера для чтения нескольких записей. Эта длина будет соответствовать максимальному количеству читаемых байт. Как и в случае чтения первичной порции данных, SeekRead прочитает кратное число записей, суммарная длина которых не превышает заданный в вызове размер буфера. Например, если размер буфера 1024 байт, а размер записи 300 байт, то будет прочитано только 900 байт (3 записи).

Следующей интересной возможностью работы с курсорами является поиск данных в курсоре. В приложении выберем из выпадающего списка полей поле, по которому будет выполняться поиск (пусть это будут cName), и введем значение для поиска (пусть это будет строка «Мет»). Нажмем кнопку «Искать строку в курсоре» (bFindRec). Внешний вид приложения после нажатия кнопки будет следующий ().

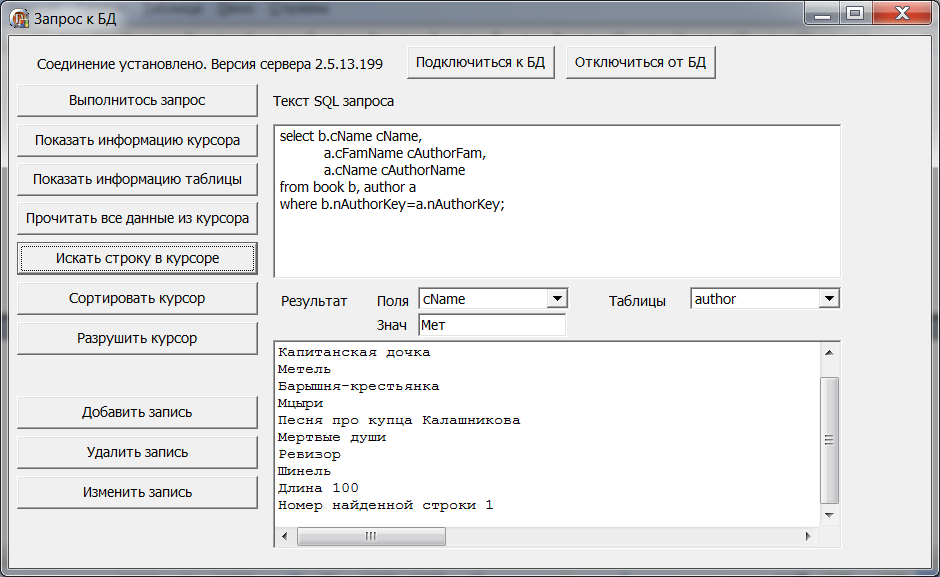


Рис. 96. Внешний вид главной формы приложения после нажатия кнопки «Искать строку в курсоре»

Как можно видеть, на экране в поле визуализации результатов выведен результат поиска «Номер найденной строки 1». Действительно, первые 3 символа значения поля cName = «Метель» и эталонного значения для поиска «Мет» совпадают. При нумерации записей с 0 найденная запись с cName = «Метель» имеет номер 1. Рассмотрим реализацию поиска в нашем примере. Исходный текст обработчика нажатия на кнопку «Искать строку в курсоре» приведен в .

Листинг

|  |
| --- |
| // обработчик нажатия кнопки "Искать строку в курсоре"  procedure TConnectForm.bFindRecClick(Sender: TObject);  var FoundRow : integer;  begin  FoundRow := FindStr();  ResultData.Lines.Add( 'Номер найденной строки ' +  IntToStr( FoundRow ));  end;  // Поиск строки в курсоре по содержимому  function TConnectForm.FindStr() : integer;  var FoundRow : integer;  Dst: Pointer;  Data : PChar;  Data2 : PChar;  s : string;  begin  // получаем строку для поиска  s:= eFind.Text;  Data := PChar( s );  // выделяем под строку область памяти и преобразуем ее в  // кодировку Oem  GetMem( Data2, length(s) + 1);  AnsiToOem( Data, Data2 );  // выделяем память для чтения результирующей записи  GetMem( Dst, htCursorBase.GetCursorRec().RecLen + 8 );  // ищем запись  FoundRow := htb.FindRecordInCursor( htCursorBase,  ListFieldsCombo.ItemIndex, // номер поля !поля нумеруются с 0  Data2, // эталон для сравнения  Dst, // буфер для чтения результирующей записи  length(s) // количество символов для сравнения  );  // Внимание!!! FoundRow = 0 в том числе и при ненайденной записи  // требуется чтение и сравнение  // В случае достижения последней строки данные в Dst  // могут не совпадать с эталоном  FreeMem( Dst );  FreeMem( Data2 );  Result := FoundRow;  end; |

Заметим, что в примере мы ищем совпадение по символьной строке (этим в частности обусловлено преобразование значения для поиска к кодировке OEM), однако, поиск выполняется путем побайтного сравнения значения поля с эталоном, поэтому искать можно по полю любого типа. Поиск выполняется при помощи вызова метода htb.FindRecordInCursor. В качестве параметров в FindRecordInCursor передаются: указатель для доступа к интерфейсу курсора, номер поля в курсоре для поиска (поля нумеруются с 0), указатель на буфер, куда помещено эталонное значение для сравнения, указатель на буфер, куда будет считана найденная запись (переменная Dst) и количество байт, которое сравнивается. FindRecordInCursor просматривает записи курсора последовательно, начиная с 0-ой, и сравнивает заданное число байт для заданного поля с эталоном. Заметим, что можно указать большее количество байт, чем длина поля, таким образом, сравниваться с эталоном могут значения нескольких смежных полей. При обнаружении первого совпадения с эталоном поиск останавливается, в буфер Dst копируется найденная запись и метод FindRecordInCursor возвращает номер найденной записи.

После выполнения вызова FindRecordInCursor успешность поиска требует уточнения (в примере не рассмотрено, в качестве упражнения обучающемуся предлагается дополнить пример соответствующей функциональностью). Так, например, если запись не найдена, FindRecordInCursor возвращает 0, 0 также будет возвращен FindRecordInCursor, если найдена запись с 0-ым номером. Кроме того, в случае достижения последней записи при поиске данные в Dst могут не совпадать с эталонным значением для поиска.

В завершающей части раздела рассмотрим возможности сортировки результатов запроса в курсоре. В приложении выберем в качестве поля для сортировки в выпадающем списке поле cName и нажмем кнопку «Сортировать курсор» (bOrderCursor). Внешний вид приложения после нажатия кнопки будет следующий ().

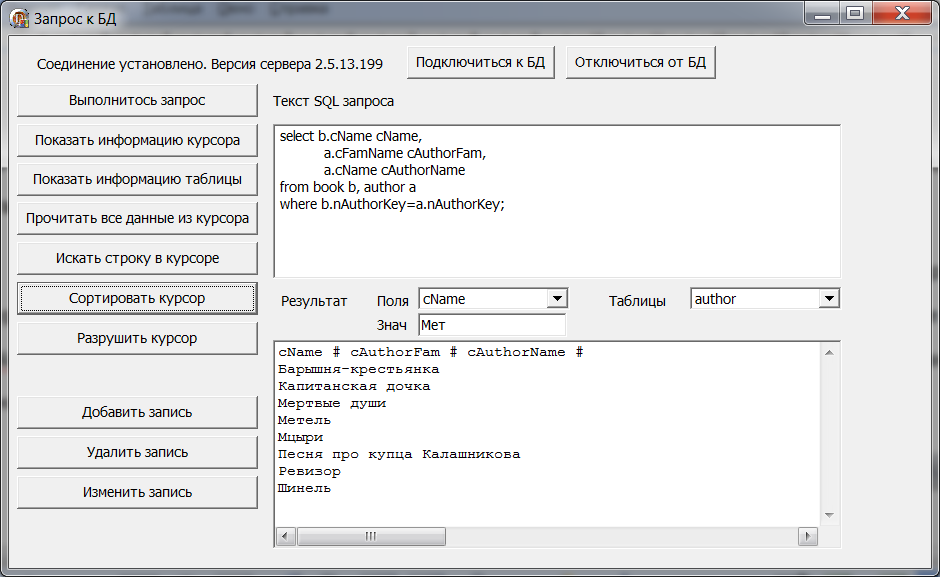


Рис. 97. Внешний вид главной формы приложения после нажатия кнопки «Сортировать курсор»

Как можно видеть, на экране записи упорядочены по значению поля cName. Рассмотрим реализацию сортировки. Исходный текст обработчика нажатия на кнопку «Сортировать курсор» приведен в .

Листинг

|  |
| --- |
| // упорядочить курсор по полю  procedure TConnectForm.bOrderCursorClick(Sender: TObject);  var  Flds: TAFieldNo; // описание типа приведено в модуле HtDefs  begin  // будем упорядочивать по единственному полю  SetLength( Flds, 1 );  // Столбцы нумеруются с 1  Flds[0] := ListFieldsCombo.ItemIndex + 1;  // сортировать курсор  htb.CursorOrder(htCursorBase, Flds);  ResultData.Lines.Clear;  // визуализируем отсортированные данные  readCursorAllData();  end; |

Сортировка записей курсора выполняется вызовом htb.CursorOrder. В качестве параметров в CursorOrder передаются указатель для доступа к интерфейсу курсора и массив номеров полей для сортировки. Поля нумеруются с 1. В нашем примере мы сортируем по единственному полю. Заметим, что сортировка может быть выполнена только по реальному полю таблицы (например, если в результате есть вычисляемое при помощи функции поле, то сортировка не производится). Также заметим, что если по какому-либо полю требуется обратная сортировка, то указывается его отрицательный номер.

Для возможности выполнения сортировки при открытии курсора должно быть соблюдено еще одно условие - в вызове:

|  |
| --- |
| htb.ComplexEval(HtCurs,  SqlToOEM, CalcOnly,  Statement, InfoHandler, Param); |

параметр CalcOnly должен иметь значение false.

## **Добавление, удаление, изменение записи в курсоре**

Помимо рассмотренных в предыдущих разделах возможностей работы с курсорами важными функциями интерфейса HyTech по работе с курсорами являются функции добавления, удаления и изменения записей в курсоре. При добавлении, удалении и изменении записей в курсоре HyTech также выполняет добавление, удаление и изменение записей в таблицах БД, таким образом, информация курсора и таблиц БД остается согласованной. Для демонстрации добавления, удаления и изменения записей на примере нам удобно будет воспользоваться другим SQL-запросом:

|  |
| --- |
| fix all;  select cFamName, cName from author; |

Добавление, удаление и изменение записей мы будем выполнять в таблице author и нам удобно в дальнейшем просматривать данные именно этой таблицы. В приложении введем указанный выше текст запроса, нажмем последовательно на кнопки «Выполнить запрос» и «Прочитать все данные из курсора». Внешний вид приложения после нажатия кнопки будет следующий ().

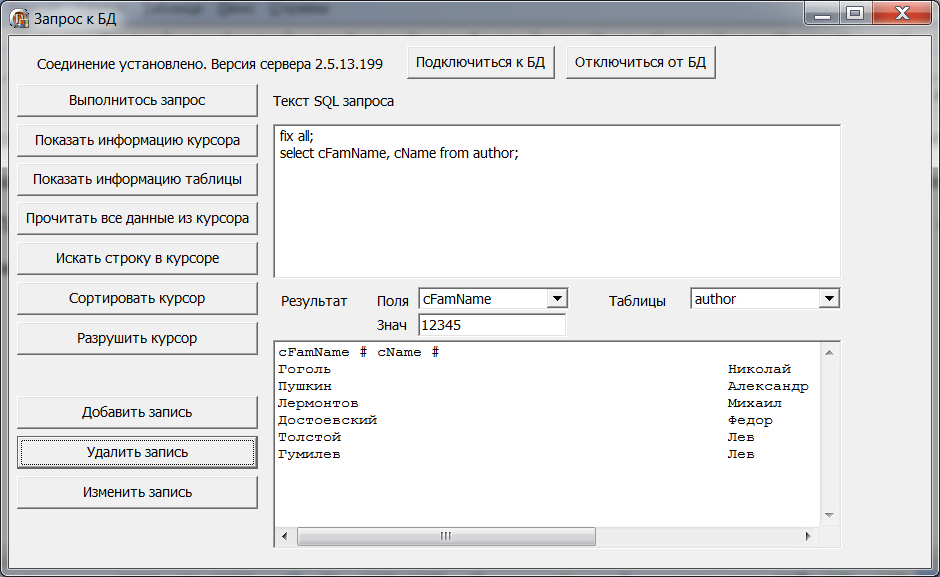


Рис. 98. Внешний вид главной формы приложения после изменения текста запроса

Введем в поле «Знач» значение поля cFamName (пусть это будет «12345») и нажмем кнопку «Добавить запись» (bAddRec). После добавления записи внешний вид приложения будет следующий ().

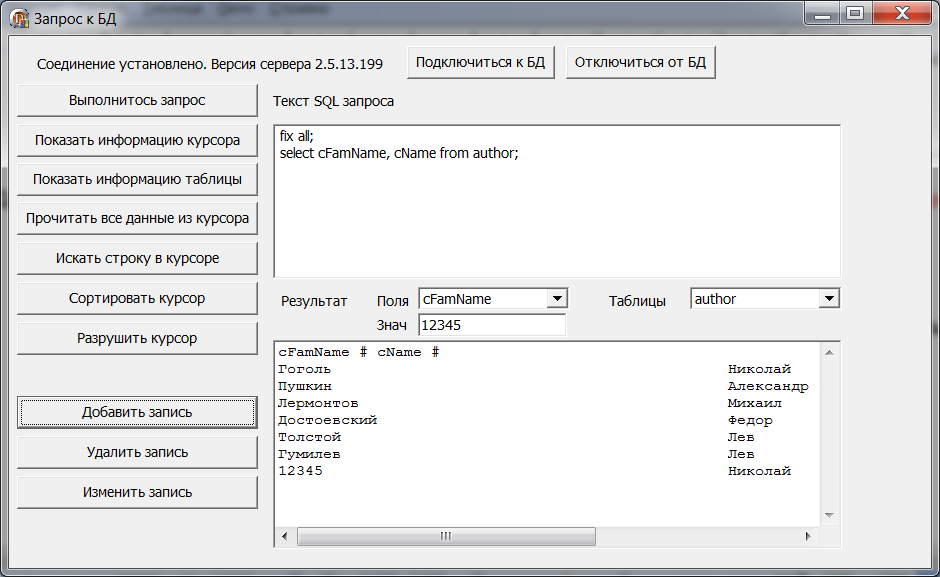


Рис. 99. Внешний вид главной формы приложения после нажатия кнопки «Добавить запись»

Как можно видеть из рисунка, в курсор добавлена новая запись и поле cFamName в этой новой записи имеет значение «12345». Можно убедиться, что новая запись добавлена не только в курсор, но и в таблицу БД author. Рассмотрим, каким образом данная функциональность реализована в примере. Исходный текст обработчика нажатия на кнопку «Добавить запись» приведен в .

Листинг

|  |
| --- |
| // добавить запись в курсор и таблицу БД  procedure TConnectForm.bAddRecClick(Sender: TObject);  var  CursorPtr : PHtCursor;  Row : Longint;  Buf : Pointer;  begin  // закрываем ранее открытый курсор  freeResults();  // открываем курсор, данные которого можно менять  // обращаем внимание на for update,  // без этого работать не будет!  htCursorBase := execQuery( 'fix all; select \* from ' +  ListTablesCombo.Items[ListTablesCombo.ItemIndex] + ' for update;' );  CursorPtr := htCursorBase.GetCursorRec();  // выбираем в качестве значения нового первичного ключа  // количество записей + 1  Row := CursorPtr.Records + 1;  // выделяем память под буфер новой записи  GetMem( Buf, CursorPtr.RecLen + 1 );  // читаем первую запись в буфер  htb.SeekRead( htCursorBase, 0, Buf, CursorPtr.RecLen );  // первичному ключу присваиваем значение  dword( Buf^ ) := Row;  // меняем в буфере значение поля на введенное пользователем  changeStr(Buf, eFind.Text);  // вставляем запись в курсор перед первой записью  htb.InsertRec( htCursorBase, 0, Buf );  // освобождаем буфер с содержимым записи  FreeMem( Buf );  // начитываем курсор с новыми данными  // визуализируем данные  bQueryClick( Self );  bReadCursorClick( Self );  end; |

Не в любом курсоре может быть выполнено добавление/удаление или изменение данных. Для реализации такой возможности текст запроса должен удовлетворять определенным требованиям. Во-первых, выборка должна выполняться из единственной таблицы. Во-вторых, текст запроса должен содержать конструкцию «for update» или конструкцию «order by». В примере, в итоге, мы открываем курсор с запросом, удовлетворяющий таким требованиям:

|  |
| --- |
| select \* from author for update; |

Заметим, что добавление и изменение записей в примере носит не универсальный характер, а ориентировано только на цели демонстрации. Сложности в реализации универсального механизма связаны, в основном, с необходимостью корректного заполнения буфера новой/измененной записи. Такая функциональность требует учета особенностей типов полей HyTech, занимает довольно большой объем программного кода и, потому, выходит за рамки учебного примера.

В примере новую запись создадим из уже существующей. Для этого выделим память под запись и прочитаем первую запись из курсора (метод htb.SeekRead). Чтобы отличить новую запись от существующей, изменим в записи первичный ключ (будем предполагать, что он имеет тип dword и находится в самом начале буфера записи), а также значение одного из полей, имеющего строковый тип (вызов метода changeStr).

Добавление записи выполняется вызовом htb.InsertRec. В качестве параметров в InsertRec передаются: указатель для доступа к интерфейсу курсора, номер записи в курсоре, перед которой будет вставлена запись, и буфер с новой записью.

Приведем также исходный текст метода changeStr():

Листинг

|  |
| --- |
| // вставить в буфер содержимое строки  // вставка выполняется для выбранного в списке поля  procedure TConnectForm.changeStr(  Buf : PChar; // буфер для вставки значения  str : string); // строка для вставки  var  i : integer;  begin  // ищем по имени поле в списке полей  for i := 0 to Length(FieldInfo) - 1 do begin  if FieldInfo[ i ].eName =  ListFieldsCombo.Items[ ListFieldsCombo.ItemIndex ] then begin  // нашли поле, FieldInfo[ i ].eOffset задает  // смещение в буфере в байтах, с этого смещения начинается  // значение поля  // копируем значение  StrLCopy( @Buf[ FieldInfo[ i ].eOffset ], PChar(str), length(str) );  break;  end;  end;  end; |

Введем в поле «Знач» значение поля cFamName (пусть это будет «12345-CHANGE») и нажмем кнопку «Изменить запись» (bUpdRec). После изменения записи внешний вид приложения будет следующий ().

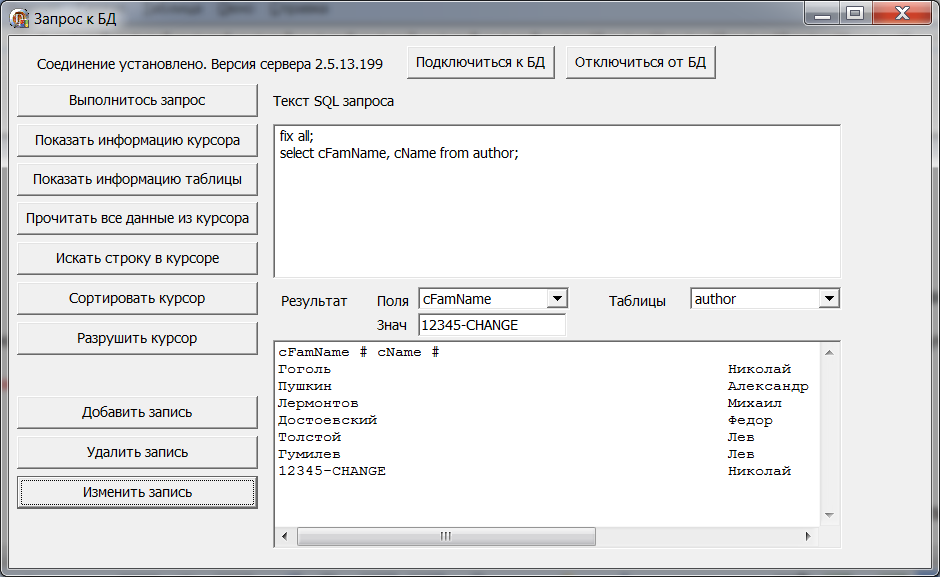


Рис. 100. Внешний вид главной формы приложения после нажатия кнопки «Изменить запись»

Как можно видеть из рисунка, в курсоре значение последней записи изменено и поле cFamName в этой новой записи имеет значение «12345-CHANGE». Можно убедиться, что новая запись изменена не только в курсоре, но и в таблице БД author. Рассмотрим, каким образом данная функциональность реализована в примере. Исходный текст обработчика нажатия на кнопку «Изменить запись» приведен в .

Листинг

|  |
| --- |
| // изменить запись в курсоре и таблице БД  procedure TConnectForm.bUpdRecClick(Sender: TObject);  var  CursorPtr : PHtCursor;  Row : Longint;  Buf : Pointer;  begin  // закрываем ранее открытый курсор  freeResults();  // открываем курсор, данные которого можно менять  // обращаем внимание на for update,  // без этого работать не будет!  htCursorBase := execQuery( 'fix all; select \* from ' +  ListTablesCombo.Items[ListTablesCombo.ItemIndex] + ' for update;' );  CursorPtr := htCursorBase.GetCursorRec();  // устанавливаем в качестве изменяемой записи последнюю  Row := CursorPtr.Records - 1;  // выделяем память под буфер изменяемой записи  GetMem( Buf, CursorPtr.RecLen + 1 );  // читаем изменяемую запись в буфер  htb.SeekRead( htCursorBase, Row, Buf, CursorPtr.RecLen );  // меняем в буфере значение поля на введенное пользователем  changeStr(Buf, eFind.Text);  // изменяем запись в курсоре и таблице  htb.UpdateRec( htCursorBase, Row, Buf, true );    // освобождаем буфер с содержимым записи  FreeMem( Buf );  // начитываем курсор с новыми данными  // визуализируем данные  bQueryClick(Self);  bReadCursorClick(Self);  end; |

Исходный текст обработчика во многом похож на исходный текст обработчика при добавлении записи. В примере изменять будем последнюю строку в наборе данных. Для этого выделим память под запись и прочитаем последнюю запись из курсора (метод htb.SeekRead). Далее в буфере изменим значение одного из полей, имеющего строковый тип (вызов метода changeStr).

Изменение записи выполняется вызовом htb.UpdateRec. В качестве параметров в UpdateRec передаются: указатель для доступа к интерфейсу курсора, номер изменяемой записи в курсоре, буфер с изменяемой записью и параметр блокировки. Если параметр блокировки установлен в значение true, то перед изменением записи выполняется ее блокировка, а после выполнения изменений в таблице блокировка снимается.

И, наконец, для удаления записи в приложении нажмем кнопку «Удалить запись» (bDelRec). После удаления записи внешний вид приложения будет следующий ().

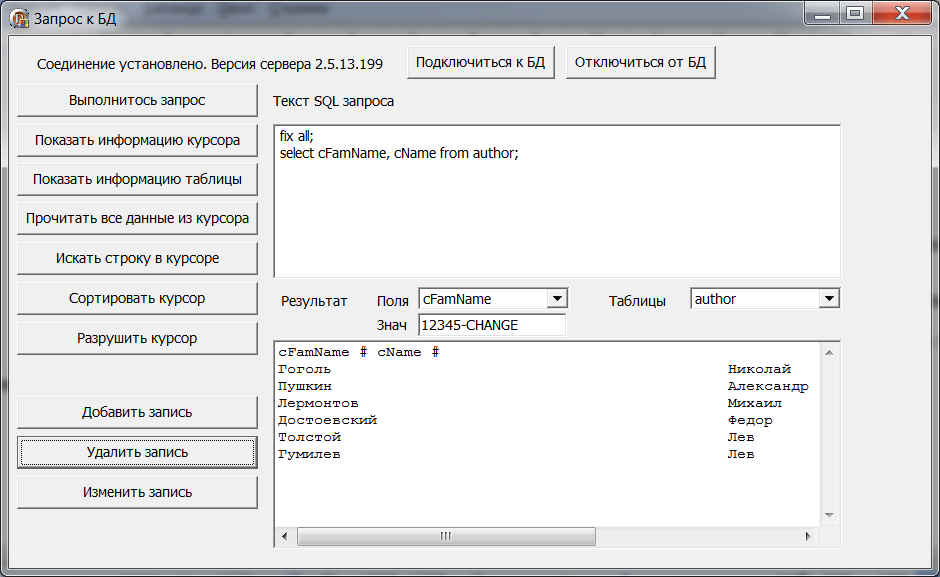


Рис. 101. Внешний вид главной формы приложения после нажатия кнопки «Удалить запись»

Исходный текст обработчика нажатия на кнопку «Удалить запись» приведен в .

Листинг

|  |
| --- |
| // удалить запись из курсора и таблицы БД  procedure TConnectForm.bDelRecClick(Sender: TObject);  begin  // закрываем ранее открытый курсор  freeResults();  // открываем курсор, данные которого можно менять  // обращаем внимание на for update,  // без этого работать не будет!  htCursorBase := execQuery( 'fix all; select \* from ' +  ListTablesCombo.Items[ListTablesCombo.ItemIndex] + ' for update;' );  // удаляем последнюю строку  htb.DeleteRec(htCursorBase, htCursorBase.GetCursorRec().Records - 1 );  // начитываем курсор с новыми данными  // визуализируем данные  bQueryClick(Self);  bReadCursorClick(Self);  end; |

Удаление записи выполняется вызовом htb.DeleteRec. В качестве параметров в DeleteRec передаются указатель для доступа к интерфейсу курсора и номер удаляемой записи. В примере удаляется последняя запись в наборе.

В заключении заметим, что работа с курсорами из клиента HyTech в Дельфи не исчерпывается перечисленными возможностями. Для изучения других возможностей следует обратиться к справочнику по компонентам (приводится в данном курсе) или к документации. Также следует обратить внимание, что примеры, приводимые в данном разделе, для их практического использования нуждаются в серьезном дополнении. В частности, «за кадром» остались вопросы обработки ошибок.

# Компоненты для визуализации

## **Работа с MS Excel**

Автоматизация работы с MS Excel часто требуется в информационных системах, например, для экспорта данных или формирования отчетов. В связи с наличием такой потребности компонент работы с Excel также включен разработчиками в набор компонент HyTech для работы с Дельфи. Работа компонента с Excel выполняется на основе довольно распространенного для задач интеграции приложений в среде Windows протокола DDE – Dynamic Data Exchange (часть технологии COM). Хотя протокол DDE и считается устаревшим, Microsoft обеспечивает поддержку этого протокола в современных версиях своих офисных приложений и на уровне API Windows. Рассматриваемый нами компонент работы с Excel позволяет управлять Excel в полном объеме API, предоставляемом Excel внешним приложениям (см. метод exec компонента). Сам по себе API Excel довольно объемный и в наши цели не входит его описание. API Excel опубликован Microsoft, можно, например, ознакомиться с ним на официальном сайте.

Далее мы рассмотрим только основные возможности компонента работы с Excel и продемонстрируем их на примере. Усовершенствуем пример из пункта 2.3. добавив на главную форму приложения новые компоненты. Перечень всех компонент приложения и их назначение приведены в .

Таб. . Компоненты главной формы примера работы с Excel

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Название | Тип | Закладка | Статус | Назначение |
|  | htb | THtBase | HyTech | Без изменений | Осуществление связи с БД |
|  | htq | THtQuery | HyTech | Без изменений | Выполнение запроса к БД. Данные результирующего набора в дальнейшем будут экспортированы в Excel |
|  | ds | TDataSource | DataAccess | Без изменений | Установка связи компонента htq с grd |
|  | grd | TDbGrid | DataControls | Без изменений | Отображение набора данных htq |
|  | bConnect | TButton | Standard | Без изменений | Установка связи с БД |
|  | bDisconnect | TButton | Standard | Без изменений | Отключение от БД |
|  | bSelect | TButton | Standard | Без изменений | Активация компонента htq. Выполнение запроса к БД |
|  | ExcelComp | TExcel | HtComp | Добавлен | Компонент для работы с Excel |
|  | bConnectExcel | TButton | Standard | Добавлен | Установка связи с Excel |
|  | bExportExcel | TButton | Standard | Добавлен | Выполнение экспорта в Excel с использованием прямого вызова функции передачи данных в область ячеек |
|  | bExport2 | TButton | Standard | Добавлен | Выполнение экспорта в Excel из DataSet (встроенная функциональность компонента) |
|  | bExport3 | TButton | Standard | Добавлен | Выполнение экспорта в Excel из Grid – табличной формы визуализации набора данных (встроенная функциональность компонента) |
|  | bGetRegion | TButton | Standard | Добавлен | Получение данных из области ячеек Excel, визуализация данных компонентом mResults |
|  | bExec | TButton | Standard | Добавлен | Выполнение произвольной команды Excel |
|  | bBatch | TButton | Standard | Добавлен | Выполнение пакетной операции передачи данных в Excel |
|  | bDisconnectExcel | TButton | Standard | Добавлен | Отключение от Excel |
|  | mResults | TMemo | Standard | Добавлен | Вывод результатов импорта данных из Excel и отладочная печать |

Внешний вид формы в среде проектирования приведен на .

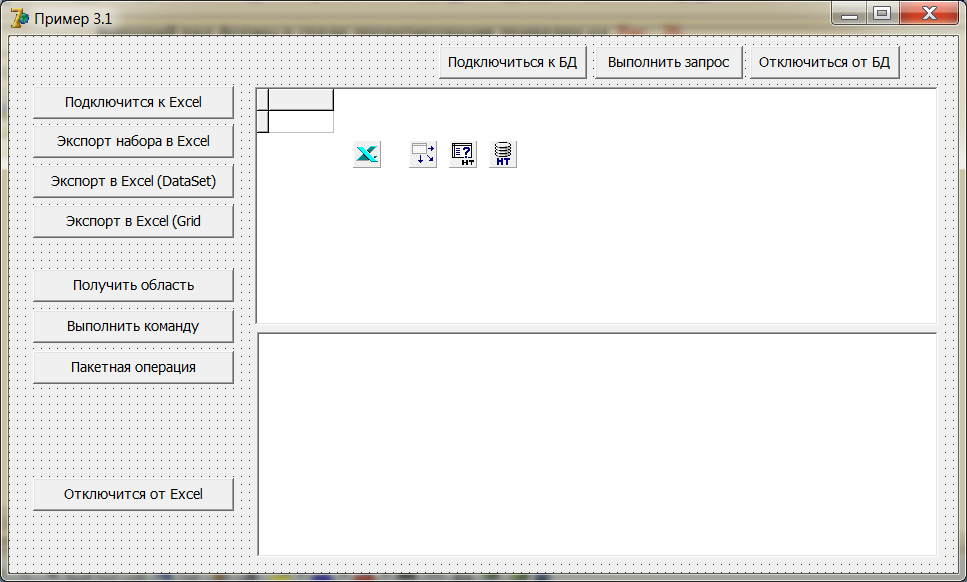


Рис. 102. Внешний вид формы (среда разработки)

Скомпилируем и запустим приложение. Установим связь с базой данных, нажмем на кнопку «Выполнить запрос» и нажмем кнопку «Подключится к Excel» (). Программный код запроса мы оставили прежним.

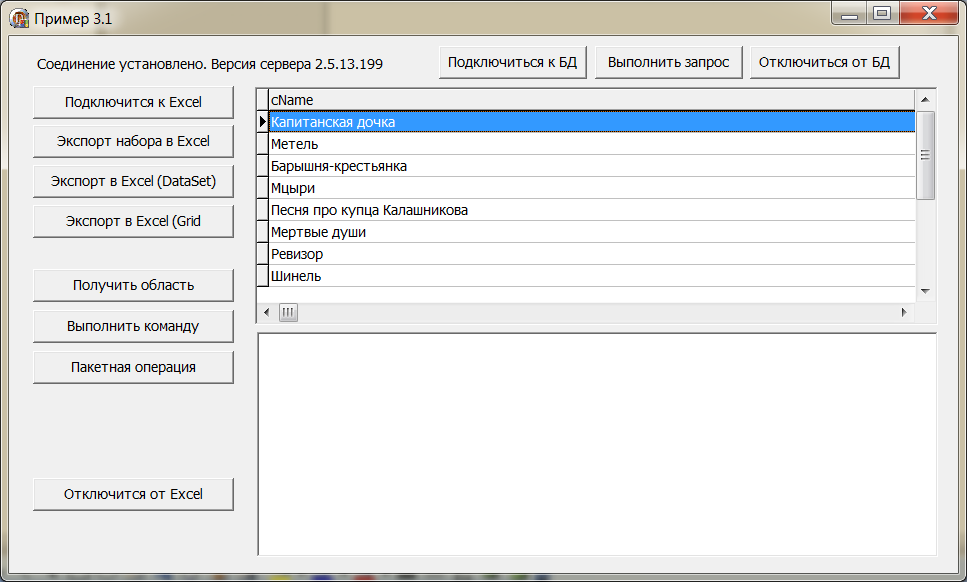


Рис. 103. Внешний вид формы (среда разработки)

После нажатия кнопки «Подключится к Excel» он будет загружен, если не был загружен ранее. Далее Excel готов принимать команды нашего приложения. Завершение работы с Excel выполняется нажатием кнопки «Отключится от Excel». Обратите внимание, что при этом завершения работы Excel не происходит. Коды обработчиков нажатия на кнопки bConnectExcel («Подключится к Excel») и bDisConnectExcel («Отключится от Excel») приведены в .

Листинг

|  |
| --- |
| // подключение к Excel  procedure TConnectForm.bConnectExcelClick(Sender: TObject);  begin  ExcelComp.Connect;  end;  // отключение от Excel  procedure TConnectForm.bbDisconnectExcelClick(Sender: TObject);  begin  ExcelComp.DisConnect;  end; |

И подключение, и отключение выполняется при помощи единственного вызова: ExcelComp.Connect и ExcelComp.DisConnect соответственно.

Обычно типовая работа информационной системы с Excel сводится к тому или иному варианту приема/передачи данных из ячеек/в ячейки Excel. Далее рассмотрим, какие возможности здесь предоставляет компонент. Нажмем кнопку «Экспорт набора в Excel». Результат экспорта представлен на ().

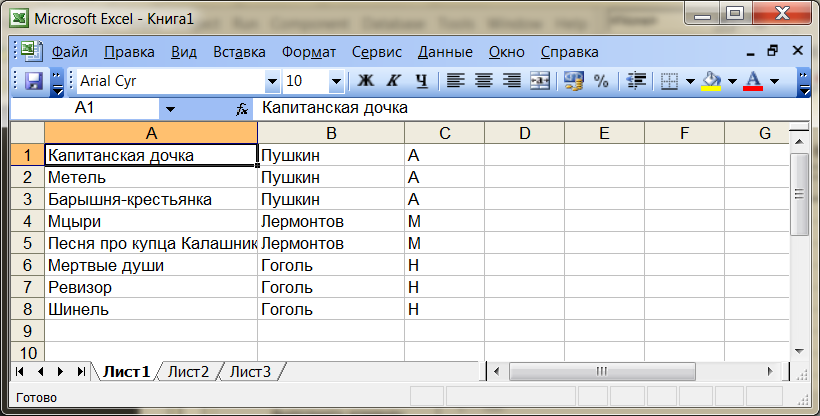


Рис. 104. Результат экспорта в Excel

Как можно видеть из рисунка, результаты запроса к БД успешно экспортировались в Excel. Для экспорта использован следующий код обработчика нажатия кнопки bExportExcel («Экспорт набора в Excel»): .

Листинг

|  |
| --- |
| // экспорт набора данных в Excel - демонстрация  // вызова метода DoRect  procedure TConnectForm.bExportExcelClick(Sender: TObject);  var  CountRows : integer;  datalist : TStringList;  begin  CountRows := 0;  datalist := TStringList.Create;  htq.First;  while not htq.Eof do begin  datalist.Add( htq.FieldByName( 'cName' ).AsString + chr(9) +  htq.FieldByName( 'cAuthorFam' ).AsString + chr(9) +  htq.FieldByName( 'cAuthorName' ).AsString );  CountRows := CountRows + 1;  htq.Next;  end;  htq.First;  ExcelComp.DoRect( 1, 1, CountRows, 3, datalist, false);  datalist.Free;  end; |

Для экспорта данных используется метод ExcelComp.DoRect. Этот метод предназначен для передачи в Excel строковых данных в табличном виде (прямоугольного фрагмента данных – результаты нашего запроса к БД могут быть представлены именно как такой фрагмент). Первые четыре параметра предназначены для задания прямоугольного фрагмента ячеек, куда должны быть экспортированы данные: номер строки (левый верхний угол), номер столбца (левый верхний угол), количество строк в прямоугольном фрагменте, количество столбцов в прямоугольном фрагменте. Пятый параметр задает список строк, отображаемых в фрагменте. Этот список мы формируем пройдя по записям результата запроса к БД. Следует обратить внимание, что для корректного разделения по колонкам внутри каждой строки значения полей нужно разделить символом табуляции (chr(9)). Шестой параметр в DoRect для выполнения экспорта должен быть установлен в false. Если этот параметр в DoRect установлен в true, то назначение вызова DoRect меняется строго на противоположное – вызов используется не для передачи данных в Excel, а для их получения. Впрочем, для получения данных компонент предоставляет другой вызов, использование DoRect напрямую для этих целей является избыточным.

Разумеется, передавать данные в Excel не всегда удобно в виде прямоугольного фрагмента, часто гораздо удобней поместить данные в конкретную ячейку. Для этих целей используются методы: PutStr, PutExt, PutDay, PutInt. Каждый метод принимает три параметра: индекс строки, индекс столбца, куда должны быть помещены данные, и значение соответствующего типа (строка, число с плавающей точкой, дата или целое число).

Экспорт данных – результатов запроса используется довольно часто, поэтому, разработчиками компонента реализованы высокоуровневые методы, которые позволяют выполнить такой экспорт путем единственного вызова метода, а не как в обработчике bExportExcel. Работу этих методов можно продемонстрировать в нашем примере при нажатии кнопок «Экспорт в Excel (DataSet)» и «Экспорт в Excel (Grid)». Обработчики нажатия этих кнопок приведены в .

Листинг

|  |
| --- |
| // экспорт набора данных в Excel  procedure TConnectForm.bExport2Click(Sender: TObject);  begin  ExcelComp.ExportData( htq, 'cName,cAuthorFam,cAuthorName', false, 1, 5 );  end;  // экспорт грида в Excel  procedure TConnectForm.bExport3Click(Sender: TObject);  begin  ExcelComp.ExportGrid( grd, true, 1, 5 );  end; |

Метод ExcelComp.ExportData используется для экспорта данных из открытого набора данных (класс TDataSet и все его наследники). В метод передается ссылка на набор данных, символьная строка со списком полей для экспорта через запятую (если список пустой, то предполагается, что будут экспортированы все поля), параметр, показывающий нужно ли экспортировать заголовки колонок (в нашем случае он равен false), номер записи, с которой выполняется экспорт и количество экспортируемых строк.

Метод ExcelComp.ExportGrid используется для экспорта данных из табличного представления (класс TDbGrid и все его наследники). В метод передается ссылка на табличное представление, а далее по аналогии с ExportData: признак экспорта заголовков, номер записи, с которой выполняется экспорт и количество экспортируемых строк.

При работе методов ExportData и ExportGrid может быть использована дополнительная возможность компонента, позволяющая отслеживать состояние экспорта. Компонент позволяет установить обработчик событий экспорта данных (событие ExcelComp.OnProgress). Исходный код обработчика приведен в .

Листинг

|  |
| --- |
| // обработка события экспорта данных  procedure TConnectForm.ExcelCompProgress(Sender: TObject;  Position: TProgressPosition; Value: Integer; var Stop: Boolean);  begin  mResults.Lines.Add( IntToStr(Value) );  end; |

Параметр Position равен одному из значений ppFirst, ppNext, ppLast в зависимости от того, начинаем ли мы экспорт, находимся в процессе экспорта или уже экспортировали последнюю строку. Параметр Value равен количеству выгружаемых записей (в момент старта выгрузки), или количеству выгруженных записей (в момент выполнения экспорта и по его завершению). Параметр Stop можно установить в значение true, если требуется прервать экспорт.

Еще одна интересная возможность использования компонента – пакетный импорт. Протокол доступа к API Excel достаточно медленный, поэтому, экспорт данных при большом количестве обращений к Excel (например, за один вызов экспортируем единственную ячейку) может оказаться неприемлемым для информационной системы по соображениям производительности. Как выход здесь предлагается сперва накопить экспортируемые данные, а затем выполнить их экспорт. В нашем примере приложения для этого нажмем кнопку «Пакетная операция». Программный код обработчика нажатия этой кнопки представлен в .

Листинг

|  |
| --- |
| // выполнение пакетной выгрузки данных  procedure TConnectForm.bBatchClick(Sender: TObject);  begin  ExcelComp.batchStart(5,5);  ExcelComp.PutStr(6,6,'12345');  ExcelComp.PutStr(6,7,'6789');  ExcelComp.batchSend;  end; |

Вызов ExcelComp.batchStart задает номер строки и номер столбца Excel (левый верхний угол) где начинается область пакетной выгрузки (экспорта) данных. Все операции по выгрузке данных, которые не попадают в эту область, будут выполняться немедленно, остальные операции – откладываться. Так, будут отложены операции ExcelComp.PutStr(6,6) и ExcelComp.PutStr(6,7), поскольку они попадают в область. Вызов ExcelComp.batchSend приведет к тому, что все ранее отложенные операции выгрузки будут выполнены единым пакетом, за одно обращение к Excel.

Помимо возможностей экспорта данных компонент работы с Excel дает возможности импорта. В нашем примере эти возможности демонстрируются при нажатии на кнопку «Получить область». Внешний вид приложения после нажатия на эту кнопку представлен на ().

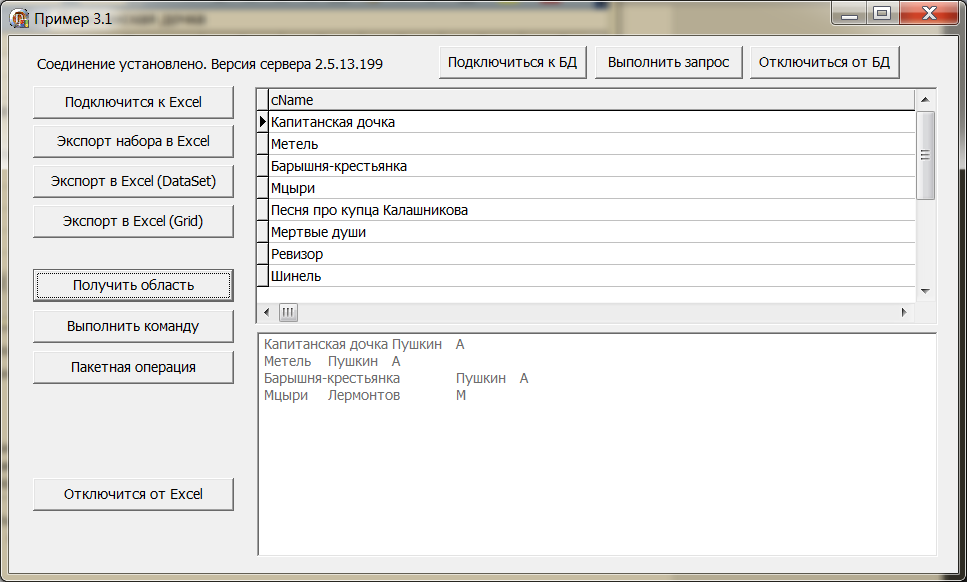


Рис. 105. Внешний вид приложения, после нажатия кнопки «Получить область»

Как видно из рисунка, мы импортировали данные из Excel и визуализировали их компонентом mResults. Исходный код обработчика нажатия кнопки «Получить область» приведен в .

Листинг

|  |
| --- |
| // получить ообласть ячеек Excel  procedure TConnectForm.bGetRegionClick(Sender: TObject);  var Lines : TStringList;  Rect : TRect;  i : integer;  begin  Lines := TStringList.Create;  Rect.Top := 1;  Rect.Left := 1;  Rect.Bottom := 4;  Rect.Right := 3;  ExcelComp.getRange( Rect, Lines );  mResults.Lines.Clear;  for i := 0 to Lines.Count - 1 do  mResults.Lines.Add( Lines[i] );  Lines.Free;  end; |

Получение данных из прямоугольной области ячеек Excel выполняется вызовом метода ExcelComp.getRange. Параметрами этого метода является структура с координатами прямоугольного фрагмента и ссылка на компонент, куда должны быть помещены результирующие строки. В каждой строке значения ячеек разделены символом табуляции (chr(9)).

Помимо использования getRange для получения значения отдельной ячейки можно воспользоваться методом GetCell. Индекс строки и столбца указывается в качестве входных параметров. Метод возвращает значение типа строка.

Не рассмотренным в нашем примере остался единственный вызов, а именно нажатие на кнопку «Выполнить команду». Код обработчика нажатия на данную кнопку приведен в .

Листинг

|  |
| --- |
| // выполнение команды Excel  procedure TConnectForm.bExecClick(Sender: TObject);  begin  ExcelComp.Exec('[SAVE.AS("export.xls")]');  end; |

Вызов метода ExcelComp.Exec позволяет выполнить произвольную команду, описанную в API Excel для внешних приложений. В примере это вызов команды сохранения текущей книги Excel в файле export.xls. После нажатия кнопки «Выполнить команду» можно убедится, что файл действительно создан и он имеет ожидаемое содержимое.

В заключение обзора компонента работы с Excel, помимо перечисленных методов компонента стоит упомянуть еще методы Select, SelRange. Метод Select позволяет выделить ячейку на листе Excel. Принимает два параметра – номер строки и номер столбца. Метод SelRange позволяет установить курсор на область листа Excel. Границы области задаются 4я входными параметрами – номер строки начала области, номер столбца начала области, номер строки окончания области, номер столбца окончания области.

## **Доступ к данным, организованным в виде дерева**

Визуализация данных в виде дерева (иерархии) в информационных системах в настоящее время достаточно распространена, особенно в задачах, где требуется работа с иерархическими справочниками или в интерфейсах пользователя предполагающих самостоятельную настройку функций приложений (многие современные системы масштаба предприятия предлагают для своей настройки пользовательский интерфейс на основе деревьев). На рынке представлен широкий спектр компонент для работы с деревьями в Delphi, каждый набор таких компонент акцентируется на тех или иных особенностях работы с деревьями, компоненты HyTech здесь не являются исключением. В основу визуальных компонент HyTech по работе с деревьями положены следующие архитектурные принципы:

* Расширение функциональности стандартных компонент, что упрощает обучение программистов работе с деревьями (помимо документации имеется большое количество литературы и примеров по использованию стандартных компонент). В стандартной поставке Дельфи имеется компонент TTreeView, в HyTech он выбран в качестве базового для создания компонентов визуализации деревьев.
* Гибкость представления и хранения данных в деревьях. Компоненты HyTech предлагают базовые классы для создания вершин дерева различных типов как расширений базовых классов, что позволяет использовать уже существующую функциональность без изобретения решений по представлению и хранению данных «с нуля».
* Визуализация дерева и функциональная логика разделены. Одни компоненты используются для построения структуры деревьев и хранения данных в узлах, другие для визуализации. Базовые классы работы с деревьями позволяют синхронное использование и того, и другого. Как следствие, компоненты позволяют отображать изменения одних и тех же данных в разных деревьях, которые может видеть пользователь на экране.

Заметим, что с точки зрения описания приемов использования большинство компонентов работы с деревьями достаточно сложны, это следствие той «мощности» функций для конечных пользователей, которые они предоставляют. Обычно компоненты работы с деревьями содержат десятки свойств и методов которыми можно воспользоваться. Нашей целью в данном разделе не является их описание, сосредоточимся на демонстрации того, как реализуются в компонентах HyTech указанные выше архитектурные принципы, а также на представлении данных из БД. Как и в других разделах курса воспользуемся демонстрационным примером.

Усовершенствуем пример из пункта 2.2. добавив на главную форму приложения новые компоненты. Из ранее созданных примеров нам понадобятся только функции соединения и отключения от БД.

В примере мы покажем реализацию следующих функций:

* Создание структуры дерева из БД. Дерево будет содержать два уровня иерархии. В первом уровне будут авторы книг, а во втором книги. На втором уровне книги будут находиться под их авторами.
* Визуализация структуры дерева. Мы визуализируем одно и тоже дерево два раза для демонстрации разделения структуры дерева и его внешнего представления.
* Синхронизация визуальных представлений. После того, как в структуру дерева внесены изменения они будут отображены во всех его визуальных представлениях.

Перечень всех компонент нашего демонстрационного приложения и их назначение приведены в .

Таб. . Компоненты главной формы примера работы с данными, организованными в виде дерева

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Название | Тип | Закладка | Статус | Назначение |
|  | Htb | THtBase | HyTech | Без изменений | Осуществление связи с БД |
|  | bConnect | TButton | Standard | Без изменений | Установка связи с БД |
|  | bDisconnect | TButton | Standard | Без изменений | Отключение от БД |
|  | HTreeView1 | THTreeView | HtComp | Добавлен | Визуализация структуры дерева |
|  | HTreeView2 | THTreeView | HtComp | Добавлен | Визуализация структуры дерева, визуализируется та же структура, что и в компоненте HTreeView1. Цель – показать синхронные изменения изображения в HTreeView1 и HTreeView2 при изменениях в данных |
|  | ImageList1 | TImageList | Win32 | Добавлен | Список иконок для пометки вершин дерева |
|  | bCreateTree | TButton | Standard | Добавлен | Создание структуры дерева. Сама структура не визуализируется |
|  | bShowTree1 | TButton | Standard | Добавлен | Визуализация структуры дерева, созданного по нажатию bCreateTree |
|  | bShowTree2 | TButton | Standard | Добавлен | Визуализация структуры дерева, созданного по нажатию bCreateTree. Убеждаемся, что одна и та же структура визуализируется разными компонентами |
|  | bChangeTree | TButton | Standard | Добавлен | Изменение элемента данных в дереве. Синхронные изменения изображения в HTreeView1 и HTreeView2 при изменении элемента данных |

Внешний вид формы в среде проектирования приведен на .

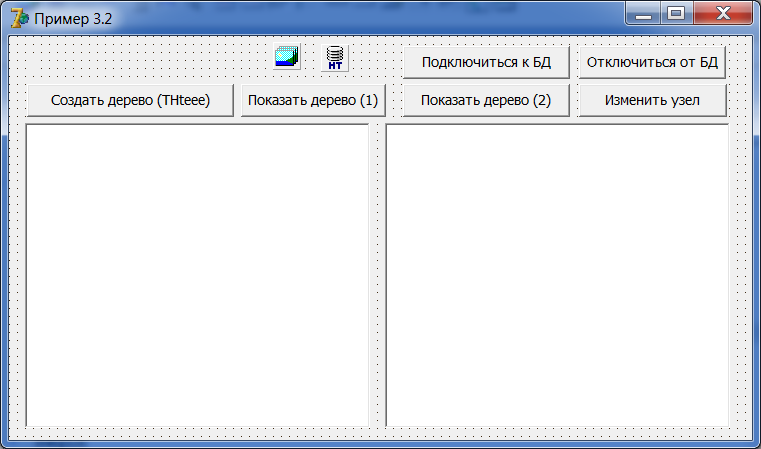


Рис. 106. Внешний вид формы (среда разработки)

Скомпилируем и запустим приложение. Установим связь с базой данных, нажмем на кнопку «Создать дерево (THteee) ().

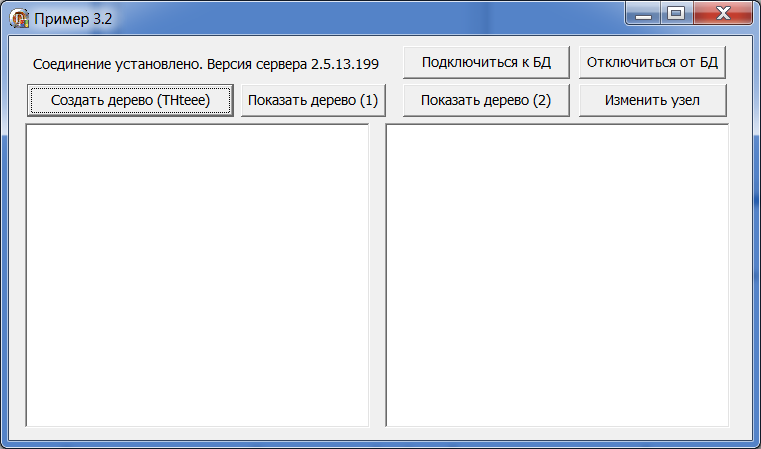


Рис. 107. Внешний вид приложения, дерево с данными БД создано

Программный код по созданию дерева приведен .

Листинг

|  |
| --- |
| // создание структуры дерева  procedure TConnectForm.bCreateTreeClick(Sender: TObject);  begin  if Assigned(Htree) then  Htree.Free;  // создаем дерево  Htree := THtree.Create( Self );  // добавляем корневую вершину  AddNode( HTree, nil, -1, 'Библиотека', HRootNode, 0 );  // создаем дерево из БД  FillTree(HTree, 0, HRootNode,  ['author', 'book'],  ['cFamName', 'cName'],  ['nAuthorKey','nBookKey'],  ['','nAuthorKey'] );  end; |

Структура дерева создается с использованием класса THtree. THtree не входит в стандартные компоненты Дельфи, это разработка команды Hytech. Этот класс используется для хранения структуры дерева. В нашем приложении мы определили переменную Htree типа THtree. Поскольку дерево у нас одно, переменная Htree определена в классе главной формы приложения и далее в коде упоминается неоднократно. Метод AddNode используется для добавления новой вершины, его код приведен в .

Листинг

|  |
| --- |
| // добавить узел в структуру дерева  procedure TConnectForm.AddNode(  HTree: THTree; // дерево  Parent : TExNode; // родительская вершина  ParentKey : integer; // ключ родительской вершины (из БД0  cNodeName: string; // имя вершины  var HNode : TExNode; // ссылка на новый созданный узел дерева  Level : integer ); // уровень вложенности (начинается с 0)  begin  HNode := TExNode.Create(Htree);  // проверяем, есть ли родительский узкл  if Assigned(Parent) then begin  // добавляем родительский узел  Parent.AddChild(-1, HNode);  HNode.SetParent(Parent);  end;  // устанавливаем атрибуты  HNode.SetImage( Level );  HNode.SetSelectedImage( Level + 2 );  HNode.SetOvlImage(0);  HNode.Visible := true;  HNode.SelectedImage := 0;  HNode.Text := cNodeName;  HNode.ParentKey := ParentKey;  end; |

Ключевым архитектурным решением в HyTech, предлагаемым разработчиками технологии работы с деревьями, является использование разных классов узлов в дереве. В примере используется узел класса TExNode. Определение TExNode и реализация методов приведены в .

Листинг

|  |
| --- |
| TExNode = class(THNode)  private  FParentKey : integer;  protected  FText : string; // для отображения в TreeView  FImage : integer;  FSelImage : integer;  public  procedure SetParentKey(const Value: integer);  function GetParentKey: integer;  procedure SetText(const Value: string); override;  function GetText: string; override;  procedure AddChild(ndx: integer; var Node :TExNode);  procedure SetImage(const Value: integer); override;  procedure SetSelectedImage(const Value: integer); override;  function GetImage: integer; override;  function GetSelectedImage: integer; override;  published  property Text: string read GetText write SetText;  property ParentKey: integer read GetParentKey write SetParentKey;  end;  procedure TExNode.SetImage(const Value: integer);  begin  FImage := Value;  end;  procedure TExNode.SetSelectedImage(const Value: integer);  begin  FSelImage := Value;  end;  function TExNode.GetImage: integer;  begin  result:= FImage;  end;  function TExNode.GetSelectedImage: integer;  begin  result:= FSelImage;  end;  procedure TExNode.AddChild(ndx: integer; var Node :TExNode);  begin  if ndx < 0 then  FItems.Add(Node)  else  FItems.Insert(ndx, Node);  end;  procedure TExNode.SetText(const Value: string);  begin  FText := Value;  end;  function TExNode.GetText: string;  begin  result:= FText;  end;  procedure TExNode.SetParentKey(const Value: integer);  begin  FParentKey := Value;  end;  function TExNode.GetParentKey: integer;  begin  result:= FParentKey;  end; |

Класс TExNode наследник класса THNode, что является обязательным условием его использования в рассматриваемой технологии работы с деревьями. В дереве могут использоваться разные классы узлов, но только являющиеся наследником THNode. TExNode в нашем примере «заточен» под конкретную задачу – например, в нем определено значение ключа родительской записи для чтения данных из БД. Конкретные методы TExNode далее разъясняются при описании метода FillTree. Перейдем к его рассмотрению.

Метод FillTree в нашем приложении используется дл создания иерархической структуры в памяти клиентского приложения на основе структуры БД. Иерархия организуется по связям между таблицами БД по внешним ключам. В примере такими связанными таблицами являются author и book (авторы и книги). Каждая книга имеет автора. В дереве первый уровень это авторы, а второй – их книги.

Исходный код метода FillTree приведен приведен в .

Листинг

|  |
| --- |
| // заполнить структуру дерева данными из БД  procedure TConnectForm.FillTree(  HTree : THTree; // дерево для заполнения  Level : integer; // уровень дерева, на котором вызван метод  // (0 - начальный уровень)  ParentNode : TExNode; // родительский узел  Table: array of string; // список названий таблиц,  // на первом уровне дерева  // читаются данные первой таблицы, на втором второй и т.п.  NodeNameList: array of string; // названия полей для именования  // вершин, для каждой таблицы из Table свое имя поля  KeyList: array of string;  // названия полей - первичных ключей  // для каждой таблицы из Table свой ключевое поле  ParentList: array of string);  // названия полей - внешних ключей для фильтрации данных  // например для BOOK данные фильтруются по nAuthorKey,  // значение nAuthorKey для фильтрации устанавливается из  // текущей строки AUTHOR (nAuthorKey в этой таблице первичный ключ)  // для каждой таблицы из Table свой внешний ключ  var  q1 : THtQuery;  SQL : string;  Node : TExNode;  ParentKey : integer;  begin  // формируем текст запроса к БД  SQL := 'select ' + NodeNameList[ Level ] + ', ' +  KeyList[ Level ] +  ' from ' + Table[ Level ];  if Level = 0 then begin  SQL := SQL + ';';  end else begin  // для уровней дерева ниже чем корень запрос  // будет содержать условие фильтрации по  // первичному ключу родителя  ParentKey := ParentNode.ParentKey;  SQL := SQL + ' where ' + ParentList[ Level ] +  '=' + IntToStr(ParentKey) + ';';  end;  q1 := THtQuery.Create(Self);  q1.BaseName := 'htb';  q1.SQL.Text := SQL;  q1.Active := true;  // формируем подчиненные вершины дерева ParentNode  while not q1.eof do begin  AddNode( HTree,  ParentNode,  q1.FieldByName(KeyList[Level]).AsInteger,  // ключ вершины - родителя  q1.FieldByName(NodeNameList[Level]).AsString,  // название вершин - потомка  Node, Level );  // Node - новая вершина, для нее формируем своих потомков  if Level + 1 < Length( Table ) then begin  HTree.MakeSubTree( Node );  FillTree(HTree, Level + 1,  Node, Table, NodeNameList, KeyList, ParentList);  end;  q1.Next;  end;  q1.Free;  end; |

Метод FillTree в примере вызывается рекурсивно для формирования подчиненных узлов дерева. Визуализировать результаты работы метода FillTree можно нажав в приложении кнопку «Показать дерево (1)». Аналогичный результат можно получить нажав кнопку «Показать дерево (2)». Результат после открытия узлов дерева можно увидеть на ().

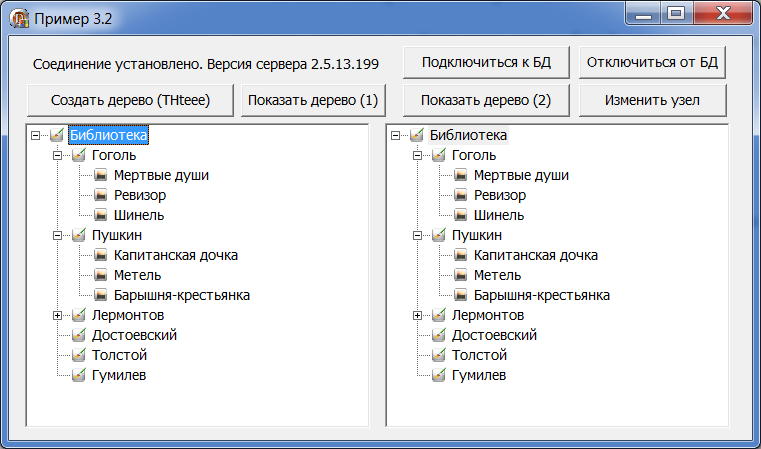


Рис. 108. Внешний вид приложения, дерево с данными БД визуализировано двумя компонентами

Программный код по визуализации дерева приведен .

Листинг

|  |
| --- |
| // визуализировать дерево (1 вариант)  procedure TConnectForm.bShowTree1Click(Sender: TObject);  begin  FillVisualTree(HTreeView1, HTree, HRootNode );  end;  // визуализировать дерево (2 вариант)  procedure TConnectForm.bShowTree2Click(Sender: TObject);  begin  FillVisualTree(HTreeView2, HTree, HRootNode );  end;  // визуализировать дерево  procedure TConnectForm.FillVisualTree(  Tree: THTreeView; // компонент для визуализации  HTree : THTree; // дерево с данными  HNodeRoot : TExNode // корневая вершина с данным  // для визуализации  );  var  Node, NodeParent : TmNode;  HNode, HNodeChild : TExNode;  i : integer;  begin  // создаем вершину - корень визуального дерева  NodeParent := TmNode.Create(Tree.Items);  // добавляем корень визуального дерева  Tree.Items.AddNode(NodeParent, nil, HNodeRoot.Text, nil, naAdd);  // Важно! Связываем узел визуального дерева и  // дерева с данными  NodeParent.HNode := HNodeRoot;  NodeParent.ImageIndex := 0;  // идем по потомкам первой вершины дерева с данными  // Важно! Рекурсия не нужна, все подчиненные узлы созздадутся сами  for i := 0 to HNodeRoot.FItems.Count - 1 do begin  Node := TmNode.Create(Tree.Items);  HNodeChild := TExNode( HNodeRoot.FItems[ i ].Body );  // добавляем узел в визуальное дерево  Tree.Items.AddNode(Node, NodeParent, HNodeChild.Text, nil, naAddChild);  // Важно! Связываем узел визуального дерева и  // дерева с данными  Node.HNode := HNodeChild;  end;  // присваиваем свойства визуализации из дерева данных  for i := 0 to Tree.Items.Count - 1 do begin  Tree.Items[ i ].ImageIndex :=  TexNode( TmNode(Tree.Items[ i ]).HNode.Body ).GetImage();  Tree.Items[ i ].SelectedIndex :=  TexNode( TmNode(Tree.Items[ i ]).HNode.Body ).GetSelectedImage();  end;  end; |

И, наконец, покажем, как можно синхронизировать информацию в двух деревьях. Для этого в нашем приложении нажмем на кнопку «Изменить узел» Результат показан на ().

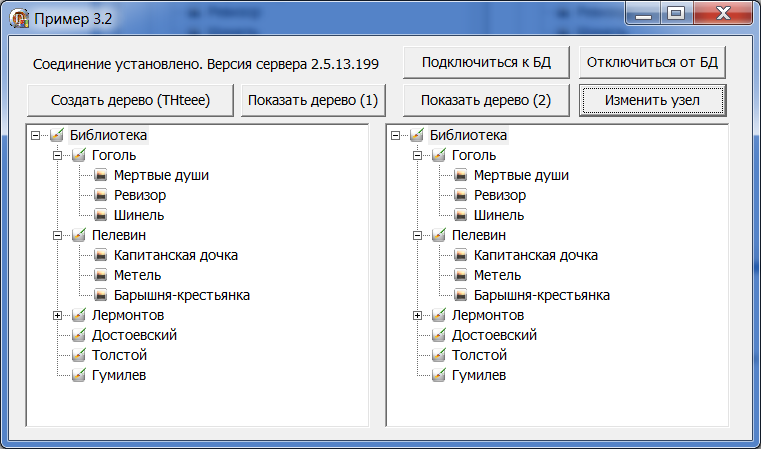


Рис. 109. Внешний вид приложения, Пушкин стал Пелевиным

Как можно видеть, фамилия «Пушкин» изменена в дереве на фамилию «Пелевин». Достигается это при помощи следующего кода ().

Листинг

|  |
| --- |
| // изменить данные в узле, синхронно показать  // пользователю в разных компонентах  procedure TConnectForm.bChangeTreeClick(Sender: TObject);  var  Node : TexNode;  begin  Node := TExNode( HRootNode.FItems[1].Body );  Node.Text := 'Пелевин';  HTreeView1.RefreshView;  HTreeView2.RefreshView;  end; |

Разумеется, описанные в данном разделе возможности компонент работы с деревьями, предлагаемые разработчиками HYTech не исчерпываются, для их изучения следует обратиться к документации.

# Комплексный пример приложения, использующего компоненты

# Настройка СУБД для комплексного примера

Рассмотрим комлексный пример для обобщения информации о компонентах Delphi, предназначенных для работы с СУБД HyTech, закрепления навыков работы с БД. В качестве примерва будет рассмотрено клиент-серверное приложение, предназначенное для учета книг, выдаваемых студентам разных кафедр. В ходе описания комплексного примера будут рассмотрены типовые сценарии работы клиент-серверного приложения с БД, такие как:

* выбор информации из таблиц БД;
* редактирования информации в таблицах БД;
* использование прагм;
* работа со справочниками;
* экспорт информации в Excel;
* фильтрация информации посредством специальных компонентов HyTech;
* импорт данных в DBF;
* экспорт данных из DBF.

Для ознакомления с примером необходимо настроить ещё один экземпляр СУБД HyTech, для чего создадим директорию c:\demo, распакуем в неё содержимое архива demo.rar, после распаковки архива в директории будут находиться две поддиректории – bin и db. Для настройки экземпляра в каталоге c:\Demo\bin необходимо выполнить файл 0ini.cmd с параметром "номер IP-порта", при этом в каталоге c:\demo\bin создается файл sql64.ini с параметрами, позволяющими СУБД работать по указанному порту. Затем необходимо выполнить файл 0.cmd, для настройки каталога с c:\demo\db как каталога для хранения таблиц БД.

Выполним файл 0ini.cmd, передав в качестве номера порта значение 1050. Выполним файл 0.cmd. Выполним файл c:\demo\bin\sql64.exe для запуска СУБД.

СУБД для демонстрации комплексного примера сконфигурирована и запущена.

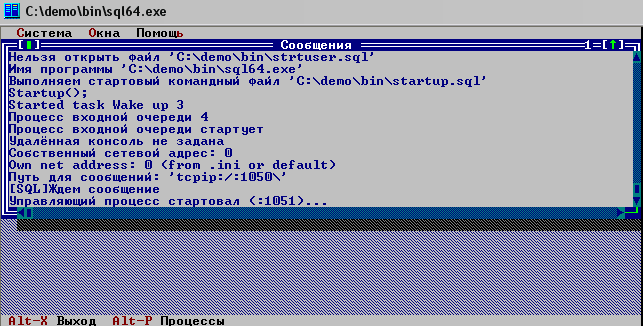


Рис. 110. СУБД для демонстрации комплексного примера сконфигурирована и запущена

В случае если настроить ещё один экземпляр СУБД не представляется возможным, скрипты создания таблиц и наполнения их данными приведены в архиве demoDB.rar. Применить их можно на уже имеющемся экзмепляре СУБД.

База данных комплексного примера предназначена для учета книг, выдаваемых студентам.

Логическая и физическая схемы БД приведены на , .

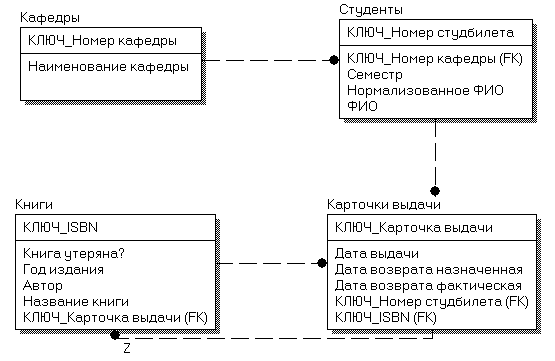


Рис. 111. Логическая схема БД

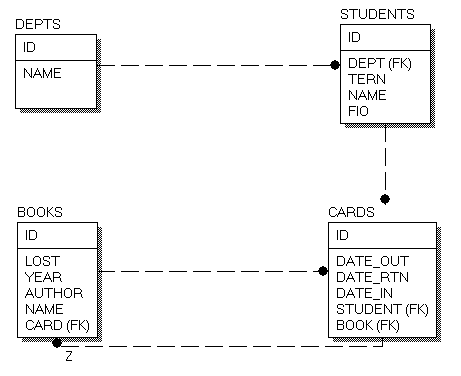


Рис. 112. Физическая схема БД

# Конфигурирование среды разработки для запуска тестового приложения

Исходный код, а также скопмилированный файл для запуска приложения содержится в архиве biblos.rar. Распаковать архив можно в любой каталог локального диска, распакуем архив в директорию c:\biblos. Для работы с исходным кодом может быть использована та же среда разработки, которая использовалась в предыдущих разделах. Запустим среду разработки и откроем в ней файл проекта, он располагается по адресу c:\biblos\Biblos.dpr. Внешний вид проекта (его главная формы) в среде разарботки представлен на .

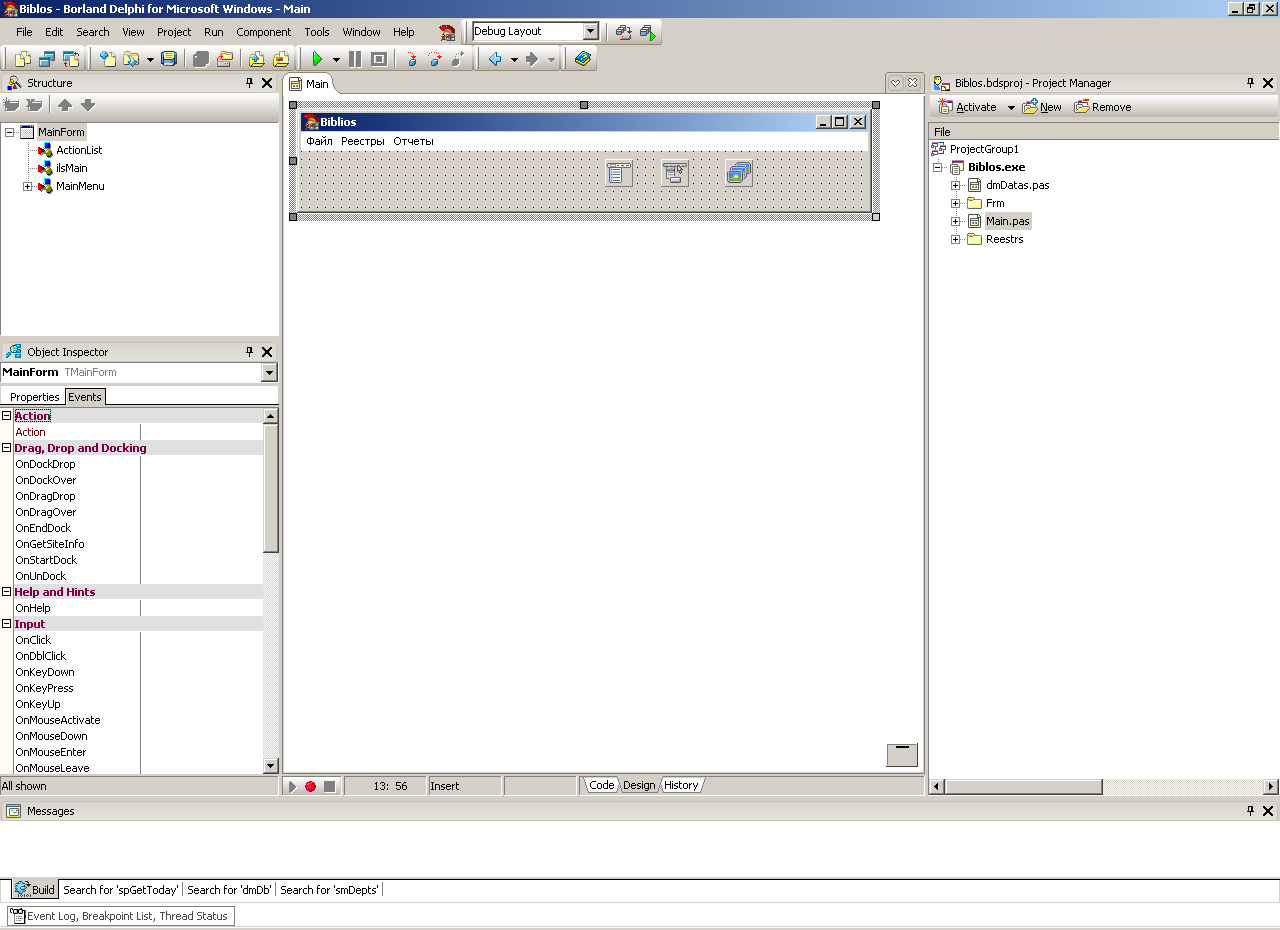
****

Рис. 113. Главная форма проекта в среде разработки

Скомпилируем и запустим приложение, на экране появится окно ввода реквизитов подключения к базе данных. Установим соединение с БД, располагающейся по адресу tcpip:/localhost:1050 под учетной записью HTADMIN с паролем PASSWORD.

Процесс установки связи с БД и внешний вид приложения после успешного соединения представлен на – .

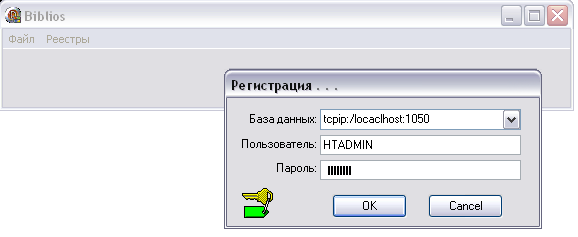


Рис. 114. Установка связи с тестовой БД

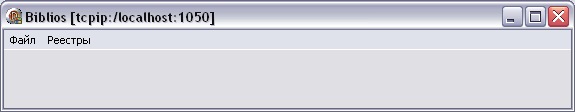


Рис. 115. Внешний вид приложения после успешной установки связи с БД

# Описание приложения

После успешного соединения с базой данных в приложении становятся доступными пункты меню «Файл»(), «Реестры» () и «Отчеты»().



Рис. 116. Внешний вид приложения, меню «Файл»



Рис. 117. Внешний вид приложения, меню «Реестры»



Рис. 118. Внешний вид приложения, меню «Отчеты»

Для реализации данных пунктов меню использован компонент TMainMenu, расположенный на главной форме. Добавление пунктов меню в процессе разработки приложения приведено на .

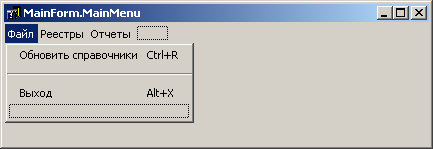


Рис. 119. Добавление пунктов меню в процессе разработки приложения

По нажатию на различные пункты меню основной формы приложения (TMainForm в модуле Main.pas) осуществляется: открытие некоторых форм, обновление справочники, открытие формы для генерации отчета и закрытие приложение, более подробно мы рассмотрим открытие форм и обновление справочников в следующих ниже.

Открытие формы осуществлеятся посредством метода FindShow (см. ).

Листинг

|  |
| --- |
| procedure TMainForm.miBooksClick(Sender: TObject);  begin  TfrmBooks.FindShow;  end; |

Формы «Книги», «Студенты», «Кафедры» реализованы посредством классов TfrmBooks (модуль Books), TfrmStudents( модуль Students) и TfrmDepts (модуль Depts). Все три класса имеют общий класс-предок TfrmReestr (модуль TfrmReestr), в котором реализован метод FindShow (см ). Сам метод является класс – функцией, то есть его вызов можно осуществлять, не создавая заранее экземпляр класса.

Листинг

|  |
| --- |
| class function TfrmReestr.FindShow: TForm;  var i: integer;  P: TClass;  begin  asm // метод ClassType не может быть вызыван напрямую в класс-функции  mov P, eax  end;  result:= nil;  // цикл по уже открытым на экране формам  for i:= 0 to Screen.FormCount -1 do  begin  if Screen.Forms[i].ClassType <> P then  Continue;  // если требуемая форма найдена в открытых, вернуть указатель на неё  result:= Screen.Forms[i];  Break;  end;  // в случае если требуемая форма среди открытых не обнаружена,  // необходимо её создать и отобразить на экране  if not assigned(result) then  result:= Create(Application);  result.Show;  end; |

Рассмотрение класса TfrmReestr продолжим в разделе .

Остановимся более подробно на каждом пункте меню, рассмотрим возможности, которые предоставляются пользователю и код, который для этого необходим.

# Обновление справочников

Справочниками в данном случае являются списки студентов и кафедр, используемые в качестве LookUp – полей при редактировании полей форм (см , ). Списки реализованы посредством компонентов TSqlMem: smStudents для списка студентов и smDepts для списка кафедр. Напомним, что при активизации компонент TSqlMem выполняет запрос к БД и сохраняет результат запроса (набор данных) в свой буфер. Списки располагаются в модуле dmDatas, ссылка на который находится в разделе “uses” главного модуля. В модуле dmDatas реализован класс – наследник TDataModule – TdmDb. Атрибутами класса в данном случае являются:

* HtBase, класс THtBase;
* smDepts, класс TSqlMem (с полями smDeptsID, smDeptsNAME)
* smStudents, класс TSqlMem (с полями smStudentsID, smStudentsNAME);
* spGetToday: THtStoredProc.

Внешний вид модуля представлен на .

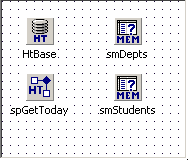


Рис. 120. Внешний вид модуля dmDatas (среда разработки)

HtBase используется компонентами во всех модулях приложения, которым необходима связь с базой данных. Компонент spGetToday предназначен для получения текущей даты. Кроме того, в модуле реализована первоначальная загрузка справочников при успешном соединении с БД, обработка ошибок, возникающих при работе с базой данных.

Вернемся к пункту меню главной формы «Обновить справочники». Код, выполняемый при нажатии на него, приведен в .

Листинг

|  |
| --- |
| procedure TMainForm.miRefreshDictClick(Sender: TObject);  begin  // dmDb - переменная класса TdmDb  with dmDb do  begin  smDepts.Close;  smDepts.Open;  smStudents.Close;  smStudents.Open;  ShowMessage('Обновление локальных справочников завершено.');  end;  end; |

# Реестры

Реестры предназначены для отображения форм просмотра и редактирования информации в таблицах базы данных. Все реестры реализованы по одной и той же схеме и имеют общего предка, которым является класс класс TfrmReestr. Описание класса приведено в модуле fReestr, внешний вид формы в среде разработки представлен на .

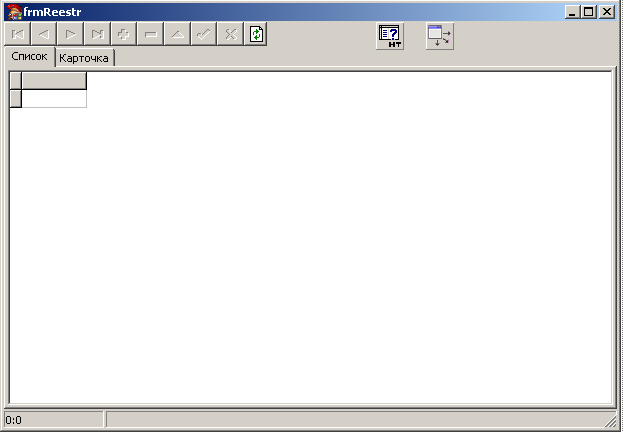


Рис. 121. Внешний вид формы (среда разработки)

Список основных компонентов, располагающихся на форме, представлен в .

Таб. . Компоненты формы frmReestr

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название | Тип | Назначение |
|  | qryBase | THtQuery | Формирует основной набор данных |
|  | srcMain | TDataSource | Устанавливает связь между компонентами qryBase и grdMain |
|  | grdMain | TDBGrid | Отображает набор данных qryBase |
|  | DBNavigator | TDBNavigator | Осуществляет навигацию по набору данных |
|  | btnRefresh | TSpeedButton | Предназначен для обновления набора данных |
|  | pgsMain | TPageControl | Содержит страницы для отображения и редактирования |
|  | tsGrid | TTabSheet | Является страницей, на которой расположены компоненты для отображения данных |
|  | tsCard | TTabSheet | Является страницей, на которой расположены компоненты для редактирования данных |
|  | pnlTop | TPanel | Предназначена для группировки функциональных кнопок |

Полю BaseName компонента qryBase присвоим значение HtBase (HtBase объявлен в модуле dmDatas). Полю DataSet компонента srcMain присвоим значение qryBase. Полю DataSource компонента grdMain присвоим значение srcMain. В качестве активной страницы контейнера pgsMain укажем tsGrid (pgsMain.ActivePage = tsGrid), на которой расположен компонент grdMain. На странице tsCard будут расположены компоненты в формах-наследниках.

Полю Datasource компонента DBNavigator присвоим значение srcMain.

На панель pnlTop поместим кнопку btnRefresh. Код, выполняющийся при нажатии на кнопку btnRefresh, приведен в .

Листинг

|  |
| --- |
| procedure TfrmReestr.btnRefreshClick(Sender: TObject);  begin  qryBase.Close;  qryBase.Open;  end; |

# Реестр «Кафедры»

При нажатии на пункт меню главной формы Реестр ->Кафедры открывается (или становится активной, если была открыта ранее) форма «Книги». Внешний вид формы представлен на .

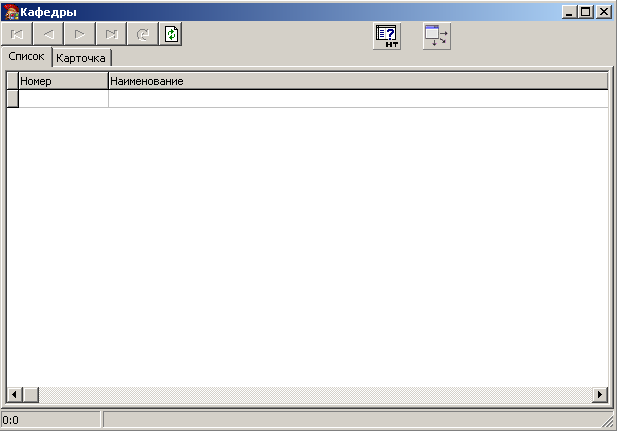


Рис. 122. Внешний вид формы в среде проектирования

Форма реализована в модуле Depts.pas посредством класса TfrmDepts, который является наследником TfrmReestr. Изменим свойство SQL компонента qryBase - поместим в него текст, приведенный в .

Листинг

|  |
| --- |
| fix all;  select \* from DEPTS  order by ID; |

Также в компоненте создадим два поля

* qryBaseID типа TSmallintField для ID кафедры, свойству FieldName присвоим значение «ID», свойству DisplayLabel присвоим значение «Номер»;
* qryBaseNAME типа TStringField для названия кафедры, свойству FieldName присвоим значение NAME, свойству DisplayLabel присвоим значение «Наименование».

Отредактируем свойство VisibleButtons компонента DBNavigator так, чтобы связанный с компонентом набор данных невозможно было бы отредактировать, то есть присвоим значение False свойствам nbInsert, nbDelete, nbEidt, nbPost, nbCancel, nbRefresh.

# Реестр «Книги»

При нажатии на пункт меню главной формы Реестр ->Книги открывается (или становится активной, если была открыта ранее) форма «Книги». Внешний вид формы представлен .

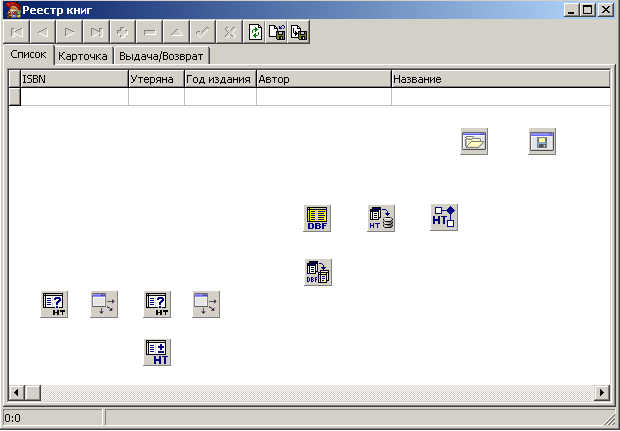


Рис. 123. Внешний вид формы (среда разработки)

Объявление формы располагается в модуле Books. Список основных компонентов, отличных от компонентов формы-предка, располагающихся на форме, представлен в .

Таб. . Компоненты формы frmBooks

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название | Тип | Назначение |
|  | dbeISBN | TDBEdit | Предназначено для добавления/ редактирования атрибута ID таблицы BOOKS |
|  | dbeYear | TDBEdit | Предназначено для добавления/ редактирования поля YEAR таблицы BOOKS |
|  | dbeAuthor | TDBEdit | Предназначено для добавления/ редактирования атрибута AUTHOR таблицы BOOKS |
|  | dbeName | TDBMemo | Предназначено для добавления/ редактирования атрибута NAME таблицы BOOKS |
|  | dbchkLost | TDBCheckBox | Предназначено для добавления/ редактирования атрибута LOST таблицы BOOKS |
|  | tsInOut: | TTabSheet | Является страницей для добавления/ редактирования информации о выдаче/возврате книг студентами |
|  | qryInOut | THtQuery | Обеспечивает работу с набором данных при добавлении/редактировании информации о выдаче/возврате книг студентами |
|  | srcInOut | TDataSource | Предназначен для связи компонентов редактирования данных (dbeOut, dbeRet, dbeIn, lcmbStudent) с набором данных |
|  | dbeOut | TDBEdit | Предназначен для ввода даты выдачи книги |
|  | dbeRet | TDBEdit | Предназначен для ввода даты, не позднее которой необходимо вернуть книгу |
|  | dbeIn | TDBEdit | Предназначен для ввода фактической даты возврата |
|  | lcmbStudent | TDBLookupComboBox | Предназначен для выбора студента, которому выдается книга |
|  | btnToday | TSpeedButton | Проставляет значение текущей даты в поле dbeIn, если оно доступно для редактирования |
|  | DBNavigator1 | TDBNavigator | Предоставляет возможность сохранить информацию в БД или отменить изменения при работе с информацией о возврате/выдаче книг |
|  | updInOut | THtUpdateSQL | Осуществляет согласованное изменение двух таблиц – CARDS и BOOKS при работе с информацией о возврате/выдаче книг. |
|  | sbImport | TSpeedButton | Инициирует импорт информации из файла формата dbf |
|  | dbfImport | THtDbfTable | Является источником информации для компонента dspImport, получает данные непосредственно из dbf файла. |
|  | dspImport | THtDSPump | Осуществляет импорт информации из файла компонента dbfImport во временную таблицу |
|  | spImport | THtStoredProc | Обновляет информацию в таблице BOOKS информацией из временной таблицы |
|  | dlgOpenDBF | TOpenDialog | Предоставляет возможность выбора файла импорта |
|  | sbExport | TSpeedButton | Инициирует экспорт информации в файл формата dbf |
|  | dbfExport | TDBFMove | Осуществляет экспорт информации в файл формата dbf |
|  | dlgSaveDBF | TSaveDialog | Предоставляет возможность выбора файла экспорта |

В свойство SQL компонента qryBase поместим текст SQL запроса, приведенный в .

Листинг

|  |
| --- |
| fix all;  select \* from BOOKS  order by ID; |

В компоненте qryBase создадим ряд полей:

* qryBaseID для ISBN книги, тип поля - TStringField, свойству FieldName присвоим значение ID, свойству DisplayLabel присвоим значение «ISBN», EditMask значение «999-9-99-999999-9»;
* qryBaseNAME для названия книги, тип поля - TStringField, свойству FieldName присвоим значение NAME, свойству DisplayLabel присвоим значение «Название»;
* qryBaseAUTHOR для автора книги, тип поля - TStringField, свойству FieldName присвоим значение AUTHOR, свойству DisplayLabel присвоим значение «Автор»;
* qryBaseYEAR для года издания, тип поля - TSmallintField, свойству FieldName присвоим значение YEAR, свойству DisplayLabel присвоим значение «Год»;
* qryBaseLOST для признака «Утеряна», тип поля - TIntegerField, свойству FieldName присвоим значение LOST, свойству DisplayLabel присвоим значение «Утеряна»;
* qryBaseCARD для карточки, тип поля - TIntegerField, свойству FieldName присвоим значение CARD, свойству DisplayLabel присвоим значение «CARD».

Для полей ID, LOST, YEAR, AUTHOR, NAME создадим столбцы (TColumn) в компоненте grdMain (при этом необходимо указать название поля в каждом создаваемом компоненте в свойстве FieldName). Назначим событию grdMain.OnDrawColumnCell обработчик. Код обработчика приведен в .

Листинг

|  |
| --- |
| procedure TfrmBooks.grdMainDrawColumnCell(Sender: TObject; const Rect: TRect;  DataCol: Integer; Column: TColumn; State: TGridDrawState);  Begin  // проверим, выдана ли книга по атрибуту CARD текущей записи.  if qryBaseCARD.AsInteger <> 0 then  // если в атрибуте присутствует информация о карточке выдачи,  // будем отображать строки таблицы жирным шрифтом  grdMain.Canvas.Font.Style:= [fsBold];  grdMain.DefaultDrawColumnCell(Rect, DataCol, Column, State);  end; |

Для навигации по строкам таблице будем использовать компонент DbNavigator, объявленный в предке класса. Также этот компонент предоставляет стандартные возможности добавления, редактирования, удаления информации из набора данных.

Альтернативную возможность редактирования набора данных реализуем на странице tsCard, где разметим компоненты, предназначенные для редактирования и добавления информации о книгах в таблицу BOOKS - dbeISBN, dbeYear , dbchkLost, dbeAuthor, dbeName. Свойству DataSource каждого из этих компонентов присвоим значение srcMain, свойству DataDield значения ID, YEAR, LOST, AUTHOR, NAME соответственно.

Таким образом, обе страницы (и tsGrid, и tsCard) используют один и тот же набор данных (qryBase). Назначим обработчик событию qryBase.BeforePost. Код обработчика приведен в .

Листинг

|  |
| --- |
| procedure TfrmBooks.qryBaseBeforePost(DataSet: TDataSet);  var YearNow: Word;  begin  YearNow := YearOf(dmDb.DateNow);  if qryBaseYEAR.AsInteger > YearNow then  begin  ShowMessage('Издание книги не может быть позже текущего (' + IntToStr(YearNow) + ') года');  Abort;  end  else  if qryBaseYEAR.AsInteger < 0 then  begin  ShowMessage('Издание книги не может быть раньше Рождества Христова');  Abort;  end;  end; |

На странице tsInOut разметим компоненты, предназначенные для редактирования и добавления информации о выданных/возвращенных книгах в таблицу CARDS – lcmbStudent, dbeOut, dbeRet, dbeIn. Кроме того, для работы с набором данных из таблицы CARD будут необходимы компоненты DBNavigator1, qryInOut, srcInOut, updInOut.

Свойству DataSource компонента qryInOut присвоим значение srcMain для связи с основным набором данных. В свойство SQL компонента qryInOut поместим текст SQL запроса, приведенный в .

Листинг

|  |
| --- |
| fix all;  select \* from CARDS  where ID = :CARD  for update; |

Также в компоненте qryInOut создадим параметр CARD. (см ).

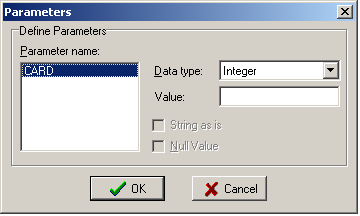


Рис. 124. Параметр CARD

При добавлении/редатировании информации в таблице CARDS необходимо такде обновлять таблицу BOOKS – проставлять, удалять значение атрибуту CARD в таблице BOOKS., если книгу выдают или возращают, таким образом, работа с данными должна осуществляться под транзакцией, обновляться информация должна сразу для двух таблиц. Для реализации требуемого алгоритма работы с данными будем использовать компонент updInOut. Свойству UpdateObject компонента qryInOut присвоим значение updInOut. В свойстве updInOut.InsertSQL разметим код, приведенный в .

Листинг

|  |
| --- |
| // Объявим прагму для проверки успешности выполнений последней операции  #def ErrMsg(msg, rlb) #{  // проверим, возникла ли ошибка после выполнения операции  if (@rc = lastsqlerr()) {  // если переданный прагме параметр rlb равен 1, то  // необходимо откатить транзакцию  #ifdef(rlb,1)  rollback work Wrk;  #endif  usersend(-1, #msg + ' ['+@rc+'] '+ sqlermsg(@rc));  retcode(-8001);  quit;  }  #}  var @rc = 0, @rn = long(0), @id = dword(0);  fix all;  // объявим транзакцию на таблицах CARDS, BOOKS  begin work Wrk table CARDS, BOOKS;  #ErrMsg('Не могу начать транзакцию', 0)  insert into CARDS(STUDENT,BOOK,DATE\_OUT,DATE\_RTN)  values(:STUDENT,:BOOK,:DATE\_OUT,:DATE\_RTN);  #ErrMsg('Ошибка добавления в CARDS', 1)  @rn = recnu(); // recno добавленной записи  @id = svalue(); // значение суррогатного поля (ID)  update BOOKS set CARD = @id  where ID = :BOOK;  #ErrMsg('Ошибка редактирования BOOKS', 1)  commit work Wrk;  #ErrMsg('Не могу завершить транзакцию', 1)  retcode(@rn); |

В свойстве updInOut.UpdateSQL разметим код, приведенный в .

Листинг

|  |
| --- |
| #def ErrMsg(msg, rlb) #{  if (@rc = lastsqlerr()) {  #ifdef(rlb,1)  rollback work Wrk;  #endif  usersend(-1, #msg + ' ['+@rc+'] '+ sqlermsg(@rc));  retcode(-8001);  quit;  }  #}  var @rc = 0;  fix all;  begin work Wrk table CARDS, BOOKS;  #ErrMsg('Не могу начать транзакцию', 0)  update CARDS set DATE\_IN = :DATE\_IN  where ID = :ID;  #ErrMsg('Ошибка редактирования CARDS', 1)  update BOOKS set CARD = 0  where ID = :BOOK;  #ErrMsg('Ошибка редактирования BOOKS', 1)  commit work Wrk;  #ErrMsg('Не могу завершить транзакцию', 1)  retcode(0); |

В коде скриптов InsertSQL и UpdateSQL присутствует пример прагмы для проверки успешности выполнения последней операции и обработки ошибок, в случае их наличия. Обратите внимания, после редактирования информации о выданных книгах необходимо обновить основной набор данных qryBase, для чего напишем код обработчика события AfterPostEx компонента qryInOut ().

Листинг

|  |
| --- |
| procedure TfrmBooks.qryInOutAfterPostEx(Sender: THtDataSet; UpdateState: TDataSetState);  begin  qryBase.RefreshRecord;  end; |

Алгоритм работы пользователя со страницей выдачи/возврата реализуем следующим образом: если книга, на которую указывает курсор в компоненте grdMain ещё не выдана, то есть атрибуту CARD текущей записи набора данных qryBase присвоено значение нуль, то на закладке tsInOut («Выдача/Возврат») будут доступными для заполнения компоненты для выбора студента – lcmbStudent, внесение даты выдачи – dbeOut и даты, не позднее которой надо вернуть dbeRet. Если же книга уже выдана, то есть атрибуту CARD текущей записи присвоено значение, отличное от нуля, то на закладке будут доступна для редактирования только дата возврата – dbeIn. Для применения или отмены изменений на странице будем использовать компонент DBNavigator1. По нажатию на кнопку btnToday компоненту dbeIn будем присваивать текущую дату.

Обратите внимание, что перед удалением книги необходимо проверять, не выдана ли она кому-либо. Для реализации такой проверки используйте как пример редактирование списка студентов.

В компоненте qryInOut создадим ряд полей:

* qryInOutID для ID карточки выдаваемой книги, тип поля - TIntegerField, свойству FieldName присвоим значение ID;
* qryInOutSTUDENT для студента, которому выдается книга, тип поля - TIntegerField, свойству FieldName присвоим значение STUDENT;
* qryInOutDATE\_OUT для даты выдачи, тип поля - THtDateField, свойству FieldName присвоим значение DATE\_OUT, EditMask значение «!99/99/9999;1;»;
* qryInOutDATE\_RTN для даты, не позднее которого надо вернуть книг , тип поля - THtDateField, свойству FieldName присвоим значение DATE\_RTN, EditMask значение «!99/99/9999;1;»;
* qryInOutDATE\_IN для даты возврата книги, тип поля - THtDateField, свойству FieldName присвоим значение DATE\_IN, свойству EditMask присвоим значение «!99/99/9999;1;»;
* qryInOutBOOK для ID (ISBN) книги, тип поля - TStringField, свойству FieldName присвоим значение BOOK;
* qryInOutSTUDENT\_LOOKUP используется как LookUp-поле, тип поля - TStringField, свойству FieldName присвоим значение STUDENT\_LOOKUP, свойству FieldKind присвоим значение fkLookUp, свойству LookupDataSet присвоим значение dmDb.smStudents, KeyFields значение STUDENT, LookupKeyFields значение ID, LookupResultField значение NAME. Обратите внимание, в качестве LookUp набора данных использован компонент smStudents, объявленный и наполянемый данными в модуле dmDatas.

По нажатию на кнопку sbExport будем осуществлять экспорт информации из таблицы BOOKS в файл формата dbf, для чего будем использовать компоненты dlgSaveDBF, dbfExport. Код обработчика нажатия на кнопку представлен в .

Листинг

|  |
| --- |
| procedure TfrmBooks.sbExportClick(Sender: TObject);  var Stm: TFileStream;  RecCnt: Integer;  Begin  // выберем файл, в который будем осуществлять экспорт  if dlgSaveDBF.Execute then  begin  RecCnt := qryBase.RecordCount;  Stm := TFileStream.Create(dlgSaveDBF.FileName, fmCreate);  try  dbfExport.MoveTo(Stm);  finally  Stm.Free;  end;  ShowMessage(Format('Экспорт завершен (выгружено записей - %d).', [RecCnt]));  end;  end; |

По нажатию на кнопку sbImport будем осуществлять импорт информации из таблицы BOOKS в файл формата dbf, для чего будем использовать компоненты dlgOpenDBF, dbfImport, dspImport, spImport.

Компонент dspImport предназначен для пакетного добавления записей в таблицу. Свойству dspImport.Src присвоим значение dbfImport, dspImport.BaseName значение HtBase, dspImport.TableName значение \_books. Создадим в компоненте dspImport поля для работы с атрибутами таблицы. Поля компонента приведены в .

Таб. . Поля компонента dspImport

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название поля (Name) | Тип(HtType) | Уникальность (Key) | Может ли быть не заполнено (Nullable) |
|  | ID | etChar | hkUnique | False |
|  | YEAR | etInt | hkNone | True |
|  | AUTHOR | etChar | hkNone | True |
|  | NAME | etChar | hkNone | True |

В поле SQL компонента spIpmport разместим код, приведенный в .

Листинг

|  |
| --- |
| fix all;  // если книга с таким ISBN уже есть, то  // переписываем год, ФИО автора и название  update BOOKS a  from \_books b  set a.YEAR = b.YEAR,  a.AUTHOR = b.AUTHOR,  a.NAME = b.NAME  where a.ID = b.ID;  // а если ISBN-а нет, то добавляем книгу  insert into BOOKS(ID,YEAR,AUTHOR,NAME)  select ID,YEAR,AUTHOR,NAME  from \_books  where ID not in (select ID from BOOKS);  fix all; |

Алгоритм импорта данных в таблицу BOOKS реализуем следующим образом. Выбор файла-источника данных для импорта будем предоставлять посредством компонента dlgOpenDBF. Информацию из файла будем загружать в компонент dbfImport, который, в свою очередь будет являеться источником данных для компонента dspImport. Поля компонента dspImport созданы заранее, поэтому можно воспользоваться наиболее простым и быстрым способом наполнения данными таблицы \_book, создаваемой компонентом – при помощи метода Move. Обновление таблицы BOOKS информацией из таблицы \_book, полученной в результате предыдущего шага, осуществляется при помощи компонента spIpmport.

Импорт будем инициировать по нажатию на кнопку spIpmport. Код обработчика представлен в .

Листинг

|  |
| --- |
| procedure TfrmBooks.sbImportClick(Sender: TObject);  var RecCnt: Integer;  begin  dbfImport.Close;  // выберем файл-источник данных для импорта  if dlgOpenDBF.Execute then  begin  // наполним данными компонент dbfImport  dbfImport.TableName := dlgOpenDBF.FileName;  dbfImport.Open;  RecCnt := dbfImport.RecordCount;  if dbfImport.Active then  begin  if dspImport.Active then  dspImport.Close;  // создадим временную таблицу  dspImport.Move;  // перенесем данные из временной таблицы в постоянную  spImport.ExecNoSelect;  // обновим основной набор данных  qryBase.Requery;  end;  ShowMessage(Format('Импорт завершен (добавлено записей - %d).',[RecCnt]));  end;  end; |

# Реестр «Студенты»

При нажатии на пункт меню главной формы Реестр ->Студенты открывается (или становится активной, если была открыта ранее) форма «Книги». Внешний вид формы представлен на .

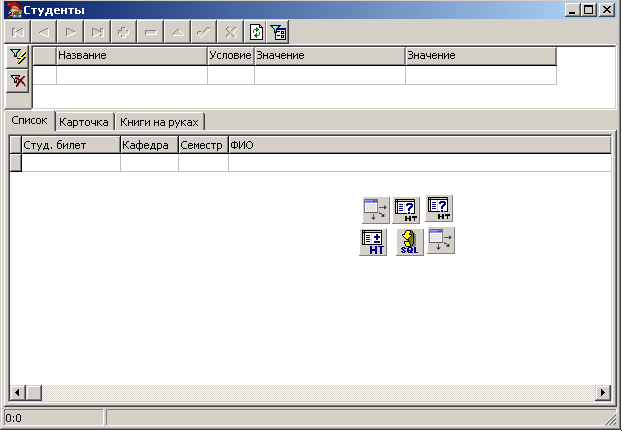


Рис. 125. Внешний вид формы (среда разработки)

Объявление формы располагается в модуле Students. Список основных компонентов, отличных от компонентов формы-предка, располагающихся на форме, представлен в .

Таб. . Компоненты формы frmStudents

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название | Тип | Назначение |
|  | Parser | TSqlParser | Предназначен для преобразования кода |
|  | updMain | THtUpdateSQL | Реализует редактирование/добавление/ удаление информации в таблице STUDENTS |
|  | dbeFIO | TDBEdit | Предназначено для добавления/ редактирования ФИО студента в таблице STUDENTS |
|  | dbeID | TDBEdit | Предназначено для добавления/ редактирования номера студенческого билета в таблице STUDENTS |
|  | lcmbDept | TDBLookupComboBox | Предназначено для добавления/ редактирования номера кафедры в таблице STUDENTS |
|  | cmbTern | TDBComboBox | Предназначено для добавления/ редактирования номера семестра в таблице STUDENTS |
|  | tsBooks | TTabSheet | Является страницей просмотра информации о книгах, выданных студенту |
|  | qryBooks | THtQuery | Обеспечивает работу с набором данных при просмотре информации о книгах, выданных студенту |
|  | DBGrid2 | TDBGrid | Отображение информации о книгах, выданных студенту |
|  | srcBooks | TDataSource | Предназначен для связи компонентов qryBooks и DBGrid2 |
|  | Filter | ThtxDbFilter | Предоставляет возможность фильтрации списка студентов по фамилии |
|  | sbFilter | TSpeedButton | Отображает/скрывает панель с фильтром |
|  | sbApplyFilter | TSpeedButton | Применяет фильтрацию набора данных |
|  | sbRemoveFilter | TSpeedButton | Отменяет фильтрацию набора данных |

В свойство SQL компонента qryBase поместим текст SQL запроса, приведенный в .

Листинг

|  |
| --- |
| fix all;  select a.recno,a.ID,a.DEPT,a.TERN,a.FIO,b.ID  from STUDENTS a, STUDENTS b  where b.ID in(  select STUDENT from CARDS where DATE\_IN = 0  )  and b.ID \*= a.ID  // #Filter предназначен для обеспечения фильтрации информации,  // см подробнее ниже  #Filter  order by a.ID; |

В запросе использованы две таблицы, поэтому необходимо указать qryBase.MainAlias, присвоим этому свойству значение “a”. По этой же причине все манипуляции с данными (удаление, редактирование, добавление) необходимо осуществлять посредством компонента UpdMain. Свойству UpdateObject компонента qryBase присвоим значение UpdMain. В свойствах UpdMain.InsertSQL, UpdMain.DeleteSQL, UpdMain.ModifySQL разместим код, приведенный в , , .

Листинг

|  |
| --- |
| var @rn = 0, @rc = 0;  fix all;  insert into STUDENTS(ID,DEPT,TERN,FIO)  values (:a.ID,:a.DEPT,:a.TERN,:a.FIO);  @rn = recnu();  if(@rc = lastsqlerr()) {  retcode(@rc);  usersend(-1, 'Ошибка добавления в STUDENTS ['+@rc+'] '+ sqlermsg(@rc));  quit;  }  retcode(@rn);  fix all;  select a.recno  from STUDENTS a, STUDENTS b  where a.recno = @rn  and b.ID in(select STUDENT from CARDS where DATE\_IN = 0)  and b.ID \*= a.ID  order by a.ID; |

Листинг

|  |
| --- |
| var @rn = 0, @rc = 0;  fix all;  begin work Wrk table STUDENTS, BOOKS;  if(@rc = lastsqlerr()) {  usersend(-1, 'Не могу начать транзакцию ['+@rc+']: '+ sqlermsg(@rc));  retcode(-8001);  quit;  }  // Перед удалением записи о студенте необходимо проверять, не числятся ли  // за студентом книги  select \* from CARDS where STUDENT = :a.ID and DATE\_IN = 0;  if(reccount()) {  rollback work Wrk;  usersend(-1, 'Нельзя удалить: у студента есть задолженность');  retcode(-8001);  quit;  }  delete from STUDENTS where ID = :a.ID;  if(@rc = lastsqlerr()) {  rollback work Wrk;  usersend(-1, 'Ошибка удаления из STUDENTS ['+@rc+']: '+ sqlermsg(@rc));  retcode(-8001);  quit;  }  commit work Wrk;  if(@rc = lastsqlerr()) {  rollback work Wrk;  usersend(-1, 'Не могу завершить транзакцию ['+@rc+']: '+ sqlermsg(@rc));  retcode(-8001);  quit;  }  retcode(0); |

Листинг

|  |
| --- |
| var @rc = 0;  fix all;  update STUDENTS set  DEPT = :a.DEPT,  TERN = :a.TERN,  FIO = :a.FIO  where ID = :a.ID;  @rc = lastsqlerr();  retcode(@rc);  if(@rc) quit;  fix all;  select a.recno  from STUDENTS a, STUDENTS b  where a.ID = :a.ID  and b.ID in(select STUDENT from CARDS where DATE\_IN = 0)  and b.ID \*= a.ID  order by a.ID; |

В компоненте qryBase создадим ряд полей:

* qryBasearecno для поля recno (таблицы a), тип поля - TIntegerField, свойству FieldName присвоим значение a.recno;
* qryBaseID для студенческого билета (таблицы a), тип поля - TIntegerField, свойству FieldName присвоим значение a.ID, свойству DisplayLabel присвоим значение «Студ. билет»;
* qryBaseTERN для номера семестра (таблицы a), тип поля - TSmallintField, свойству FieldName присвоим значение a.TERN, свойству DisplayLabel присвоим значение «Семестр»;
* qryBaseFIO для ФИО студента (таблицы a), тип поля - TStringField, свойству FieldName присвоим значение a.FIO, свойству DisplayLabel присвоим значение «ФИО»;
* qryBasebID для студенческого билета (таблицы b), тип поля - TIntegerField, свойству FieldName присвоим значение b.ID;
* qryBaseDEPT\_LOOKUP для номера кафедры, тип поля - TStringField, при редактировании информации посредством компонента lcmbDept на закладке tsCard будет использоваться как LookUp-поле, свойству FieldName присвоим значение DEPT\_LOOKUP, свойству FieldKind присвоим значение fkLookUp, свойству LookupDataSet присвоим значение dmDb.smDepts, KeyFields значение a.DEPT, LookupKeyFields значение ID, LookupResultField значение NAME. Обратите внимание, в качестве LookUp набора данных использован компонент smDepts, объявленный и наполянемый данными в модуле dmDatas.

Для полей a.ID, a.DEPT, a.TERN, a.FIO создадим столбцы (TColumn) в компоненте grdMain (при этом необходимо указать название поля в каждом создаваемом компоненте в свойстве FieldName). Назначим событию grdMain.OnDrawColumnCell обработчик. Код обработчика приведен в .

Листинг

|  |
| --- |
| procedure TfrmStudents.grdMainDrawColumnCell(Sender: TObject; const Rect: TRect;  DataCol: Integer; Column: TColumn; State: TGridDrawState);  Begin  // если у студента есть выданная в настоящий момент книга,  // отобразим строку жирным шрифтом  if qryBasebID.AsInteger <> 0 then  grdMain.Canvas.Font.Style:= [fsBold];  grdMain.DefaultDrawColumnCell(Rect, DataCol, Column, State);  end; |

Для навигации по строкам таблице будем использовать компонент DbNavigator, объявленный в предке класса. Также этот компонент предоставляет стандартные возможности добавления, редактирования, удаления информации из набора данных.

По нажатию на кнопку sbFilter будем отображать фильтр (или скрывать его, если он уже отображен). Код обработчика события нажатия на кнопку приведен в .

Листинг

|  |
| --- |
| procedure TfrmStudents.sbFilterClick(Sender: TObject);  begin  if pnlFilter.Visible then  // скрыть панель с фильтром  pnlFilter.Hide  else  // отобразить панель с фильтром  pnlFilter.Show;  end; |

В компоненте Filter создадим условие фильтрации «ФИО студента». Полю FieldName созданного условия присвоим значение a.FIO, DisplayName – «ФИО студента». По нажатию на кнопку sbApplyFilter будем применять условие фильтрации(подробнее об реализации фильтрации информации см ниже), sbRemoveFilter – отменять фильтрацию информации. Код обработчиков нажатия на кнопки приведен в .

Листинг

|  |
| --- |
| // применить фильтр  procedure TfrmStudents.sbApplyFilterClick(Sender: TObject);  begin  FilterApply := True;  qryBase.Requery;  end;  // отменить фильтр  procedure TfrmStudents.sbRemoveFilterClick(Sender: TObject);  begin  FilterApply := False;  qryBase.Requery;  end; |

Компонент Parser предназначен для преобразования кода SQL запроса. Свойству Parser компонента qryBase присвоим значение Parser (ссылку на компонент Parser типа TSqlParser). В компоненте Parser создадим функцию Filter. Назначим обработичк событию OnSubst функции Filter. Код обработчика приведен в .

Листинг

|  |
| --- |
| procedure TfrmStudents.ParserFunctionsFilterSubst(Sender: TCustomSqlParser;  const OutStream: TOutStream; var Handled: Boolean);  var FilterStr: String;  begin  FilterStr := '';  if FilterApply then  // сформируем строку с условием фильтрации  FilterStr := 'and ' + Filter.MakeText;  // впишем строку вместо конструкции #Filter в запросе  OutStream.WriteStr(FilterStr);  end; |

Альтернативную возможность редактирования набора данных реализуем на странице tsCard, где разметим компоненты, предназначенные для редактирования и добавления информации о книгах в таблицу STUDENTS - dbeFIO, dbeID, lcmbDept, cmbTern. Свойству DataSource каждого из этих компонентов присвоим значение srcMain, свойству DataDield значения a.FIO, a.ID, DEPT\_LOOKUP, a.TERN соответственно.

Таким образом, обе страницы (и tsGrid, и tsCard) используют один и тот же набор данных (qryBase).

На странице tsBooks разметим компоненты, предназначенные для просмотра информации о выданных в настоящий момент книгах, для этого необходимы компоненты qryBooks, srcBooks, DbGrid2. В свойстве qryBooks.SQL разместим код, приведенный в .

Листинг

|  |
| --- |
| fix all;  select \* from BOOKS  where ID in(  select BOOK from CARDS  where STUDENT = :a.ID  and DATE\_IN = 0  )  order by ID; |

В компоненте qryBooks создадим ряд полей:

* StringField1 для ISBN книги, тип поля - TStringField, свойству FieldName присвоим значение ID, свойству DisplayLabel присвоим значение «ISBN», EditMask значение «999-9-99-999999-9»;
* qryBaseCARD для номера карточки выдачи книги, тип поля - TIntegerField, свойству FieldName присвоим значение CARD;
* qryBaseLOST для признака «Утеряна», тип поля - TIntegerField, свойству FieldName присвоим значение LOST, свойству DisplayLabel присвоим значение «Утеряна»;
* qryBaseYEAR для ISBN книги, тип поля - TSmallintField, свойству FieldName присвоим значение YEAR, свойству DisplayLabel присвоим значение «Год издания»;
* qryBaseAUTHOR для автора книги, тип поля - TStringField, свойству FieldName присвоим значение AUTHOR, свойству DisplayLabel присвоим значение «Автор»;
* qryBaseNAME для названия книги, тип поля - TStringField, свойству FieldName присвоим значение NAME, свойству DisplayLabel присвоим значение «Название».

Для отображения книг, находящихся у студента, запись о котором выделена на странице tsGrid, свойству DataSource компонента qryBooks присвоим значение srcMain.

Свойству DataSource компонента srcBooks присвоим значение qryBooks.

Для полей ID, LOST, YEAR, AUTHOR, NAME создадим столбцы (TColumn) в компоненте DBGrid2 (при этом необходимо указать название поля в каждом создаваемом компоненте в свойстве FieldName). Свойству DataSource компонента DBGrid2 присвоим значение srcBooks для связи с набором данных.

# Отчет «Студенты - должники»

При нажатии на пункт меню главной формы Отчеты ->Студенты-должники открывается форма выбора вида отчета – «Все должники» или «Только потерявшие книги». Внешний вид формы представлен на .

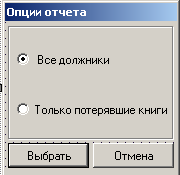


Рис. 126. Внешний вид формы (среда разработки)

Форма реализована в модуле fRepOptions. Список компонентов, располагающихся на форме, представлен в .

Таб. . Компоненты формы fRepOptions

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название | Тип | Назначение |
|  | rgLost | TSqlParser | Предназначен для выбора вида отчета |
|  | btnOk | TBitBtn | Открывает отчет выбранного вида |
|  | btnCancel | TBitBtn | Закрывает форму |

Назначим обработчик нажатию на кнопки btnOk, btnCancel ().

Листинг

|  |
| --- |
| procedure TfrmRepOptions.btnCancelClick(Sender: TObject);  begin  ModalResult := mrCancel;  end;  procedure TfrmRepOptions.btnOkClick(Sender: TObject);  begin  // запомним указанные условия построения отчета  // IsLost – public поле класса TfrmRepOptions, реализующего форму  IsLost:= rgLost.ItemIndex;  ModalResult := mrOk;  end; |

В случае если форма выбора вида отчета будет закрыта с результатом mrOk, откроем форму просмотра отчета о студентах, не сдавших книги. Это поведение приложения реализуем в обработчике события нажатия на пункт меню «Студенты-должники» (см. ).

Листинг

|  |
| --- |
| procedure TMainForm.miDebtStudentsClick(Sender: TObject);  begin  if frmRepOptions.ShowModal = mrOk then  TfrmDebtStudents.FindShow;  end; |

Форма «Студенты-должники» реализована в модуле DebtStudents. Внешний вид формы в среде разработки приведен на .

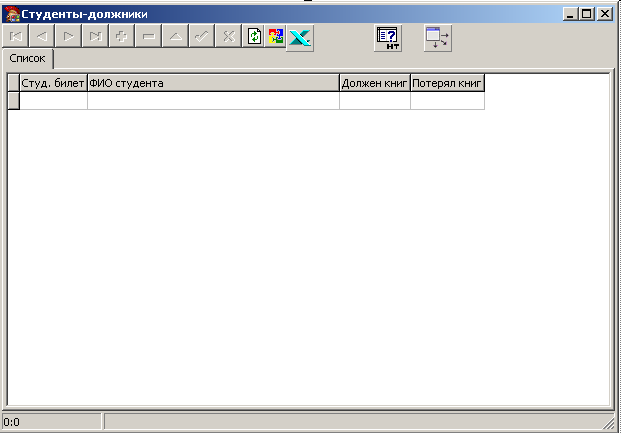


Рис. 127. Внешний вид формы (среда разработки)

Предком класса, реализующего форму, также является класс TfrmReestr. Список компонентов, отличных от компонентов класс-предка, располагающихся на форме, представлен в ..

Таб. . Компоненты формы frmDebtStudents

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название | Тип | Назначение |
|  | ExcelExport | TExcel | Предназначен для экспорта в Excel |
|  | sbExcel | TBitBtn | Инициирует экспорт в Excel |

Свойству SQL компонента qryBase присвоим код, приведенный в .

Листинг

|  |
| --- |
| // воспользуемся прагмой для учета выбора вида отчета  #def LOST\_ONLY :LOST  fix all;  select a.ID as ID, a.DEPT as DEPT, a.FIO as FIO,  count(\*) as DEBT\_CNT, sum(b.LOST) as LOST\_CNT  from STUDENTS a, BOOKS b, CARDS c  where #ifdef (LOST\_ONLY,1)  b.LOST = 1 and  #endif  a.ID = c.STUDENT and  c.ID = b.CARD  group by a.ID; |

В запросе используются прагмы #def, #ifdef и #endif для учета выбора вида отчета, совершенного пользователем. В качестве конструкции подстановки будем использовать параметр :LOST, который создадим в компоненте qryBase(см. ).

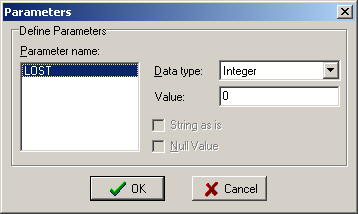


Рис. 128. Параметр LOST

Значение параметра будем проставлять в обработчике события qryBase.BeforeRunSQL. Код обработчика приведен в

Листинг

|  |
| --- |
| procedure TfrmDebtStudents.qryBaseBeforeRunSql(DataSet: TDataSet);  begin  qryBase.ParamByName('LOST').AsInteger := frmRepOptions.IsLost;  end; |

По нажатию на кнопку sbExcel будем эспортировать информацию в Excel. Код обработчика нажатия на кнопку приведен в .

Листинг

|  |
| --- |
| procedure TfrmDebtStudents.sbExcelClick(Sender: TObject);  begin  with ExcelExport do  begin  Connect;  if Connected then  begin  ExportGrid(grdMain, true, 1, qryBase.RecordCount);  Maximize;  end;  DisConnect;  end;  end; |

# Демонстрация работы приложения

Скомпилируем и запустим приложение. Установим связь с базой данных и откроем реестр кафедр. Внешний вид приложения представлен на .

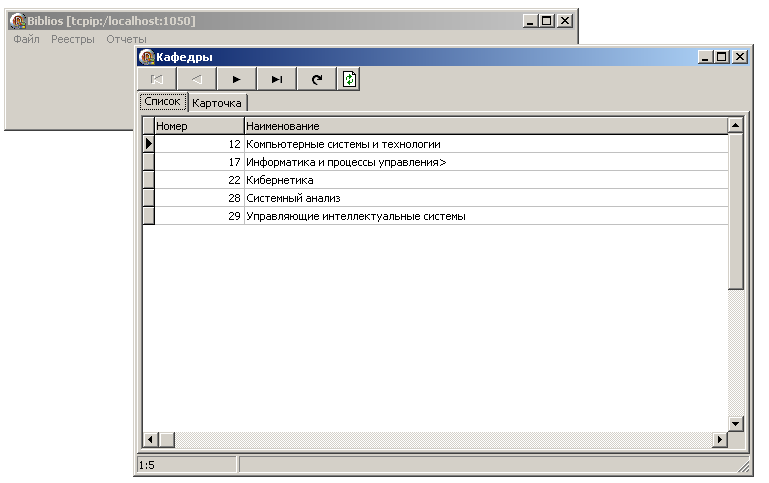


Рис. 129. Реестр «Кафедры»

Откроем реест студентов посредством пунктов меню Реестры -> Студенты и добавим запись в таблицу ().

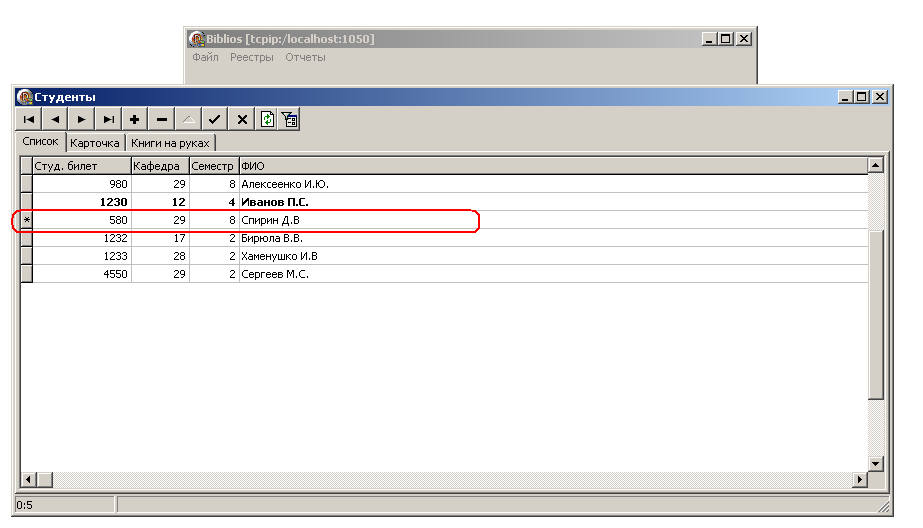


Рис. 130. Добавление записи

Добавим ещё одного студента – Иванова В.Ф., после чего воспользуемся фильтрацией, в качестве условия выберем конструкцию «like», в поле «Значение» впишем «Иванов» и нажмем «Применить фильтр». Внешний вид приложения приведен на .

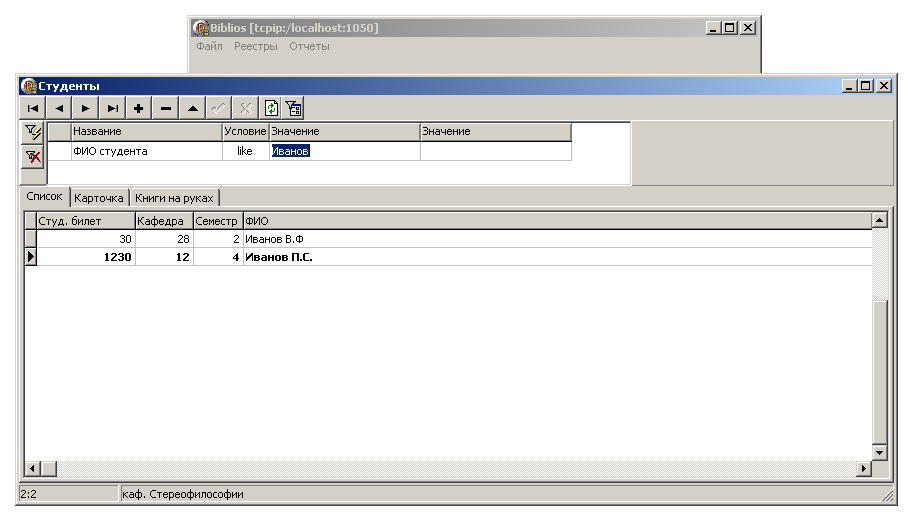


Рис. 131. Применение фильтра

Отредактируем год издания книги «Практические занятия по синтаксису и пунктуации» при помощи страницы «Карточка» ( - ).

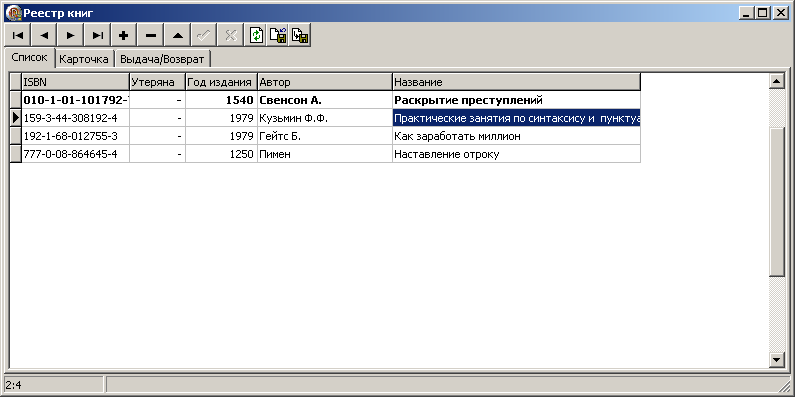


Рис. 132. Реестр «Книги» до редактирования данных

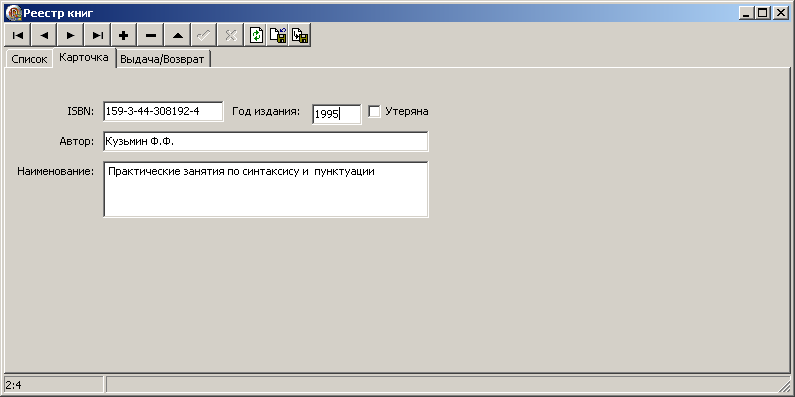


Рис. 133. Редактирование года издания

Внесем в базу информацию о выдаче этой книги студенту Бирюла В.В. ().

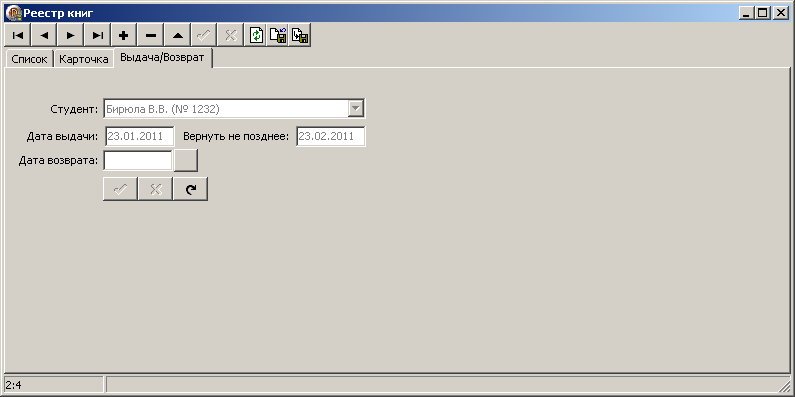


Рис. 134. Информация о картчке выдачи внесена

Выгрузим отчет о студентах должниках (выберем пункт «Все должники») и экспортируем информацию в Excel ( - ).

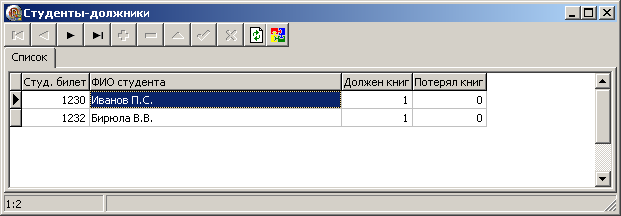


Рис. 135. Отчет по студентам – должникам сформирован

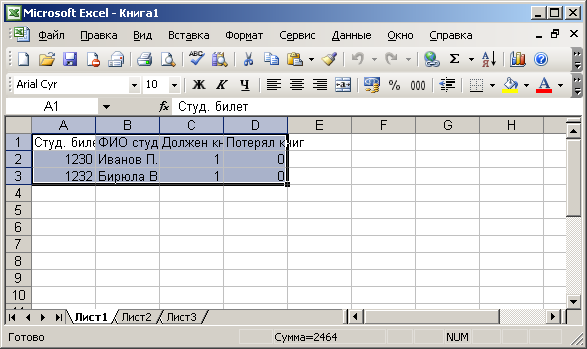


Рис. 136. Отчет по студентам – должникам выгружен в Excel

Информацию из реестра книг экспортируем в файл формата dbf ( - ):

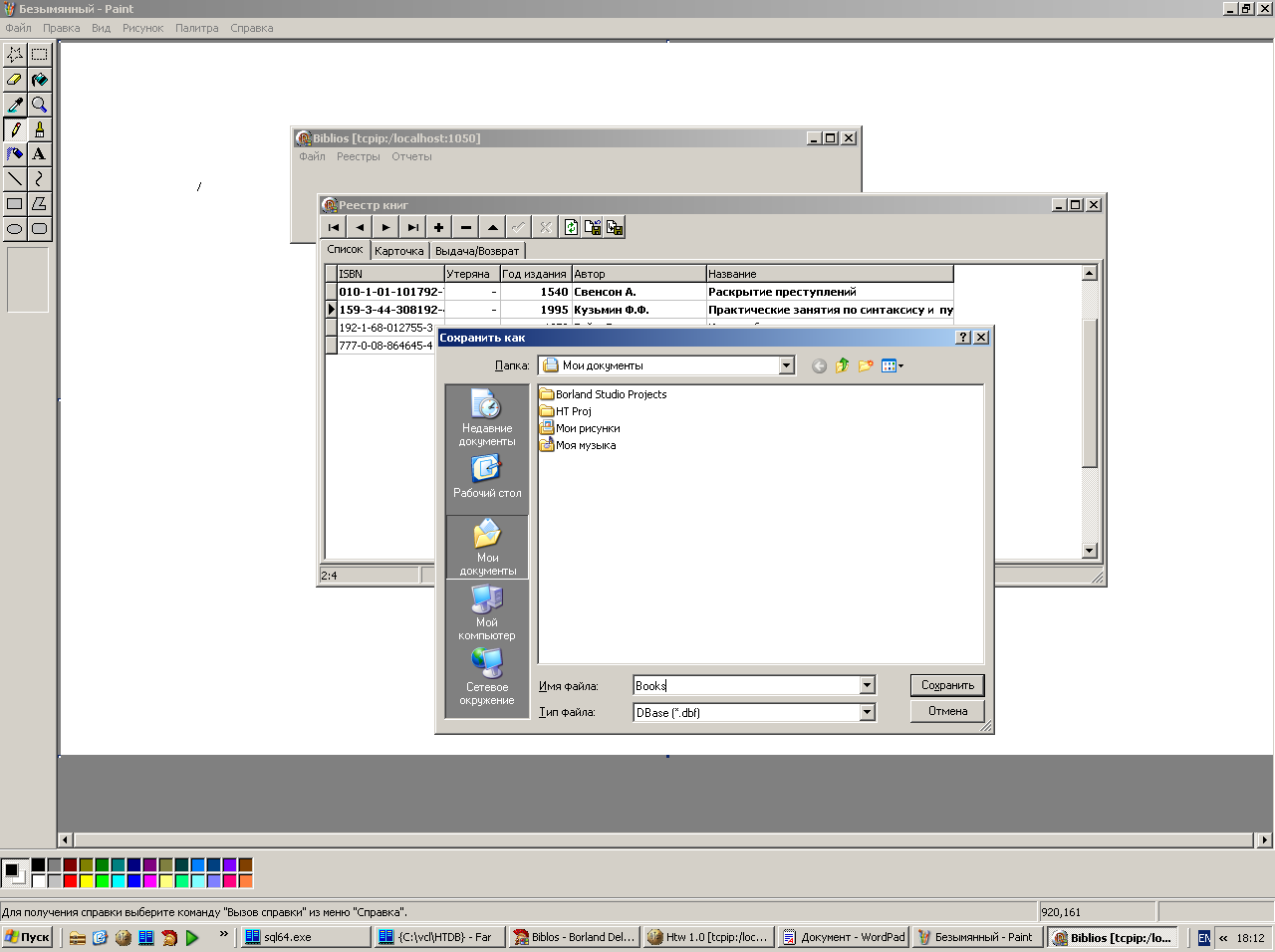


Рис. 137. Выбор файла при экспорте

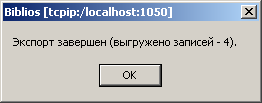


Рис. 138. Сообщение о успшеном завершении экспорта данных

Удалим книгу «Наставление отроку» ( - ).

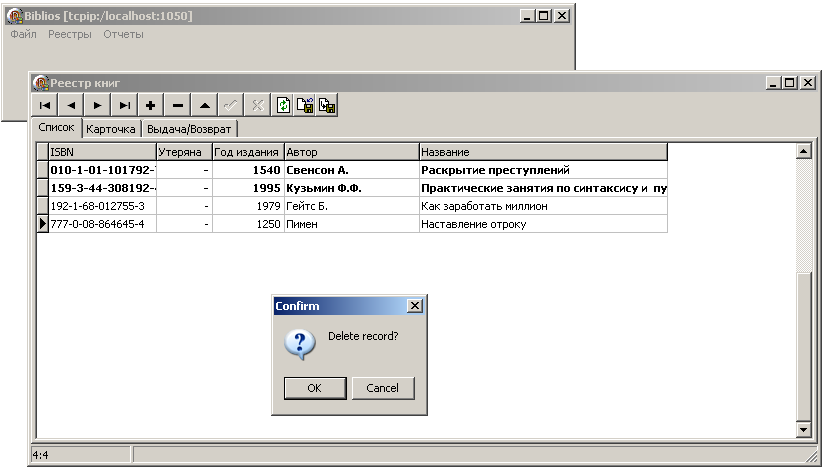


Рис. 139. Удаление книги «Наставление отроку»

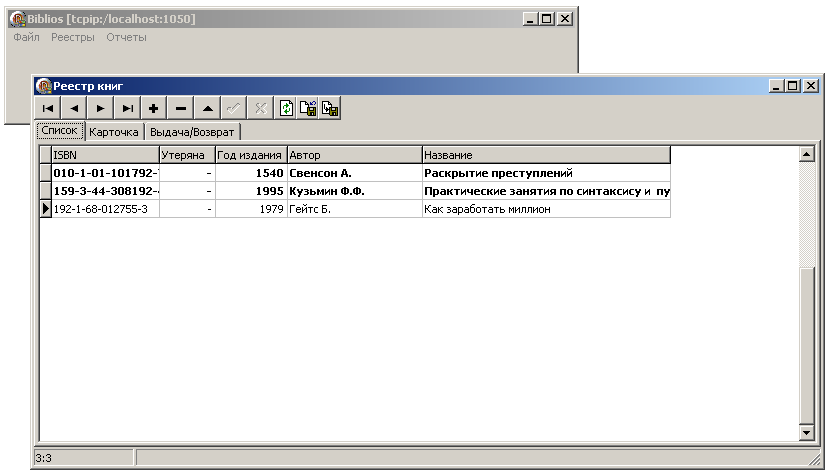


Рис. 140. Книга удалена

Импортируем информацию из файла Books.dbf. Убедимся в том, что информация об удаленной книге снова присутствует в базе данных ( - ).

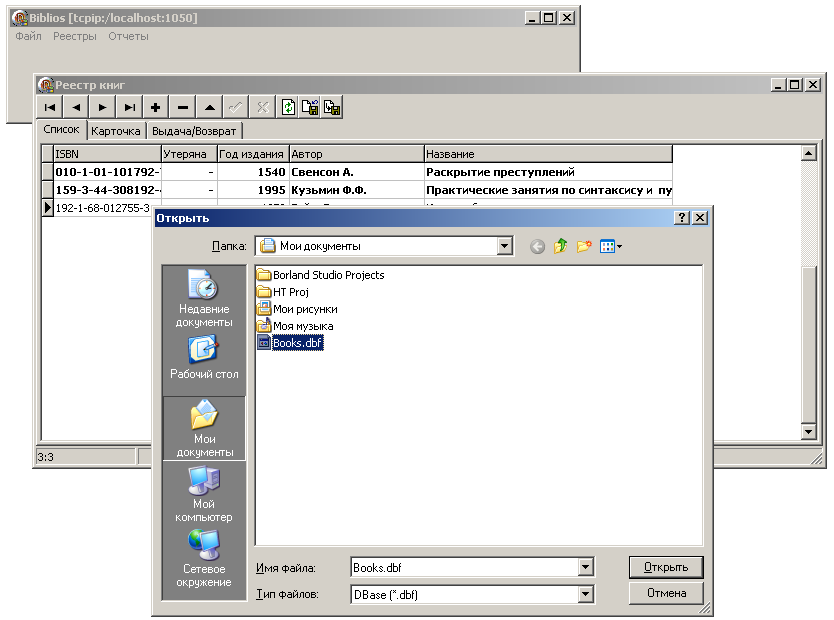


Рис. 141. Выбор файла при импорте информации

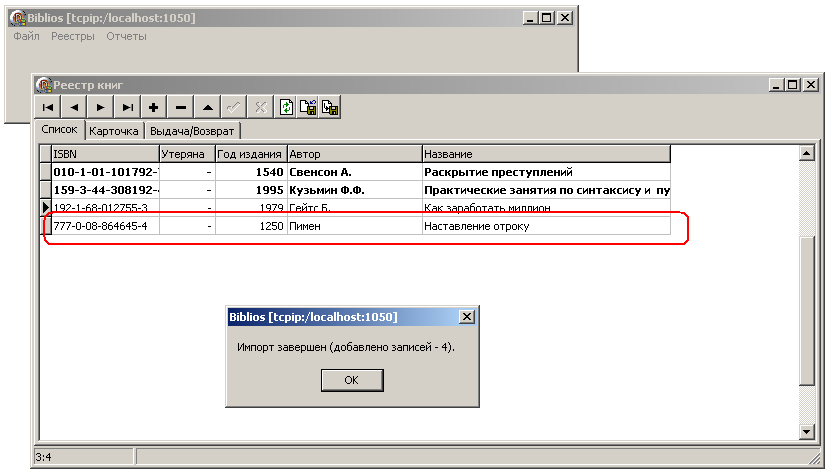


Рис. 142. Сообщение об успешности завершения импорта

Проставим флаг «Утеряна» книге «Раскрытие преступлений» ().

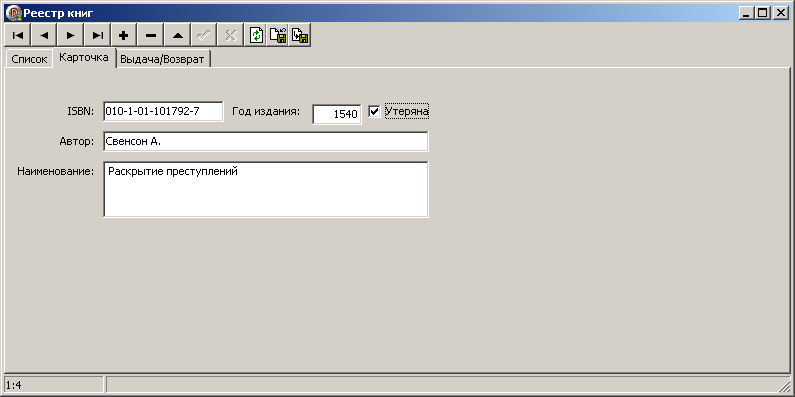


Рис. 143. Проставление флага «Утеряна»

Ещё раз сформируем отчет «Студенты-должники», выберем вариант «Только потерявшие книги» ().

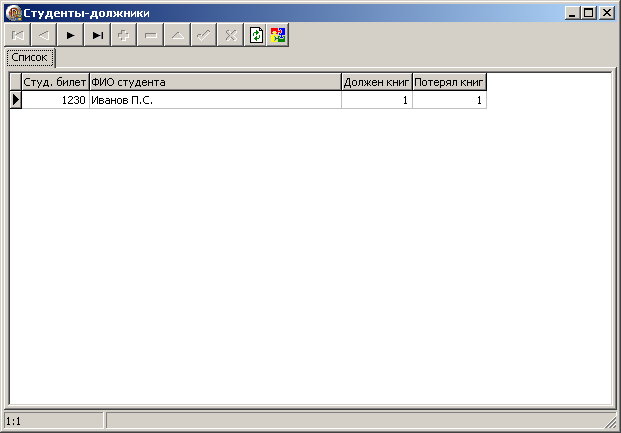


Рис. 144. Отчет по студентам, потерявшим книги, сформирован

# Справочник по компонентам

В данноме разделе представлен материал по ряду компонентов Delphi, предназначенных для разработки клиент-серверных приложений, серверная часть которых представлена СУБД HyTech. В курсе приведена информация по следующим компонентам:

* THtBase – предназначен для обеспечения связи с базой данных HyTech;
* THtQuery – обеспечивает доступ к данным в таблицах и представления БД HyTec. Обладает кэш-буфером начитываемых записей.
* THtStoredProc – предназначен для выполнения произвольных запросов, не требующих чтения более одной строки результата.
* TCustomMemTable – является наследником стандартного класса TDataSet, может использоваться как Lookup DataSet или кэш данных. Позволяет сохранять/восстанавливать данные в потоке/файле, копировать данные и описания полей из других DataSet, сортировать строки по любому пользовательскому алгоритму.
* TMemTable – представляет собой набор данных, который хранит в памяти записи, при этом вычисляемые поля хранятся в каждой записи. Компонентом поддерживается журналирование измений.
* TCustomSqlMem – предназначен для кеширования результатов запросов. При активизации создает THtQuery, выполняет в нем запрос, копирует в себя все данные и уничтожает Query, удобен для кеширования на клиенте мелких справочников и использования в качестве lookup DataSet.
* TSqlMem – является наследником TCustomSqlMem, позволяет назначить обработчики событий создаваемого компонентом THtQuer.
* THtPump – предназначен для буферизованного заполнения таблицы HyTech, подразумевает создание принимающей таблицы и запроса информации из неё/
* THtDsPump – является наследником THtPump, обладает несколькими дополнительными свойствами и методами для загрузки информации из различных наборов данных.
* TDataMove – предназначен выгрузки информации из БД Hytech в формат dbf на рабочей станции пользователя, при этом необходимо построить соответствия полей источника и полей пункта назначения, достоинством компонента является возможность использования выражений.
* TDBFMove – является визуальным классом-наследником TDataMove.
* TTXTMove – предназначен для выгрузки информации из DataSet`a в текстовый файл.
* THTDBFTable –является набором данных для БД в формате DBF, не использующим BDE .
* TMacList – предназначен для работы с макросами, хранит текст макросов в DFM файле.
* TMacrosList – предназначен для работы с макросами, реализует хранилище макросов как отдельный от DFM файл
* TSQLParser – является препроцессором SQL.
* THtUpdateSQL – позволяет назначить произвольные модфикационные SQL на THtDataSet, выполняет свои SQL выражения при операциях THtDataSet.Post и THtDataSet.Delete, возвращая результат выполнения THtDataSet'у.
* TExcel – презназначен для выгрузки информации в Excel и загрузки её из Excel в БД
* THtxDbFilter - позволяет осуществлять фильтрацию информации, получаемой из БД, условия фильтрации проектируются в процессе разработки приложения, а значения для сравнения и логические отношения между условиями (AND,OR) выбираются пользователем в процессе работы приложения.
* THtxCondition – является условием, применямым в компоненте THtxDbFilter.
* TXTreeView – предназначен для показа ветвящихся иерархических структур на основе информации из базы данных.
* THTree, THTreeView – компоненты используются в паре вместо TXTreeView. THTree кеширует в узлах информацию; при подключении нескольких THTreeView (наследник стандартного TTreeView), обеспечивает их взаимодействие и идентичность отображаемой информации. THTree предназначен для хранения древовидной структуры узлов. THTReeView используется для отображения узлов.

# Закладка HyTech:

# **THtBase**

Для обеспечения связи с базой данных HyTech используется компонент THtBase.

Иерархия классов приведена на .



Рис. Иерархия классов

Для задания пути к базе данных используется свойство DbPath, путь к базе данных, располагающейся локально на порте 1000 будет следующим: ‘tcpip:/localhost:1000’.

В случае, если связь с базой данных необходимо установить при проектировании приложения, свойству Active необходимо присвоить значение ‘true’. Если никаких иных изменений с компонентом произведено не было, на экране появится окно соединения с БД (см. ). В окне уже прописан путь к БД, в качестве имени пользователя по умолчанию проставляется текущий пользователь операционной системы, однако необходимо внести пароль (и изменить пользователя при необходимости).

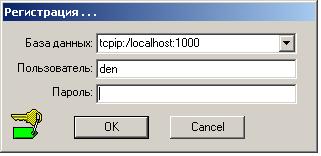


Рис. Окно соединения с БД

Свойства ConnectUser и LoginPrompt используются в случае, когда связь с базой данных устанавливается не в процессе проектирования, а после запуска приложения.

Если при установке связи с БД нет необходимости в отображении окна «Регистрация», приведенного на , то свойству LoginPrompt необходимо присвоить значение false.

Свойству ConnectUser соответствует значение false, если имя пользователя и пароль БД совпадает с именем пользователя и паролем операционной системы. В случае, если реквизиты пользователя в БД и ОС не совпадают и свойству LoginPrompt присвоено значение false, то имя пользователя и пароль необходимо задавать программно через свойства THtBase.UserName и THtBase.Password, свойству ConnectUser присваивать значение true.

Если при соединении с БД необходимо выполнять какой-либо код (например, вставку в таблицу учета сессий пользователей БД), его код необходимо внести в свойство StartSql, например ***insert into sttable values(1)***.

Свойства компонента представлены в .

Таб. Свойства компонента THtBase

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | false | published |
|  | ActiveNotifier | - | public |
|  | AltPswChanel | -21 | published |
|  | BreakTimeOut | 5 | published |
|  | ClientModuleHandle | - | public |
|  | ClientDllVersion | - | public |
|  | Connected | - | public |
|  | ConnectUser | false | published |
|  | DbAlias | - | public |
|  | DbPath | - | published |
|  | DoubleLogin | false | public |
|  | EncryptPassw | false | published |
|  | ErrList | - | public |
|  | ErrToUser | true | published |
|  | HandleMsgWait | true | published |
|  | HtError | - | public |
|  | IdlePeriod | - | published |
|  | LastSql | - | public |
|  | LoginPrompt | true | published |
|  | MacrosList | - | published |
|  | MinPassLen | 0 | published |
|  | Name | HtBase1 | published |
|  | NumVersion | - | public |
|  | OSName | - | public |
|  | Password | - | public |
|  | RtPath | - | public |
|  | SameThread | false | published |
|  | SqlError | - | public |
|  | StartSql | - | published |
|  | StrVersion | - | public |
|  | Tag | - | published |
|  | Transport | - | public |
|  | UserName | - | public |
|  | Version | 0 | public |

Остановимся более подробно на каждом свойстве.

Active: свойству присваивается «истина» в момент активизации компонента.

ActiveNotifier: в свойстве хранится ссылка на компонент, который в настоящий момент воспринимает все события.

AltPswChanel: используя это свойство можно задать реакцию на определенный код ошибки, присланный через usersend. Если код ошибки совпадает со значением AltPswChanel, например, эта ошибка может трактоваться компонентом как ситуация, когда пользователю необходимо сменить пароль (компонент выдаст стандартный диалог смены пароля) – для этого также необходима разработка кода на стороне сервера и использование файла strtuser.sql, находящегося на стороне сервера и выполняющегося при попытке авторизации пользователя.

BreakTimeOut: промежуток времени, заданный в секундах, через который срабатывает событие OnAskBreak, в случае, если компонент (THtBase) ожидает окончания выполнения Ht API функции, вызванной в исполняющем потоке (см. также свойство SameThread, событие OnIdle).

ClientModuleHandle: свойство хранит ссылку на промежуточную динамическую библиотеку, посредством которой работает компонент в настоящий момент.

ClientDllVersion: свойство хранит версию динамической библиотеки, посредством которой работает компонент в настоящий момент.

Connected: свойству присваивается «истина» при успешной установке соединения с сервером. Возможна ситуация, когда connected=true и при этом active = false (например, при проектировании).

ConnectUser: свойство показывает, необходимо ли запускать процедуру авторизации после процедуры установки соединения (в СУБД HyTech это две отдельные операции – установка соединения и авторизация). Если в настройках сервера (sql64.ini) указаны userdefault и passworddefault и свойствам компонента User и Password не присвоены значения, то после установки соединения авторизация происходит под учетными данными, заданными по умолчанию.

DbAlias: содержит имя алиаса БД, если DbPath содержит имя алиаса БД, иначе не содержит значения.

DbPath: содержит либо строку подключения, либо алиас БД. В момент активизации компонента идёт поиск файла hytech.ini (сначала текущий каталог, затем каталог программы, затем по пути, указанным в переменной окружения Path), в файле компонент ищет раздел aliases, если раздел не найден, то значения свойства считается строкой подключения.

DoubleLogin: в свойстве указывается, надо ли предпринимать попытку авторизации незашифрованным паролем в случае, если имеет место шифрования пароля (св. EncryptPassw = true) и попытка авторизации зашифрованным паролем не удалась.

EncryptPassw: в свойстве указывается необходимость шифрования пароля пользователя перед отправкой серверу. Если свойству присвоить значение “истина”, то перед авторизацией вызывается событие OnEncryptPsw , то есть пароль можно зашифровать (это можно использовать, например, для того, чтобы разрешить подключение к серверу только конкретным приложением, в котором реализовано шифрование).

ErrList: свойство предназначено для хранения списка кодов ошибок и сообщений пользователю, которые необходимо выдавать в случае ошибки. Если в StartSql указать прагму #errmsg (#errmsg{список параметров}, список параметров представляет собой пары число = строка, разделенные запятыми или точкой с запятой, список используется THtDataSet'ом для формирования сообщений об ошибках), то при иницииализации THtBase препроцессор SQL занесет этот список в ErrList: сообщения в ErrList.strings, коды ошибок - в ErrList.Objects. ErrList можно также в любой момент редактировать, изменяя список ошибок. Используется ErrList компонентом THtDataSet: если при выполнении запроса код ошибки есть в списке ErrList, то исключение создается с сообщением из ErrList , если же в ErrList - пустая строка, то вместо генерации соответствующего исключения выполняется Abort - операция прерывается без каких-либо сообщений.

ErrToUser: свойство показывает, необходимо ли присылать текстовые сообщения об ошибках клиенту (ErrToUser= true) или только коды ошибок (ErrToUser = false), в любом случае информация об ошибках присылается по каналу CallBack.

HandleMsgWait: свойство показывает, обрабатывать ли сообщения windows во время ожидания результата в случае, если SameThread = false.

HtError: в свойстве сохраняется значение lasthterr() после каждого вызова HyTech API.

IdlePeriod: свойство задает максимальный период между вызовами OnIdle (который вызывается или когда есть сообщения, или через период idleperiod).

LastSql: в свойстве сохраняется текст последнего выполненного SQL запроса.

LoginPrompt: свойство показывает, будет ли вызван диалог подключения при активизации компонента. Если соединение с сервером установить не удалось, то диалог всё равно будет вызван (изменить поведение компонента можно в событии OnConnectError).

MacrosList: в поле указывается ссылка на глобальный макрос лист.

MinPassLen: указывает минимальная длина пароля ( -1 = нет проверять минимальную длину пароля).

NumVersion: свойство хранит номер версии сервера после подключения.

OSName: после подключения к серверу свойство содержит имя операционной системы, под управлением которой запущен HyTech сервер.

Password: свойство хранит пароль для подключения к БД.

RtPath: свойство содержит расшифровку алиаса (после инициализации), если в свойстве DbPath содержится имя алиаса БД.

SameThread : возможна ситуация, когда сам компонент создан в контексте отдельного потока и ещё один дополнительный поток не нужен. В этом случае SameThread надо установить в true, и все вызовы Ht API будут выполняться в том же потоке, что и вызовы метода Call().

SqlError: свойство содержит lastsqlerr() после каждого вызова HyTech API.

StartSql: в свойстве указывается код, который необходимо выполнить в момент подключения, можно задать макросы, объявить и инициализировать глобальные переменные и тп.

StrVersion : содержит версию сервера после подключения к БД.

Transport: свойство содержит «начало» строки подключения (не менее 3-х символов, начиная с первого и заканчивая двоеточием. По умолчанию поддерживаются три вида транспорта: «sql» - работа непосредственно с таблицами HyTech (локальное подключение), «tcpip» - работа с сервером по протоколу tcpip, «file» работа с сервером по файловому протоколу. В HyTech.ini в секции TransDll можно зарегистрировать новые протоколы или переназначить DLL - посредника имеющимся.

UserName: свойство хранит имя пользователя для подключения к БД.

Version: свойство содержит номер сборки сервера после подключения.

Рассмотрим события компонента. Будем останавливаться подробно на событиях, специфичных для HyTech.

* + - 1. OnAdminMsg (TAdminMessageEvent). Вызывается при поступлении административного сообщения от сервера. Принимает два параметра – Sender(TObject) и Code(integer). В параметре Code передается код сообщения.
      2. OnAfterCallHt (TOnCallHtEvent) Вызывается после любого обращения к серверу. Принимает два параметра Sender(TObject) и ProcNo(integer). В параметре ProcNo передается номер процедуры, которую необходимо выолпнить (одна из констант вида DM\_XXX).
      3. OnAfterConnect (TNotifyEvent) Вызывается после попытки установить соединение.
      4. OnAfterDisconnect (TNotifyEvent) Вызывается после разрыва соединения с сервером.
      5. OnAltPassword (TOnAltPassword) Вызывается при смене пароля. Здесь можно шифровать пароль, если установлен EncryptPsw. Принимаем ряд параметров:
         * Sender(TObject);
         * MinLength(integer) – параметр используется для проверки минимальной длины;
         * ParentWND(HWND) – в параметре может быть передан хэндл родительского окна;
         * Caption( string) – параметр используется для указания заголовка окна диалога смены пароля, если используется диалог;
         * X, Y (integer) – параметры используются для регулирования положения окна диалога смены пароля;
         * NewPass(string) – параметр используется для хранения нового пароля;
         * Ok(boolean )- параметр используется для хранения инфомрации об успешности операции.
      6. OnAskBreak (TAskBreakEvent) Вызывается по истечении BreakTimeOut. (см. BreakTimeOut и OnIdle). Принимает два параметра: Sender(TObject) и CancelMode(TCancelMode). Второй параметр может принимать значения cmNothing (отменить прерывание выполнения), cmBreak (прервать выполнение средствами HyTech API) или cmCrash (завершить поток (TerminateThread), в котором выполняется работа с HyTech). Не рекомендуется использовать cmCrash: кроме потери виртуальной памяти клиента из-за TerminateThread, выполнение запроса (в случае работы с сервером) в большинстве случаев все равно будет продолжаться. В случае же работы с локальными файлами велик риск потерять данные и испортить таблицы.
      7. OnBeforeCallHt (TOnCallHtEvent) Вызывается до любого обращения к серверу. Описание события типа TOnCallHtEvent см. выше.
      8. OnBeforeConnect (TNotifyEvent) Вызывается до установки соединения.
      9. OnBeforeDisconnect (TNotifyEvent) Вызывается до разрыва соединения.
      10. OnConnected (TNotifyEvent) Вызыватся при авторизации пользователя. Сюда необходимо поместить методы, которые считаются (разработчиком) частью авторизации, то есть если эти процедуры не выполнены, то считается, что авторизация не прошла.
      11. OnConnectError (TOnConnectError) Вызывается при ошибке соединения, здесь есть параметр-переменная action, в котором можно вернуть дальнейшую реакцию: fail - процедура подключения завершается с ошибкой, abort - прерывается, retry - снова вызывается диалог подключения. Принимает ряд параметров:
          * Sender(TObject);
          * E (Exception ) - исключение;
          * EAddr (Pointer) - адрес возникновения ошибки;
          * Action (TDataAction) - действие после возврата из обработчика.
      12. OnConnecting (TNotifyEvent) Вызывается в процессе процедуры подключения.
      13. OnEncryptPsw (TEncryptEvent) Вызывается при шифровании пароля. Принимает ряд параметров:
          * Sender: TObject;
          * User (string) – содержит имя пользователя;
          * Pass ( string) - – содержит пароль.
      14. OnErrMsg (TMessageEvent) Вызывается при обработке сообщений об ошибках, присланных по каналу колбэка. Принимает два параметра: Sender (TObject) и Mess(string). Предназначен для обработки сообщений, не имеющих дополнительного числового параметра. Во втором параметре передается текс сообщения.
      15. OnException (TExceptionEvent) Вызывается при возникновении исключительной ситуации, обработку которой можно реализовать здесь и остановить распространение исключения. Принимает два параметра: Sender(TObject) и E(Exception) – собственно исключение.
      16. OnGetMacro (TOnGetMacro) Вызывается при обнаружении в теле SQL запроса вызов макроса. Принимает ряд параметров:
          * Sender( TObject)
          * S(string) - содержит на входе имя макроса (без лидирующих сиволов #);
          * Found(boolean) -если замена произведена, параметр должен будет установлен в true, иначе - false.
      17. OnGetParentWND (TOnGetParentWND) Обработчик должен вернуть хэндл родительского окна для диалогов компонента (изменения пароля, диалог устновки соединения). Принимает два параметра: Sender(TObject) и WND(HWND) –хэндл текущего окна.
      18. OnIdle (TIdleEvent) Во время ожидания завершения Ht API функции, вызванной в исполняющем потоке, в главном потоке выполняется обработка сообщений и CallBack вызовов. По-умолчанию обрабатываются сообщения завершения, перерисовки, перемещения, изменения размеров окон и т.п. Сообщения, порождаемые нажатиями кнопок мыши на клиентской части окна, игнорируются. Это позволяет "разморозить" приложение, не позволяя в то же время управлять элементами интерфейса (кнопками, скролбарами и т.п.). Назначив обработчик на это событие, можно разрешить каким-либо оконным элементам интерфеса обрабатывать все сообщения. Обработчик будет вызван как минимум два раза: перед вызовом HyTech API (YieldPos = ypFirst) и после (YieldPos = ypLast). Все прочие вызовы (YieldPos = ypNext) происходят с периодом не менее THtFork.IdlePeriod (мс) или (если THtFork.WithMsgs = true) для обработки сообщений (messages) из очереди процесса. Принимает ряд параметров:
          * Sender(TObject);
          * YieldPos (TYieldPos) – содержит информацию о том, какой раз вызывается обработчик ;
          * CancelMode (TCancelMode);
          * Msg (PMsg) – содержит обрабатываемое сообщение.
      19. OnLocalErr (TMessageEvent) Вызывается при возникновении ошибки клиентских dll (канал получения ошибок клиентских dll), до настоящего времени таких ошибок не зарегистрировано.
      20. OnLoginPrompt (TLoginPromptEvent) Вызывается, если LoginPrompt = true. Может применяться для реализации нестандартных диалогов подключения или выполнения дополнительных действий. Должен установить правильные свойства DbPath, UserName, Password компонента THtBase(Sender) или вернуть false, чтобы отказаться от подключения. Если в обработчике вызван EAbort, то это равносильно отказу от подключения; любой другой Exception обрабатывается событием OnConnectError. Если обработчик не назначен, вызывается встроенный диалог.
      21. OnOutMsg (TMessageEvent) Вызывается при обработке сообщений, присланных по каналу по каналу outmessage.
      22. OnParserError (TOnParserError) Вызывается при возникновении ошибки парсера. Принимает ряд параметров:
          * Sender(TCustomSqlParser);
          * E( EHtSyntaxError) - исключение с информацией об ошибке;
          * Handled( boolean) влияет на распространение ошибки. Если false, ошибка распостраняется дальше (raise), иначе выполняется Abort. Обработка SQL завершается в любом случае.
      23. OnSqlPrepared (TSqlPreparedEvent) Вызывается перед отправкой окончательного SQL кода на сервера. Принимает два параметра: Sender(TObject) и Sql(string), содержит SQL код.
      24. OnUserMsg (TCodeMessageEvent) Вызывается при обработке сообщений присланных по каналу по каналу usersend. Предназначен для обработки вызовов, имеющих дополнительно к тексту числовой параметр. Принимает ряд параметров:
          * Sender: TObject;
          * Code (SmallInt) - содержит числовой параметр сообщения;
          * Mess (string ) - содержит сообщение.

Основные методы компонента представлены в .

Таб. 21 Основные методы компонента THtBase

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | procedure AddRecords(const TableName: string; RecCount, RecSize: dword ; Buffer: Pointer) | Позволяет добавить в таблицу буфер с записями, выполняется посредством вызова API функций для быстрого заполнения временных таблиц или для передачи на сервер файлов. Для иного использовать не следует, так как на вставку, произведенную таким образом, не реагируют триггеры.  Имя таблицы указывается в параметре TableName, количество записей и размер записи в параметрах RecCount, RecSize соответственно, буфер – источник информации в параметре Buffer. |
|  | function AlterPass(const User, OldPass, NewPass: string): THtError | Позволяет сменить пароль пользователя без вызова диалога. Имя пользователя указывается в параметре User, параметр OldPass нужен, если меняется пароль не текущему пользователю, например, при работе администратора, новый пароль указывается в параметре NewPass. Возвращает ошибку в случае неудачи. |
|  | function AlterPassDlg(ParentWND: HWND = 0; Caption: string = ''; X: integer = POS\_CENTER; Y: integer = POS\_CENTER): boolean | Позволяет изменить пароль через диалог, в параметре ParentWND можно передать хэндл окна – родителя, если параметр равен 0, то при создании окна-диалога хэндл родительского окна будет определен автоматически. Заголовок окна диалогоа можно указать в параметре Caption, абсолютные координаты положения окна можно задать параметрами X,Y. Возвращает true в случае успешной смены пароля. |
|  | function AlterPassDlgLen(MinLength: integer = -1; ParentWND: HWND = 0; Caption: string = ''; X: integer = POS\_CENTER; Y: integer = POS\_CENTER) : boolean | Позволяет изменить пароль через диалог с проверкой минимальной длины, если параметр MinLength равен -1, то длина не проверяется. В противном случае кнопка «ОК» на окне диалога будет не активна до тех пор, пока новый пароль и контроль нового пароля не будут совпадать, а также длина нового пароля будет не менее, чем MinLength. Возвращает true в случае успешной смены пароля. |
|  | function \_AlterPassDlgLen(var NewPass: string; MinLength: integer = -1; ParentWND: HWND = 0 Caption: string = '' X: integer = POS\_CENTER; Y: integer = POS\_CENTER): boolean | Позволяет вызвать диалог смены пароля, сама функция пароль не изменяет, но таким образом можно реализовать свой алгоритм работы с диалогом смены пароля. В параметре NewPass будет возвращен новый пароль, получившийся в результате. Возвращает true в случае успешной смены пароля. |
|  | function CreateUser(const User: string Pass: string = '' ; Profile: string = 'HTPROFILE' ; Startup: string = ''): THtError | Позволяет создать пользователя БД с именем пользователя User, паролем Pass, профилем Profile. В параметре Startup можно указать имя стартового файла для данного пользователя (по умолчанию у всех пользователей этот файл называется  strtuser.sql) Возвращает ошибку в случае неудачи. |
|  | procedure Connect | Позволяет установить соединение с БД. |
|  | procedure ConnectAs(Path, User, Pass: string) | Позволяет установить соединение с указанной БД (располагающейся по адресу Path) под указанным пользователем (User) с паролем(Pass). |
|  | function CallSql(Caller: IHtBaseNotifier; Sql: PChar; SqlToOEM: boolean = true; Msgs: TStrings = nil): Variant; overload | Позволяет выполнить код, указанный в параметре Sql, при этом имеется возможность указать, требуется ли преобразование текста SQL из кодировки Windows в кодировку OEM,  возвращает RetCode, в случае ошибки вызывает исключение. Все сообщения, присланные по CallBack, наодятся в Msgs, их обрабатывает Caller. |
|  | function CallSql(Sql: PChar; SqlToOEM: boolean = true; Msgs: TStrings = nil): Variant; overload | Позволяет выполнить SQL код, возвращает RetCode, в случае ошибки вызывает исключение. Все сообщения, присланные по CallBack, хранятся в Msgs, их обрабатывает сам компонент THtBase, использование остальных параметров аналогично предыдущему методу. |
|  | function \_CallSql(Caller: IHtBaseNotifier; Sql: PChar; SqlToOEM: boolean = true; RetCode: PVariant = nil; Msgs: TStrings = nil): THtError; overload | Позволяет выполнить SQL код, в случае ошибки просто возвращает код ошибки, не вызывая исключения, в Msgs накапливаются все сообщения. |
|  | function CheckNull(RecBuf: PChar; Def: THtFieldDef): boolean | Позволяет проверить на null содержимое поля Def в буфере записи RecBuf. Возвращает значение true, если содержимое поля null. |
|  | procedure ComplexEval(HtCurs: IHtCursor; SqlToOEM, CalcOnly: Boolean; const Statement: PChar; InfoHandler: TInfoHandler; Param: PComplexParamRec) | Процедура транслируется в API-функцию, которая создает курсор, осуществляет разбор SQL кода, выполняет код, читает результат, закрывает курсор, если передан существующий курсор, то выполняться будет в нем, если передается nil вместо курсора, то курсор будет создан. |
|  | procedure CursorOrder(HtCurs: IHtCursor; Flds: TAFieldNo) | Позволяет пересортировать курсор HtCurs согласно списку номеров полей Flds , сначала сортировка по первому полю, потом по второму итд, если номер отрицателен, сортировка в обратном порядке. |
|  | procedure DeleteRec(HtCurs: IHtCursor; Row: LongInt) | Позволяет удалить запись Row из курсора HtCurs, не вызывая реакции триггеров. Использовать с осторожностью. |
|  | procedure DeleteResult(HtCurs: IHtCursor; Row: LongInt) | Позволяет удалить строку Row из результата HtCurs (но не из таблицы). |
|  | procedure Disconnect | Позволяет разорвать соединение с БД. |
|  | procedure DoParserError(Sender: TCustomSqlParser; const E: EHtSyntaxError; var Handled: boolean) | Позволяет проверить, назначено ли событие на ошибку E парсера Sender. Параметр Handled позвляет указать, обработана ли ошибка. |
|  | procedure EnumClients(Proc: TEnumNotifiersProc; Param: integer) | Позволяет перечислить все текущие клиенты THtBase (компоненты, которые  получают от THtBase уведомления), вызвать для каждого из них процедуру  Proc и при вызовах подставлять в нее параметр Param. |
|  | function EvalReadClose(const Statement: PChar; Buff: Pointer; var Bytes: integer; Msgs: TStrings = nil): THtError; | Позволяет выполнить запрос Statement, вернуть результат в буфер Buff и сохранить полученные сообщения в Msgs. Bytes показывает, сколько байт результата считать. Возвращает ошибку в случае неудачи. |
|  | function FindRecordInCursor(HtCurs: IHtCursor; FieldNo: TFieldNo; Data, Dst: Pointer; DataLen: integer): integer | Позволяет найти в курсоре HtCurs нужную запись, сравнивая значения в поле номер FieldNo и значение размером DataLen из буфера Data. Если запись найдена, то она помещается в буфер Dst. Функция возвращает 0 в случае успеха или код ошибки в случае неудачи. На данный момент не рекомендуется к использованию вследствие большого времени выполнения. |
|  | procedure FreeCursor(HtCurs: IHtCursor) | Позволяет освободить SQL курсор HtCurs. |
|  | procedure GetHoldRow(HtCurs: IHtCursor const TableAlias: string; RowNo: integer; Hold: THSLocks RR, Rec: Pointer) | Позволяет выполнить захват записи в строке RowNo курсора HtCurs по таблице TableAlias (если в Hold входит hslDoLock), прочесть строку результата в буфер Rec и элемент результата в буфер RR. |
|  | function GetMacro(var S: string): Boolean | Позволяет получить тело макроса по его названию S. В случае, если тело макроса найдено не было, возвращает false. |
|  | procedure GetResult(HtCurs: IHtCursor; Row: LongInt; RR: PResultRec) | Позволяет получить элемент номер Row результата HtCurs в RR. |
|  | function GetTablesInfo(const TableNames: string; InfoProc: Pointer): THtError | Позволяет получить описание указанных в TableNames таблиц и вызвать указанный CallBack –обработчик InfoProc (указатель на внутреннюю процедуру) для каждого описания. Возвращает ошибку в случае неудачи. |
|  | function HandleFromRecNo(HtCurs: IHtCursor; const Tbl: string RecNo: Integer): TRHANDLE | Позволяет конвертировать RecNo (номер записи в таблице) из курсора HtCurs, таблицы Tbl в Handle – номер элемента результата (его и возвращает). |
|  | function HtCheck(rc: THtError): Boolean | Позволяет проверить код ошибки rc и вызвать исключение с соответствующим сообщением. Возвращает true в случае отсутствия ошибки. |
|  | procedure InsertRec(HtCurs: IHtCursor; Row: LongInt ; Buf: Pointer) | Позволяет добавить запись из буфера Buf в курсор HtCurs под номером Row. Использовать с осторожностью. |
|  | procedure InsertResult(HtCurs: IHtCursor Row: LongInt RR: PResultRec) | Позволяет добавить строку RR в результат IHtCursor , так, чтобы номер строки был Row. |
|  | function InsertResults(HtCurs: IHtCursor const TableName: string Count: LongInt RR: PInteger): IHtCursor | Позволяет добав ить в таблицу TableName, открытую курсором HtCurs , несколько (количество записей =- Count) записей (RR), не вызывая реакции триггеров. В случае, если HtCurs не nil, возвращает HtCurs, в противном случае создает новый курсор и возвращает ссылку на него. |
|  | function Login(aUserName, aPass: string ShowDialog: Boolean = false): THtError | Позволяет выполнить авторизацию пользователя aUserName с паролем aPass. Параметр ShowDialog показывает, выводить ли на экран диалог подключения. Возвращает ошибку в случае неуспеха. |
|  | function RecNoFromHandle(const Tbl: string Rh: TRHANDLE): integer | Позволяет получить номер записи RecNo (в качестве возвращаемого значения) по номеру элемента результата Rh таблицы Tbl. |
|  | procedure RefreshRecord(HtCurs: IHtCursor RowNo: integer Buf: Pointer) | Позволяет для записи номер RowNo из курсора HtCurs получить новый результат (актуальный на текущий момент) и записать его в буфер Buf. |
|  | procedure RegisterNotifier(Notifier: IHtBaseNotifier; AllEvents: boolean = false) | Позволяет зарегистировать клиента Notifier, параметр AllEvents указывает, будет ли новый клиент получать все сообщения. |
|  | procedure ReleaseRow(HtCurs: IHtCursor const TableAlias: string RowNo: integer AllJoin: boolean) | Позволяет освободить запись номер RowNo в таблице, алиас которой указан в TableAlias, курсора HtCurs, вызвать исключение в случае ошибки. |
|  | function \_ReleaseRow(HtCurs: IHtCursor const TableAlias: string RowNo: integer AllJoin: boolean): THtError | Позволяет освободить запись номер RowNo из курсора HtCurs по таблице TableAlias, в зависимости от значения AllJoin освобождаются все таблицы в join или нет. Возвращает ошибку в случае неуспеха. |
|  | function RetCode: Variant | Позволяет получить последнее значение retCode() . |
|  | function SeekRead(HtCurs: IHtCursor; Num: Integer; Dst: Pointer; Bytes: word): integer | Позволяет переместиться в курсоре HtCurs на указанную запись Num и прочитать, начиная с неё, нужное количество байтов Bytes в Dst. Возвращает количество прочитанных байт. |
|  | procedure Terminate override | Позволяет завершить поток. |
|  | procedure UnRegisterNotifier(Notifier: IHtBaseNotifier) | Позволяет отменить регистрацию клиента Notifier. |
|  | procedure UpdateRec(HtCurs: IHtCursor; Row: LongInt; Buf: Pointer; LockUnlock: boolean) | Позволяет модифицировать курсор и таблицу одновременно, при этом не срабатывают триггеры. Использовать с осторожностью. |
|  | procedure UpdateResult(HtCurs: IHtCursor Row: LongInt RR: PResultRec) | Позволяет заменить элемент результата записи номер RowNo из курсора HtCurs, указатель на новый элемент результата находится в RR. |

# **THtQuery**

Базовым классом – родителем для THtQuery является класс TDataSet. Иерархия классов приведена на рисунке .



Рис. Иерархия классов

Рассмотрим некоторые свойства компонента.

Текст запроса, который необходимо выполнить в базе данных, необходимо поместить в атрибут SQL. В случае если в запросе участвуют передаваемые параметры, их необходимо указать в Params. Название базы данных указывается в атрибуте BaseName. Необходимо обратить внимание на то, что типом атрибута является не строка или THtBase, рассмотренный в предыдущем разделе, а THtBaseName.

Если при проектировании связь с базой данных уже установлена, то есть имеется активное подключение к БД, то свойству Active можно присвоить значение true, в этом случае запрос выполнится в базе данных и в компоненте, предназначенном для отображения результатов выборки данных, связанном с THtQuery, будет видна информация, полученная из БД. Если при изменении свойства Active активное подключение к базе отсутствует, возникнет ошибка, пример приведен на .

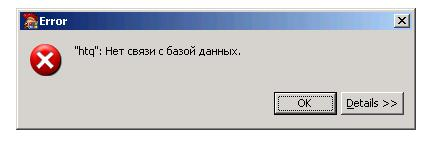


Рис. Ошибка связи с БД

Для того чтобы избежать появлений этой ошибки, достаточно вместо свойства Active изменить свойство AutoActive. В этом случае запрос выполнится тогда, когда будет подключена база данных (в режиме проектирования или в ходе работы приложения).

Если после обращения к таблицам, их необходимо закрыть, то свойству CloseTables нужно присвоить true.

В случае если оставить свойству значение по умолчанию, таблицы будут закрыты после присвоения свойству Active значения false или после закрытия связи с БД через компонент HtBase.

В случае если в запросе присутствуют атрибуты типа char, имеющие большую размерность (больше, чем 2 Кб), значение свойства BigStrAsMemo будет влиять на тип поля(Field), создающегося при запросе к БД.

По умолчанию свойству BigStrAsMemo присвоено значение true, это значит, что поля для атрибутов char большой размерности будут создаваться с типом TMemoField, если свойству присвоено значение false, вне зависимости от размерности атрибута с типом char, для него будет создано поле типа TStringField.

В случае если в запросе присутствуют атрибуты типа currency, значение типа CurrencyAsFixed будет влиять на тип поля, создающегося для этого атрибута при запросе к БД. По умолчанию свойству CurrencyAsFixed присвоено значение false, это значит, что для атрибута типа currency будет создано поле типа TCurrencyField, если CurrencyAsFixed установить в значение true, то для атрибута будет создано поле типа THtFixPointField.

Ещё одним свойством, влияющим на тип поля, является свойство DwordAsLarge. По умолчанию false, в этом случае для любого атрибута типа dword создается поле TIntegerField, в противном случае – TLargeintField.

Все свойства компонента представлены в .

Таб. Свойства компонента THtQuery

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | false | published |
|  | Aliases | - | public |
|  | AnsiTable | false | published |
|  | AutoActive | false | published |
|  | BaseName | - | published |
|  | BigStrAsMemo | true | published |
|  | BufSize | 65535 | published |
|  | CalcFieldsOnly | false | published |
|  | CalcNames | - | published |
|  | CanModify | - | public |
|  | CloseTables | false | published |
|  | CurrencyAsFixed | false | published |
|  | Cursor | - | public |
|  | DataBase | - | public |
|  | DataSource | - | published |
|  | DesignOnlyFields | true | published |
|  | DllLookUp | false | published |
|  | DoLockRecord | true | published |
|  | DwordAsLarge | false | published |
|  | HoldAllJoin | - | public |
|  | IsHold | - | public |
|  | LockMode | [] | published |
|  | MacroList | - | published |
|  | MainAlias | - | published |
|  | MainAliasIndex | - | public |
|  | MainRecnoName | - | public |
|  | MainTable | - | public |
|  | MainTableAlias | - | public |
|  | Marker | - | public |
|  | Markers | - | public |
|  | Name | - | published |
|  | NoParse | false | published |
|  | ParamCount | - | public |
|  | Params | - | published |
|  | ParamsInMacro | false | published |
|  | Parser | - | published |
|  | RaiseMode | rmSmart | published |
|  | ReadOnly | false | published |
|  | RecPerBuf | 65520 байт | public |
|  | RefreshDelay | 250 | published |
|  | RetCode | - | public |
|  | Script | - | public |
|  | SQL | - | published |
|  | Tag | 0 | published |
|  | UpdateObject | - | published |
|  | Working | - | public |

Остановимся более подробно на каждом свойстве.

Active: свойству присваивается «истина» в момент активизации компонента.

Aliases: содержит список алиасов таблиц. После выполнения запроса Aliases содержит список пар алиас=таблица. Если алиас на таблицу не назначен, считается, что алиас совпадает с именем

AnsiTable: свойство необходимо для реализации возможности работы с таблицам, кодировка которых ANSI.

AutoActive: свойство показывает, надо ли активизировать компонент при активизации соответствующего DataBase.

BaseName: свойство содержит ссылку на БД.

CanModify: Определяет реальную возможность выполнять операции редактирования записей. THtDataSet допускает редактирование записей, если выполнены три условия:

1. ReadOnly = false;
2. результат сортирован или запрос был for update;
3. результат получен по одной таблице или назначены MainAlias и UpdateObject.

Таким образом, если все три условия соблюдены, значение CanModify – true, в противном случае – false.

BufSize: свойство контролирует размер буфера для КЭШа записей, цифра выбрана исходя из размера ответа сервера.

CalcFieldsOnly: свойство показывает, необходимо ли дополнительно запрашивать у сервера служебную информацию об именах полей. Использовалось для предыдущих версий СУБД. В настоящий момент устарело.

CalcNames: свойство хранит список вычисляемых полей. В настоящий момент устарело.

CanModify: свойство содержит информацию о реальной возможности выполнять операции редактирования записей. Свойство ReadOnly выставляется пользователем, CanModify определяет, является ли результат несортированным или является ли результат объединением нескольких выборок ( в этом случае редактировать нельзя CanModify = false).

CloseTables: свойство показывает, надо ли при закрытии компонента закрывать таблицы, с которым работал компонент.

CurrencyAsFixed: см. выше.

Cursor: свойство, предназначенное для хранения основной информации о SQL курсоре.

DataBase: содержит ссылку на THtDataBase, с которым работает компонент. Можно проставить вручную.

DataSource: аналогично свойству TDataSet.

DesignOnlyFields: показывает, надо ли создавать автоматически недостающие поля ( кроме созданных вручную) в момент активизации, если таковые будут присутствовать в запросе. Аналогом свойства в стандартном компоненте TDataSet является свойство DefaultFields.

DllLookUp: свойство показывает, есть ли необходимость использовать специальную библиотеку, которая была написана для реализации дихотомического поиска данных в сортированном результате по полю сортировки.

DoLockRecord: свойство показывает, необходимо ли захватывать запись при переходе в режим редактирования. Есть два подхода к захвату записей при редактировании данных – пессимистический режим (DoLockRecord = true) –запись захватывается в момент начала редактирования, иначе (оптимистический подход) запись захватывается при отправке данных в БД (post).

DwordAsLarge: см. выше.

HoldAllJoin: свойство показывает, захвачены ли записи из всех использованных в запросе таблиц.

IsHold: свойство показывает, захвачена ли текущая запись.

LockMode: свойство содержит информацию о способе захвата записи. Работа ведется с тремя флагами – lmAllJoin, lmCanTwise , lmDeleting.

* + - * + lmAllJoin – захватывать ли все таблицы, указанные в запросе, если нет, то захват осуществляется только той таблицы, алиас которой указан в свойстве MainAlias;
        + lmCanTwise – если флаг установлен в значение «истина», то попытка повторного захвата записи не приведет к ошибке;
        + lmDeleting – захватывать ли запись при удалении, это влияет на возникновение ошибки при удалении несуществующей записи;

MacroList: содержит ссылку на хранилище макросов.

MainAlias: содержит алиас таблицы, запись которой захватывается в начале редактирования (если DoLockRecord = true).

MainAliasIndex: содержит индекс MainAlias в списке Aliases.

MainRecnoName: содержит имя поля recno для таблицы MainAlias, то есть "название таблицы.recno".

MainTable: содержит имя таблицы, чей алиас есть MainAlias.

MainTableAlias: содержит MainAlias, если он не пустой, иначе возвращает имя таблицы( если результат по одной таблице).

Marker: содержит ссылку на текущий маркер с состоянием.

Markers[RowNo: integer]: содержит информацию о том, отмечена ли указанная строка.

Name: содержит название компонента.

NoParse: свойство показывает, необходимо ли осуществлять разбор sql кода (использовать ли препроцессор).

ParamCount: содержит количество параметров в запросе.

Params: содержит список параметров запроса, как и в стандартном TQuery, параметры начинаются символом двоеточия.

ParamsInMacro: указывает, есть ли необходимость искать параметры (Params) в макросах. Когда препроцессор обрабатывает sql-код с целью извлечения списка параметров, ParamsInMacro = true заставляет его также получать и просматривать тела макросов.

Parser: содержит ссылку на внешний препроцессор SQL.Если Parser не назначен, создаётся временный экземпляр TSqlParser'а для обработки запроса, иначе используется Parser. Внешний препроцессор позволяет более широко управлять процессом, т.к. доступны все его события и методы.

RaiseMode: содержит Вид реакции компонента на ошибки, возникающие при редактировании.

* + - * + rmSmart – если RetCode имеется в списке #errmsg, то отображается соответствующее сообщение и выполняется Abort, иначе исключительная ситуация распостраняется далее;
        + rmAlwais – если RetCode имеется в списке #errmsg, отображается соответствующее сообщение, иначе исключительная ситуация распостраняется далее;
        + rmNone – если RetCode имеется в списке #errmsg, отображается соответствующее сообщение, вместо распространения исключения выполняется Abort. #errmsg можно вписать в свойство StartSQL соотствующего компонента THtBase

ReadOnly: содержит флаг запрета редактирования записей.

RecPerBuf: содержит размер буфера начитываемых записей в байтах. По умолчанию равен 65520 байт.

RefreshDelay: содержит величину задержки (мс) между изменением данных в Parent DataSet и Child DataSet.

RetCode: содержит значение retcode(), установленное последним выполненным запросом. Если retcode() в последнем запросе не выполнялся, возвращается нуль.

Script: содержит запрос, оставлено для обратной совместимости компонента. У наследников компонента расположено в секции public.

SQL: содержит запрос, выполняемый методами Open, Requery и при установке свойства Active := true. Содержит тот же текст, что и свойство Script. Имя изменено для удобства работы в Delphi: когда текст запроса открывается в редакторе, он получает "расширение" SQL и редактор включает соответствующую подсветку синтаксиса.

UpdateObject: в свойство можно записать ссылку на компонент типа THtUpdateSQL.

Working: содержит флаг, который принимает значение «истина» в процессе выполнения скрипта.

Рассмотрим события компонента. Будем останавливаться подробно на событиях, специфичных для HyTech.

1. AfterCancel (TDataSetNotifyEvent). Вызывается после отмены операции (вставки, редактирования или удаления).
2. AfterCancelEx (TOnAfterPostEx). Вызывается после отмены операции, здесь можно узнать, какую именно операцию отменили. От события AfterCancel отличается наличием дополнительного параметра: состояние DataSet перед операцией Cancel. Принимает два параметра: Sender (THtDataSet), UpdateState (TDataSetState), показывает Sender.State перед операцией Post (dsEdit или dsInsert).
3. AfterClose (TDataSetNotifyEvent). Вызывается после закрытия компонента.
4. AfterDelete (TDataSetNotifyEvent). Вызывается после операции удаления.
5. AfterEdit (TDataSetNotifyEvent). Вызывается после выполнения операции редактирования.
6. AfterInsert (TDataSetNotifyEvent). Вызывается после выполнения операции вставки.
7. AfterOpen (TDataSetNotifyEvent). Вызывается после выполнения операции открытия.
8. AfterPost (TDataSetNotifyEvent). Вызывается после выполнения операции post.
9. AfterPostEx (TOnAfterPostEx). Вызывается после выполнения операции post. От события AfterPost отличается наличием дополнительного параметра: состояние DataSet перед операцией Post.
10. AfterRunSql (TDataSetNotifyEvent). Вызывается после выполнения скрипта. Если выполнить метод Reopen, то при повторном вызове sql кода возникнет событие AfterRunSql( и не сработает событие AfterOpen).
11. AfterScroll (TDataSetNotifyEvent). Вызывается после перехода на следующую запись.
12. BeforeCancel (TDataSetNotifyEvent). Вызывается до отмены операции (вставки, удаления или редактирования).
13. BeforeClose (TDataSetNotifyEvent). Вызывается до закрытия компонента.
14. BeforeDelete (TDataSetNotifyEvent). Вызывается перед выполнением операции удаления.
15. BeforeEdit (TDataSetNotifyEvent). Вызывается перед выполнением операции редактирования.
16. BeforeInsert (TDataSetNotifyEvent). Вызывается перед выполнением операции вставки.
17. BeforeOpen (TDataSetNotifyEvent). Вызывается перед выполнением открытия компонента.
18. BeforeParse (TNotifyEvent). Вызывается перед выполнением разбора sql кода.
19. BeforePost (TDataSetNotifyEvent). Вызывается перед выполнением операции post.
20. BeforeRunSql (TDataSetNotifyEvent). Вызывается перед выполнением sql кода.
21. BeforeScroll (TDataSetNotifyEvent). Вызывается перед выполнением перемещения на следующую запись.
22. OnCalcFields (TDataSetNotifyEvent). Вызывается при расчете вычисляемых полей.
23. OnCallBack (TOnCallBack). Вызывается при обработке сообщений, полученных от сервера. Принимает ряд параметров:
    * + - * Sender (THtDataSet);
          * Oper (TCallBackOper) – содержит тип CallBack операции, возможные значения: coUnknown (неизвестный тип, при отсутствии ошибок появляться не должен), coUserSend (требуется обработка sql функции "usersend"), coOutMess (требуется обработка sql функции "?"), coIdle (во время длительного ожидания HyTech API может вызываться время от времени событие этого типа), coErrMsg(требуется обработка сообщения об ошибке), coLocErr (требуется обработка сообщения об ошибке клиентской части), coAdminMsg (требуется обработка административного сообщения);
          * Code (smallInt) - содержит числовой параметр, имеет смысл, если Oper = coUserSend или Oper = coAdminMsg;
          * Msg (string) – содержит строку сообщения (кодировка OEM), определен для всех типов, кроме coAdminMsg и coIdle.
24. OnDeleteError (TDataSetErrorEvent). Вызывается при возникновении ошибки удаления.
25. OnDbStateChanged (TNotifyEvent). Вызывается при изменении состояния базы данных.
26. OnEditError (TDataSetErrorEvent). Вызывается при возникновении ошибки редактирования.
27. OnElementUpdated (TOnElementUpdated) of object). Вызывается при обновлении элемента результата. Принимает два параметра: Sender (THtDataSet) и UpdateKind (TUpdateKind). Событие возникает после редактирования результата (методы DeleteElement, InsertElement, ReplaceElement). В параметре UpdateKind передается вид редактирования результата.
28. OnFieldsCreated (TNotifyEvent). Вызывается при создании полей.
29. OnFieldTypeConv (TOnFieldTypeConv). Вызывается при преобразовании типов полей (из поля HyTech в поле Delphi). Назначив обработчик на это событие можно изменить ппреобразование по-умолчанию. Принимает ряд параметров:
    * + - * Sender (THtDataSet);
          * Name (PChar) – содержит название поля;
          * aHtType (integer ) – содержит тип поля HyTech;
          * Order (integer) – содержит порядковый номер поля в результате;
          * FldType (TFieldType) – содержит тип поля Delphi.
30. OnGetMacro (TOnGetMacro). Вызывается при получении тела макроса. Принимает ряд параметров:
    * + - * Sender( TObject)
          * S(string) - содержит на входе имя макроса (без лидирующих сиволов #);
          * Found(boolean) -если замена произведена, параметр должен будет установлен в true, иначе - false.
31. OnIdle (TIdleEvent) – см. описание в компоненте THtBase.
32. OnNewRecord (TDataSetNotifyEvent). Вызывается при операции добавления записи.
33. OnParserError (TOnParserError). Вызывается при возникновении ошибки в работе препроцессора SQL.
34. OnPostError (TDataSetErrorEvent). Вызывается при возникновении ошибки операции post.
35. OnRefreshDelay (TNotifyEvent). Вызывается при обновлении компонента как дочернего DataSet информацией из родительского.
36. OnRequiredError (TOnRequiredError). Вызывается при возникновении ошибки при работе с обязательно заполняемыми атрибутами. Принимает ряд параметров:
    * + - * Sender(TObject);
          * Field (TField) – содержит поле;
          * Next (boolean) – устанавливается в true, если принято решение показать пользователю все незаполненые, но обязательные к заполнению поля, если остальные поля, кроме Field, проверять не надо, устанавливается в false.
37. OnSqlPrepared (TOnSqlPrepared). Вызывается после разбора sql кода препроцессором и перед отправкой его серверу. Использует ряд параметров:
    * + - * Sender (THtDataSet);
          * Sql (string) – содержит текст sql запроса;
          * DoAbort (boolean) – показывает, отменено ли действие.
38. OnUserSend (TOnUserSend). Вызывается при обработке сообщений сервера, полученных по каналу usersend. Использует ряд параметров:
    * + - * Sender (THtDataSet);
          * Code (smallint) – код пакета usersend;
          * Msg (string) – текст сообщения usersend;
          * Handled (boolean) – если параметр будет установлен в true, событие не будет обрабатываться соотствующим THtBase.

Компоненты THtQuery и THtStoredProc являются наследниками одного и того же класса – THtDataSet. Основные методы класса THtDataSet представлены в .

Таб. Основные методы класса THtDataSet

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | function BlockRead(Num: LongInt; Dst: Pointer; Bytes: Word): Word; | Позволяет прочитать несколько записей одним массивом, начиная с определенной строки Num в указанный буфер Dst. Объем строки указывается в параметре Bytes.. Читается всегда целое количество строк результата. Функция возвращает количество прочитанных байтов. |
|  | function DoGetMacro(var S: string): boolean; | Позволяет получить тело макроса по его имени S. Возвращает true, если тело макроса найдено, false, если не найдено или макрос-лист не назначен. |
|  | function GetAliasRecNo(const Alias: string): integer; | Позволяет получить для таблицы, алиас которой указан в Alias RecNo текущей строки ( используется, если в выборке объелинены несколько строк). |
|  | function GetCurrentRecord(Buffer: PChar): Boolean; override; | Позволяет прочитать тело текущей записи в буфер Buffer. Возвращает true, если тело было прочитано. |
|  | procedure InsCopy; | Позволяет добавить копию текущей записи, после операции компонент остается в режиме добавления. |
|  | function IsSequenced: Boolean; override; | Всегда возвращает значение true. |
|  | function CompareBookmarks(Bookmark1, Bookmark2: TBookmark): Integer; override; | Позволяет сравнить позиции закладок Bookmark1 и Bookmark2, возвращает разность между позициями. |
|  | function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override; | Позволяет создавать поток для работы с данными BLOB поля TField, параметр Mode предоставляет возможность задать режим использования потока (см. стандартную документацию к Delphi). |
|  | procedure DeleteElement; | Позволяет удалить элемент результата. |
|  | function FieldTypeByHtType(const Name: PChar; aHtType, Order, bytes: integer): TFieldType; virtual; | Позволяет получить TFieldType (тип поля Delphi) поля Name по THtType, указанному в aHtType, размер поля указан в bytes, параметр Order используется, если в компоненте назначен обработчик событий FOnFieldTypeConv. |
|  | function GetBlobFieldData(FieldNo: Integer; var Buffer: TBlobByteData): Integer; override; | Позволяет получить информацию поля номер FieldNo типа BLOB в буфер Buffer. Возвращает размерность поля в байтах. |
|  | function Lock(AllJoin: boolean = false): TLockError; | Позволяет заблокировать записи, параметр AllJoin указывает, надо ли блокировать записи во всех таблицах, к которым обращается sql код. Возвращает одно из значений: errNoError, errNoForUpdate, errRecordLocked, errErrLockRecord, errErrLockTwise, errBadLockParam,errLockInactive. |
|  | function Translate(Src, Dest: PChar; ToOem: Boolean): integer; override; | Позволяет перевести текст запроса из кодировки OEM в ANSI и наоборот, источник текста приводитсяв Src, текст записываеся в Dest, если ToOem = true, то перевод осуществляется в направлении ANSI->OEM, иначе наоборот. Возвращает размерность строки Src. |
|  | function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions): Boolean; override; | В текущей реализации компонентов функция не работает. |
|  | function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields: string): Variant; override; | В текущей реализации компонентов функция не работает. |
|  | function MakeMarkedCursor(const Alias: string; CurrIfNoMarks: boolean): IHtCursor; | Позволяет отметить записи в исходном THtDataSet и получить информацию из таблицы, алиас которой = Alias, в виде курсора как результат селекта. CurrIfNoMarks показывает, надо ли строить курсор, если нет отмеченных записей (если CurrIfNoMarks = true, то надо). |
|  | function MarkedToDS(const Alias: string; DS: THtDataSet; CurrIfNoMarks: boolean): IHtCursor; | Позволяет на основе отмеченных в исходном THtDataSet записей таблицы, алиас которой = Alias, сделать курсор, использовать номер курсора в наборе данных DS, выполнить запрос компонента DS и вернуть результат запроса в виде курсора. Параметр CurrIfNoMarks описан выше. |
|  | procedure MarkedToDSP(const Alias: string; DS: THtDataSet; CurrIfNoMarks: boolean); | Позволяет на основе отмеченных в исходном THtDataSet записей таблицы, алиас которой = Alias, сделать курсор, использовать номер курсора в наборе данных DS, выполнить запрос компонента DS , получить информацию как результат запроса компонента DS. Параметр CurrIfNoMarks описан выше. |
|  | function ArrayToCursNo(RR: PInteger; Count: dword; const TableName: string): integer; | Позволяет создать курсор для таблицы TableName по списку RecNo количеством Count, который хранится в буфере, на который указывает RR, после чего подставить номер получившегося курсора в параметр :cursno и затем активировать (открыть) THtDataSet. Возвращает RetCode после активации набора данных. |
|  | function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; | Позволяет получить информацию в буфер Buffer по полю Field. Возвращает true, если Buffer не nil . |
|  | function GetMacroParamList(var fst: integer; var List: TStrings): boolean; | Позволяет получить список параметров (записывается в List) из скрипта. Возвращает true, если список параметров макроса вернуть удалось. |
|  | procedure InsertElement(DS: THtDataSet); | Позволяет прочитать элемент результата из текущей строки DS и вставить его в свой результат в текущую позицию. |
|  | procedure ReplaceElement(DS: THtDataSet); | Позволяет прочитать элемент результата из текущей строки DS и заменить им текущий элемент своего результата. |
|  | function DefaultGetMacro(var S: string): boolean; | Позволяет получить тело макроса по его имени S. Если назначен MacrosList, пытается получить тело макроса у него, иначе (или если MacrosList вернул false), вызывает DataBase.GetMacro. Возвращает true, если тело макроса получить удалось. |
|  | function DefByField(Fld: TField; out Def: THtFieldDef): boolean; | Позволяет получить описание поля Fld , описание сохраняется в параметре Def. Возвращает true, если поле найдено. |
|  | function ParamByName(const Value: string): TParam; | Позволяет вернуть ссылку на TParam с именем Value. Если такой параметр не найден, вызывается исключение. |
|  | function RefreshRecord: boolean; dynamic; | Позволяет обновить буфер текущей записи и элемент результата курсора актуальными на момент выполнения данными. Возвращает true, если буфер обновить удалось. |
|  | function RowByValue(const Fld, Value: string): integer; dynamic; | Позволяет осуществить поиск строки в результате по значению Value поля Fld, работает по отсортированному списку. Функция возвращает номер строки, в случае, если строка не найдена, будет возвращено значение «-1». |
|  | procedure Resort(Flds: TAFieldNo); | Позволяет пересортировать курсор по заданным в параметре Flds полям , работает по TField.FieldNo, если номер отрицательный, сортировка по этому полю идёт по убывающей. |
|  | procedure Reorder(Flds: array of string); | Позволяет пересортировать курсор по заданным полям Flds, работает по TField.FieldName. Если первая буква в названии поля – это знак ‘-’, сортировка по этому полю идёт по убывающей. |
|  | procedure Reopen; | Позволяет закрыть и сразу же открыть компонент. |
|  | procedure Requery; | Позволяет выбрать ещё раз без закрытия. |
|  | function SetLock(Value, AllJoin: boolean; OwnedLock: boolean = true): TLockError; | Позволяет установить / снять блокировку. Параметр Value показывает, надо ли установить блокировку (Value = true) или снять её. AllJoin показывает, надо ли блокировать все таблицы, к которым обращается компонент. Параметр OwnedLock показывает, кто будет владельцем захвата - сам THtDataSet (true) или нет. Возвращает одно из значений : errNoError, errNoForUpdate, errRecordLocked, errErrLockRecord, errErrLockTwise, errBadLockParam,errLockInactive. |
|  | function UnLock: TLockError; | Позволяет снять блокировку. Возвращает одно из значений: errNoError, errNoForUpdate, errRecordLocked, errErrLockRecord, errErrLockTwise, errBadLockParam,errLockInactive. |
|  | function UserSendHandle(code: smallint; Buf: string): boolean; virtual; | Позволяет имитировать получение usersend-сообщения от сервера с кодом пакета code и сообщением Buf, после чего вызвать его обработку. Возвращает true, если сообщение обработано. |

Основные методы класса TCustomHtQuery представлены в .

Таб. Основные методы класса TCustomHtQuery

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | function RowByValue(const Fld, Value: string): integer | Позволяет найти строку по значению Value поля Fld. Поиск выполняется сначала во внутреннем буфере, и только в случае неуспеха вызывается метод предка. Функция возвращает номер строки, в случае, если строка не найдена, будет возвращено значение «-1». |

Если THtQuery в неактивном состоянии имеет какие-либо поля (TField), то никакие больше поля при активации не создадутся. Если изменить значение свойства DesignOnlyFields  на false, то после выполнения запроса у DataSet будут созданы TField для всех отобранных столбцов.

В свойстве UpdateObject есть возможность указать ссылку на компонент типа THTUpdateSql, который используется для реализации возможности применения операций DML к набору данных, который возвращает THtQuery в случае, если набор данных в силу различных обстоятельств не предназначен для изменения(ReadOnly = true). Если в запросе перечисляется несколько таблиц и для каждой приведен алиас, то алиас главной таблицы можно указать в свойстве MainAlias.

# **THtStoredProc**

Компонент предназначен для выполнения произвольных запросов, не требующих чтения более одной строки результата. Иерархия классов приведена на



Рис. Иерархия классов

Свойства компонента представлены в .

Таб. Свойства компонента THtStoredProc

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | false | published |
|  | AnsiTable | false | published |
|  | AutoActive | false | published |
|  | BaseName | - | published |
|  | BigStrAsMemo | true | published |
|  | CalcFieldsOnly | false | published |
|  | CalcNames | - | published |
|  | CloseTables | false | published |
|  | CurrencyAsFixed | false | published |
|  | DataSource | - | published |
|  | DesignOnlyFields | true | published |
|  | DoLockRecord | true | published |
|  | DwordAsLarge | false | published |
|  | LockMode | [] | published |
|  | MacroList | - | published |
|  | MainAlias | - | published |
|  | Name | - | published |
|  | NoParse | false | published |
|  | Params | - | published |
|  | ParamsInMacro | false | published |
|  | Parser | - | published |
|  | ReadOnly | false | published |
|  | RaiseMode | rmSmart | published |
|  | RefreshDelay | 250 | published |
|  | SQL | - | published |
|  | Tag | - | published |
|  | UpdateObject | - | published |

Описание одноименных свойств приведено в разделе .

Рассмотрим события компонента. Подробное описание событий, специфичных для HyTech, приведено в разделе, посвященном компоненту THtQuery.

1. AfterCancel (TDataSetNotifyEvent). Вызывается после отмены операции (вставки, редактирования или удаления).
2. AfterClose (TDataSetNotifyEvent). Вызывается после закрытия компонента.
3. AfterDelete (TDataSetNotifyEvent). Вызывается после операции удаления.
4. AfterEdit (TDataSetNotifyEvent). Вызывается после выполнения операции редактирования.
5. AfterInsert (TDataSetNotifyEvent) Вызывается после выполнения операции вставки.
6. AfterOpen (TDataSetNotifyEvent). Вызывается после выполнения операции открытия.
7. AfterPost (TDataSetNotifyEvent). Вызывается после выполнения операции post.
8. AfterRunSql (TDataSetNotifyEvent). Вызывается после выполнения скрипта. Если выполнить метод Reopen, то при повторном вызове sql кода возникнет событие AfterRunSql (и не сработает событие AfterOpen).
9. AfterScroll (TDataSetNotifyEvent). Вызывается после перехода на следующую запись.
10. BeforeCancel (TDataSetNotifyEvent). Вызывается перед отменой операции (вставки, удаления или редактирования).
11. BeforeClose (TDataSetNotifyEvent). Вызывается перед выполнением операции удаления.
12. BeforeDelete (TDataSetNotifyEvent). Вызывается перед выполнением операции редактирования.
13. BeforeEdit (TDataSetNotifyEvent). Вызывается перед выполнением операции вставки.
14. BeforeInsert (TDataSetNotifyEvent). Вызывается перед выполнением открытия компонента.
15. BeforeOpen (TDataSetNotifyEvent).Вызывается перед выполнением операции удаления.
16. BeforeParse (TNotifyEvent). Вызывается перед выполнением разбора sql кода.
17. BeforePost (TDataSetNotifyEvent). Вызывается перед выполнением операции post.
18. BeforeRunSql (TDataSetNotifyEvent). Вызывается перед выполнением sql кода
19. BeforeScroll (TDataSetNotifyEvent). Вызывается перед выполнением перемещения на следующую запись.
20. OnCalcFields (TDataSetNotifyEvent). Вызывается при расчете вычисляемых полей.
21. OnCallBack (TOnCallBack).Вызывается при обработке сообщений, полученных от сервера. Принимает ряд параметров:
    * + - * Sender (THtDataSet);
          * Oper (TCallBackOper) – содержит тип CallBack операции, возможные значения: coUnknown (неизвестный тип, при отсутствии ошибок появляться не должен), coUserSend (требуется обработка sql функции "usersend"), coOutMess (требуется обработка sql функции "?"), coIdle (во время длительного ожидания HyTech API может вызываться время от времени событие этого типа), coErrMsg(требуется обработка сообщения об ошибке), coLocErr (требуется обработка сообщения об ошибке клиентской части), coAdminMsg (требуется обработка административного сообщения);
          * Code (smallInt) - содержит числовой параметр, имеет смысл, если Oper = coUserSend или Oper = coAdminMsg;
          * Msg (string) – содержит строку сообщения (кодировка OEM), определен для всех типов, кроме coAdminMsg и coIdle.
22. OnDeleteError (TDataSetErrorEvent). Вызывается при возникновении ошибки удаления.
23. OnDbStateChanged (TNotifyEvent). Вызывается при изменении состояния базы данных.
24. OnEditError (TDataSetErrorEvent). Вызывается при возникновении ошибки редактирования.
25. OnElementUpdated (TOnElementUpdated). Вызывается при обновлении элемента результата. Принимает два параметра: Sender (THtDataSet) и UpdateKind (TUpdateKind). Событие возникает после редактирования результата (методы DeleteElement, InsertElement, ReplaceElement). В параметре UpdateKind передается вид редактирования результата.
26. OnFieldTypeConv (TOnFieldTypeConv). Вызывается при преобразовании типов полей (из поля HyTech в поле Delphi). Назначив обработчик на это событие можно изменить ппреобразование по-умолчанию. Принимает ряд параметров:
    * + - * Sender (THtDataSet);
          * Name (PChar) – содержит название поля;
          * aHtType (integer ) – содержит тип поля HyTech;
          * Order (integer) – содержит порядковый номер поля в результате;
          * FldType (TFieldType) – содержит тип поля Delphi.
27. OnGetMacro (TOnGetMacro). Вызывается при получении тела макроса. Принимает ряд параметров:
    * + - * Sender( TObject);
          * S(string) - содержит на входе имя макроса (без лидирующих сиволов #);
          * Found(boolean) -если замена произведена, параметр должен будет установлен в true, иначе - false.
28. OnIdle (TIdleEvent). Используется для создания обработчиков ожидания у компонента.
29. OnNewRecord (TDataSetNotifyEvent). Вызывается при операции добавления записи.
30. OnParserError (TOnParserError). Вызывается при возникновении ошибки в работе препроцессора SQL.
31. OnPostError (TDataSetNotifyEvent). Вызывается при возникновении ошибки операции post.
32. OnRequiredError (TOnRequiredError). Вызывается при возникновении ошибки при работе с обязательно заполняемыми атрибутами.
33. OnSqlPrepared (TOnSqlPrepared). Вызывается после разбора sql кода препроцессором и перед отправкой его серверу.
34. OnUserSend (TOnUserSend). Вызывается при обработке сообщений сервера, полученных по каналу usersend.

В таблице приведены основные методы класса предка THtCustomStoredProc.

Таб. Основные методы класса THtCustomStoredProc

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | function ExecNoSelect: Variant | Выполнить запрос, не создающий результат (не select). Возвращает RetCode. |
|  | function ExecNoSelectEx (const Names: string; const Values: array of variant): Variant; | Выполнить запрос, не создающий результат (не select). Дополнительно можно задать значения Values параметрам с именами Names запроса. Возвращает RetCode. |

# **TCustomMemTable**

Класс является таблицей в памяти. Может использоваться как LookupDataSet или кэш данных. Имеет возможность сохранять/восстанавливать данные в потоке/файле, копировать данные и описания полей из других DataSet, сортировать строки по любому пользовательскому алгоритму. Класс поддерживает интерфейс IDsMarks. Иерархия классов приведена на .



Рис. Иерархия классов

Свойства класса представлены в .

Таб. Свойства класса TCustomMemTable

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | ChangeCount | - | public |
|  | Filtered | - | public |
|  | FirstAfterSort | - | public |
|  | LogChanges | - | public |
|  | NullAsZero | - | public |
|  | Modified | - | public |
|  | ReadOnly | - | public |

Остановимся более подробно на каждом свойстве.

ChangeCount: содержит количество измененных записей.

Filtered: содержит информацию о том, применялась ли фильтрация( существует ли курсор с отфильтрованными записями).

FirstAfterSort: содержит информацию о том, на какой записи необходимо оставить курсор после сортировки, если FirstAfterSort содержит значение «истина», то курсор перемещается на первую строку, иначе – указывает на ту строку, на которой был до начала сортировки.

LogChanges: содержит информацию о том, вести ли журнал изменений.

NullAsZero: содержит информацию о способе представления null, если флаг содержит значение «истина», то вместо null возвращается 0.

Modified: флаг показывает, были ли изменения, если журнал изменений ведется.

ReadOnly: содержит запрет редактирования.

Рассмотрим события компонента. Подробное описание событий, специфичных для HyTech, приведено в разделе, посвященном компоненту THtQuery.

AfterFilter (TNotifyEvent). Вызывается после фильтрации.

AfterSort(TNotifyEvent). Вызывает после сортировки.

BeforeFilter(TNotifyEvent). Вызывается до фильтрации.

OnCancelUpdates (TUpdateEvent). Вызывается при отмене сделанных изменений. Принимает два параметра: Sender (TCustomMemTable) и Before (boolean), событие вызвается дважды – до отмены изменений и после. Второй параметр показывает, в какой момент вызвано событие.

OnCompare (TCompareEvent). Вызывается при сравнении двух строк в процессе сортировки. Сравниваются значения полей - текущее и OldValue. Результат сравнения возвращается в Cmp: больше нуля, если Value больше OldValue; меньше нуля, если Value меньше OldValue и нуль, если значения равны. В этом случае сортировка производится по возрастанию значений. Принимает ряд параметров:

OnFieldsCreated (TNotifyEvent). Вызывается после создания полей в MemTable, Может быть использован для того, чтобы создавать дополнительные поля.

* Sender (TCustomMemTable);
* Cmp (integer) содержит результат сравнения;
* Param (integer) – используется для передачи параметра в функцию сравнвения.

OnUndo (TUndoEvent ). Вызывается при отмене одного последнего изменения. Принимает ряд параметров:

* + - * + Sender (TCustomMemTable);
        + Before (boolean) - показывает, до или после изменения срабатывает событие;
        + Row (integer) – показывает номер строки, для которой отменяется изменение.

Основные методы класса представлены в .

Таб. Основные методы класса TCustomMemTable

|  |  |  |
| --- | --- | --- |
| № | Вызов метода | Описание |
|  | procedure ApplyUpdates; virtual | Осуществляет вызов Reconcile, в наследнике можно переопределить. |
|  | procedure Assign(Source: TPersistent); override; | Позволяет скопировать значения одноименных полей из текущей строки источника данных (source, чаще всего типа THtDataSet или его наследников) в текущую строку класса. При этом компонент должен находиться в состоянии dsEdit или dsInsert. |
|  | procedure CancelUpdates; | Позволяет отменить все ожидающие кэшированные обновления. Операция очищает кэш и восстанавливает компонент в состояние, которое было при открытии. |
|  | procedure CreateFields; override; | Позволяет создать поля. |
|  | function MakeUpdatesDataSet(NeedUpdates: TUpdateStatusSet; ModField: string = ''): TUpdDataSet; | Позволяет построить DataSet, представляющий множество модифицированных записей (если ведется журнал изменений). Построенный DataSet будет содержать строки, модифицированные требуемым (NeedUpdates) образом - удалённые, добавленные или отредактированные. До тех пор, пока существует хоть один TUpdDataSet, компонент не позволяет выполнять редактирование данных. Параметр ModField определяет имя дополнительного поля (TStringField длиной один символ) в создаваемом DataSet, которое определяет, как модифицирована запись: "i","m","d" - добавлена,изменена, удалена соответственно. Если ModField не указан, доп. поле не создаётся. Возвращает созданный набор данных. |
|  | function MakeIUpdDataSet(NeedUpdates: TUpdateStatusSet; ModField: string = ''): IUpdDataSet; | В отличие от предыдущего, возвращает не TUpdDataSet, а IUpdDataSet. |
|  | procedure Reconcile; | Позволяет очистить журнал, применяя изменения, то есть делает изменения легитимными, то есть присвоить modified значение «ложь», coungchanges значение 0. |
|  | procedure RevertRecord(RowNo: integer = 0); | Позволяет вернуть запись в исходное состояние, если не было удаления. Для удаленных записей можно использовать метод UndoLastChange. Значение RowNo по умолчанию соответствует 0, что указывает на то, что удалять надо текущую запись. |
|  | function UndoLastChange(FollowChange: Boolean = true; CallEvent: boolean = true): Boolean; | Позволяет отменить последнее выполненное изменение. Действует последовательно. Если FollowChange = true, то активной после восстановления станет восстановленная строка. Возвращает значение true, если набор данных был изменен и false в противном случае. |
|  | function GetCurrentRecord(Buffer: PChar): Boolean; override; | Позволяет поместить текущую запись в буфер. Возвращает true, если поместить строку в буфер удалось и Buffer не nil. |
|  | function GetUpdateStatus(RowNo: integer = 0): TUpdateStatus; overload; | Позволяет для произвольной строки получить статус изменения - изменена, удалена или добавлена. |
|  | function UpdateStatus: TUpdateStatus; override; | Позволяет получить статус изменения текущей строки. |
|  | procedure FilterCurRec(DoResync: boolean = true); | Позволяет поместить в список отфильтрованных записей текущую запись, даже если по условиям фильтрации этого не должно быть, при этом свойству Filtered присваивается значение true. Параметр показывает, надо ли вызывать стандартный метод Resync. |
|  | procedure FilterCommit; | Позволяет отфильтрованное множество сделать основным, то есть ссылке на основное множество присваивает адрес отфильтрованного (fcurs := ffilteredcurs), признак filtered помечает как false, удаляет из компонента записи, которых не было в списке отфильтрованных. |
|  | procedure SetTmpNoFiltered; | Позволяет временно отменить фильтрацию, - при этом меняются местами активные курсоры на основной список и список отфильтрованных записей. |
|  | procedure RestoreFiltered; | Позволяет вернуть списки на место после отмены временной фильтрации, то есть восстановить фильтрацию после SetTmpNoFiltered. |
|  | function ApplyFilter(FilterProc: TFilerEvent; Param: integer): integer; | Позволяет отфильтровать записи функцией FilterProc. Создает новый курсор с отфильтрованными записями (FFiltCurs) и используется вместо FCurs. Сами записи не дублируются. Вызов с FilterProc = nil отменяет фильтрацию, то есть FFiltCurs уничтожается, для работы используется FCurs. Параметр Param используется при вызове самой функции фильтрации. Для того, чтобы прервать процедуру фильтрации, FilterProc должна вернуть false, оставшиеся при этом записи считаются не прошедшими фильтрацию. Возвращает количество отфильтрованных записей. |
|  | function ApplyFilterL(FilterProc: Pointer{TFilerProc}; Param: Pointer): integer; | Отличается от предыдущей версии тем, что FilterProc - локальная вложенная функция. |
|  | procedure BatchUpdate(BatchProc {TBatchProc}, Param: Pointer; StartRow: integer = 1; Reverse: boolean = false; IgnoreFilter: boolean = false; DoResync: boolean = true); | Позволяет осуществить быструю модификацию набора записей с помощью процедуры BatchProc , принимающей параметр Param . В обработчике можно присваивать новые значения полям. Если IgnoreFilter = true, работа осуществляется с основным курсором, иначе – с курсором отфильтрованных данных. Если Reverse = true, перебор записей осуществляется от StartRow до первой записи. В любом случае в журнал изменений информация об изменениях, сделанных подобным образом, не попадает, события также не вызываются. DoResync показывает, надо ли вызывать метод Resync. С помощью этой процедуры не обязательно изменять записи, можно осуществить перебор всех записей, например, для подсчета суммы. |
|  | procedure CancelFilter; | Позволяет отменить фильтрацию, сделать основным курсор с неотфильтрованными записями, удалить курсор отфильтрованных записей. |
|  | function CopyData(DS: TDataSet; Fst: integer = 0; Cnt: integer = -1; FilterProc: Pointer = nil{TFilerProc}; Param: Pointer = nil): integer; overload; | Позволяет скопировать информацию из источника DS Копирование будет осуществлено, начиная с запипи Fst. Позволяет скопировать записи, отфильтрованные процедурой FilterProc с параметром Param. Возвращает количество скопироанных записей. Также предоставляется возможность указать, игнорировать ли фильтр в исходном наборе данных (IgnoreSrcFilter).  Если MemTable перед выполнением операции имеет поля, то данные копируются только из одноименных полей DS, если MemTable не имеет полей, то предварительно будут скопированы все поля DS процедурой CopyFields, затем будет осуществлен поиск полей по именам (источник и приемник) и непосредственное копирование информации в соответствии с параметрами. MemTable перед выполнением операции может быть неактивен. |
|  | function CopyData(IgnoreSrcFilter: boolean; DS: TCustomMemTable; Fst: integer = 0; Cnt: integer = -1; FilterProc: Pointer = nil{TFilerProc}; Param: Pointer = nil): integer; overload; |
|  | function CopyData(Copier: IDataCopier; Fst: integer = 0; Cnt: integer = -1; FilterProc: Pointer = nil{TFilerProc}; Param: Pointer = nil): integer; overload; |
|  | function CompareBookmarks(Bookmark1, Bookmark2: TBookmark): Integer; override; | Позволяет сравнить позиции закладок Bookmark1 и Bookmark2, возвращает разность между позициями. |
|  | function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions): Boolean; override; | Позволяет найти запись по значениям KeyValues полей KeyFields с учетом опций. Возвращает true в случае успешного поиска. |
|  | function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields: string): Variant; override; | Позволяет найти запись по значениям полей и предоставляет доступ ко всем полям найденной записи. Вместо этой функции рекомендуется использовать методы RowByValues или GoToValues. |
|  | procedure CopyFields(DS: TDataSet; WithCalcs: boolean = true; ClearOld: boolean = true); overload; | Позволяет скопировать поля из источника данных DS. Если WithCalcs = true, копируются также вычисляемые поля. Если ClearOld = true имеющиеся поля в компоененте удаляются. MemTable перед выполнением операции должен быть неактивен. Имеющиеся поля удаляются. |
|  | procedure CopyFields(Copier: IDataCopier); overload; |
|  | procedure DestroyFields; override; | Позволяет удалить поля. |
|  | procedure DisableEvents(Events: TDSEvents); | Позволяет временно отключить события В параметре указывается, на какие события производится отключение реакции ([ dseInsert, dseEdit, dseDelete, dseScroll]). |
|  | procedure EnableEvents(Events: TDSEvents); | Позволяет отменить временное отключение событий. В параметре указывается, для каких событий отменяется временное отключение. |
|  | procedure Exchange(Index1, Index2: integer); | Позволяет поменять местами две строки под номерами Index1 и Index2. |
|  | function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; | Позволяет получить значения поля Field в буфер Bufffer . Возвращает true, если Buffer не пуст. |
|  | function GetOrgFieldValue(Field: TField): Variant; | Позволяет получить исходное значение поля Field, которое было ДО начала всех правок. |
|  | procedure LoadFromFile(FileName: string); | Позволяет загрузить информацию из файла FileName. |
|  | procedure LoadFromStream(S: TStream; AsBinary: boolean; FieldDelimiter: char = #9; FS: PFormatSettings = nil); | Позволяет загрузить информацию из потока S, предоставляет возможность указать разделитель полей, а также указать особенности форматирования PFormatSettings и режим, в котором был записан файл, а, следовательно, в каком следует загружать из него информацию AsBinary. |
|  | procedure LoadFromText(FileName: string; FieldDelimiter: char = #9); | Позволяет загрузить информацию из текстового файла FileName, позволяет указать разделитель полей. |
|  | procedure MoveRow(CurLine, NewLine: integer); | Позволяет переместить строку со старой позиции CurLine на новую позицию NewLine. |
|  | function Reopen( SaveDefaultFields : Boolean) : boolean; | Осуществляет операции закрытия и открытия. В случае, если SaveDefaultFields истинно, при выполнении операции уже созданные поля не пересоздаются. |
|  | function RowByValue(const Fld, Value: string; Options: TLocateOptions; IgnoreFilter: boolean = false): integer; virtual; | Позволяет найти строку по значению Value поля Fld и вернуть ее номер (от 1). Можно указать, игнорировать ли фильтр IgnoreFilter, а также опции поиска. |
|  | function RowByValues(const FieldNames: string; Values: array of Variant): integer; | Позволяет найти строку по значениям Values указанных полей FieldNames, возвращает RecNo . |
|  | function GoToValues(const FieldNames: string; Values: array of Variant): boolean; | Позволяет найти запись по значениям Values полей FieldNames и переместиться на найденную строку (при этом вызывается RowByValues, если строка найдена, осуществляется перемещение). |
|  | procedure SaveAsText(FileName: string; replace: boolean; FieldDelimiter: char = #9; SaveRecCount: boolean = true); | Позволяет сохранить информацию в текстовый файл FileName, если SaveReckCount = true, то в файле в первую строку будет внесено количество записей. Если replace – true, информация в файле будет перезаписана, иначе дописана в конец. |
|  | procedure SaveToFile(FileName: string; replace: boolean); | Позволяет сохранить информацию в файл FileName. В случае, если файл с таким именем уже существует, и replace =true, то информация в файле перезаписывается, иначе новая информация дописывается в конец имеющегося файла. |
|  | procedure SaveToStream(S: TStream; AsBinary: boolean; FieldDelimiter: char = #9; SaveRecCount: boolean = true; FS: PFormatSettings = nil); | Позволяет записать информацию в поток S, задать разделитель поля FieldDelimiter,указать опции форматирования FS, сохранять ли количество записей SaveRecCount. |
|  | function SetUpdateStatus(RowNo: integer; Value: TUpdateStatus; Force: boolean = false): TUpdateStatus; overload; | Позволяет установить статус изменения(Value) записи RowNo вручную. Если Force = true, состояние записывается безусловно, инче - по приоритетам: самый высокий - rsInsert, затем rsEdit, минимальный - rsConst. Возвращает итоговый статус изменения. |
|  | procedure Sort(Param: integer = 0; AfterPos: TAfterSortPos = aspAsProp); | Позволяет отсортировать записи, при этом должен быть назначен обработчик OnCompare. Param может быть использован для передачи параметра в функцию сортировки. Второй параметр, кроме значения по умолчанию, может принимать значения aspFindCurrent, aspFirst. |

# **TMemTable**

Компонент представляет собой набор данных, который хранит в памяти записи, особенность компонента заключается в том, что вычисляемые поля хранятся в каждой записи, в обычном наборе данных вычисляемые поля хранятся в отдельном буфере.

Компонентом поддерживается журналирование измений (по умолчанию журналирование не ведется). Компонент поддерживает интерфейс IDsMarks. Иерархия классов приведена на .



Рис. Иерархия классов

Свойства компонента представлены в .

Таб. Свойства компонента TMemTable

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | False | published |
|  | BlobStrLen | DEF\_BLOBSTRLEN | published |
|  | DataSet | - | published |
|  | FirstAfterSort | false | published |
|  | FullLookup | false | public |
|  | LogChanges | false | public |
|  | ReadOnly | false | published |

Остановимся более подробно на каждом свойстве.

Active: свойству присваивается «истина» в момент активизации компонента.

BlobStrLen: содержит максимальный размер поля (в байтах) типа BLOB при хранении его в компоненте в виде строки.

DataSet: содержит ссылку на набор данных.

FirstAfterSort показывает, устанавливать указатель на первую запись после выполнения сортировки.

FullLookup: показывает, игнорировать ли фильтр при выполнении операции поиска в результате (LookUp).

LogChanges: показывает, надо ли вести историю изменений.

ReadOnly: содержит флаг запрета редактирования записей.

Рассмотрим события компонента.

1. AfterCancel (TDataSetNotifyEvent). Вызывается после отмены операции (вставки, редактирования или удаления).
2. AfterClose (TDataSetNotifyEvent). Вызывается после закрытия компонента.
3. AfterDelete (TDataSetNotifyEvent). Вызывается после операции удаления.
4. AfterEdit (TDataSetNotifyEvent). Вызывается после выполнения операции редактирования.
5. AfterFilter (TNotifyEvent). Вызывается после фильтрации.
6. AfterInsert (TDataSetNotifyEvent). Вызывается после выполнения операции вставки.
7. AfterOpen (TDataSetNotifyEvent). Вызывается после выполнения операции открытия.
8. AfterPost (TDataSetNotifyEvent). Вызывается после выполнения операции post.
9. AfterScroll (TDataSetNotifyEvent). Вызывается после перехода на следующую запись.
10. AfterSort(TNotifyEvent). Вызывает после сортировки.
11. BeforeCancel (TDataSetNotifyEvent). Вызывается до отмены операции (вставки, удаления или редактирования).
12. BeforeClose (TDataSetNotifyEvent). Вызывается до закрытия компонента.
13. BeforeDelete (TDataSetNotifyEvent). Вызывается перед выполнением операции удаления.
14. BeforeEdit (TDataSetNotifyEvent). Вызывается перед выполнением операции редактирования.
15. BeforeFilter(TNotifyEvent). Вызывается до фильтрации.
16. BeforeInsert (TDataSetNotifyEvent). Вызывается перед выполнением операции вставки.
17. BeforeOpen (TDataSetNotifyEvent). Вызывается перед выполнением открытия компонента.
18. BeforePost (TDataSetNotifyEvent). Вызывается перед выполнением операции post.
19. BeforeScroll (TDataSetNotifyEvent). Вызывается перед выполнением перемещения на следующую запись.
20. OnCalcFields (TDataSetNotifyEvent). Вызывается при расчете вычисляемых полей.
21. OnCancelUpdates (TUpdateEvent). Вызывается при отмене сделанных изменений. Принимает два параметра: Sender (TCustomMemTable) и Before (boolean), событие вызвается дважды – до отмены изменений и после. Второй параметр показывает, в какой момент вызвано событие.
22. OnCompare (TCompareEvent). Вызывается при сравнении двух строк в процессе сортировки. Сравниваются значения полей - текущее и OldValue. Результат сравнения возвращается в Cmp: больше нуля, если Value больше OldValue; меньше нуля, если Value меньше OldValue и нуль, если значения равны. В этом случае сортировка производится по возрастанию значений. Принимает ряд параметров:
    * + - * Sender (TCustomMemTable);
          * Cmp (integer) содержит результат сравнения;
          * Param (integer) – используется для передачи параметра в функцию сравнения.
23. OnDeleteError (TDataSetErrorEvent). Вызывается при возникновении ошибки удаления.
24. OnEditError (TDataSetErrorEvent). Вызывается при возникновении ошибки редактирования.
25. OnFieldsCreated (TNotifyEvent). Вызывается после создания полей в MemTable. Может быть использован для того, чтобы создавать дополнительные поля.
26. OnNewRecord (TDataSetNotifyEvent). Вызывается при операции добавления записи.
27. OnPostError (TDataSetErrorEvent). Вызывается при возникновении ошибки операции post.
28. OnUndo (TUndoEvent ). Вызывается при отмене одного последнего изменения. Принимает ряд параметров:
    * + - * Sender (TCustomMemTable);
          * Before (boolean) - показывает, до или после изменения срабатывает событие;
          * Row (integer) – показывает номер строки, для которой отменяется изменение.

# **TCustomSqlMem**

Класс предназначен для кеширования результатов запросов. При активизации создает экземпляр THtQuery, выполняет в нем запрос, копирует в себя полученные данные и уничтожает Query. Удобен для кеширования на клиенте мелких справочников и использования в качестве lookup DataSet. Поддерживает интерфейс IDsMarks. Иерархия классов приведена на .



Рис. Иерархия классов

Свойства класса представлены в .

Таб. Свойства класса TCustomSqlMem

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Aliases | - | public |
|  | AutoActive | false | published |
|  | DataBase | - | public |
|  | CurrencyAsFixed | false | published |
|  | FirstAfterSort | false | published |

Остановимся более подробно на свойствах класса:

Aliases: содержит список алиасов таблиц. После выполнения запроса Aliases содержит список пар алиас=таблица. Если алиас на таблицу не назначен, считается, что алиас совпадает с именем.

AutoActive: свойство показывает, надо ли активизировать компонент при активизации соответствующего DataBase.

DataBase: содержит ссылку на THtDataBase, с которым работает компонент. Можно проставить вручную.

CurrencyAsFixed показывает, какого типа поля создавать для атрибута типа currency(если true – ThtFixPointField, если false – TcurrencyField).

FirstAfterSort: аналогично свойству в предке.

Рассмотрим события класса.

AfterSort (TNotifyEvent). Вызывается после сортировки.

OnQueryError (TNotifyEvent). Вызывается при возникновении ошибки при выполнении запроса в создаваемом THtQuery.

Основные методы класса представлены в таблице

Таб. Основные методы класса TCustomSqlMem

|  |  |  |
| --- | --- | --- |
| № | Вызов метода | Описание |
|  | function DefaultGetMacro(var S: string): boolean; | Позволяет получить тело макроса по его имени S. |
|  | function DoGetMacro(var S: string): boolean | Позволяет выполнить процедуру, назначенную для события OnGetMacro и при необходимости вызвать стандартную функцию DefaultGetMacro. В параметре передается название макроса. |
|  | function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields: string): Variant; override; | Позволяет осуществить поиск в результате. Вместо этой функции рекомендуется использовать метод RowByValue. |
|  | procedure OpenLite | Позволяет открыть класс как обычный memtable без выполнения запроса. |
|  | function ParamByName(const Value: string): TParam; | Позволяет вернуть ссылку на TParam с именем Value. Если такой не найден, вызывается исключение. |
|  | function RowByValue(const Fld, Value: string; Options: TLocateOptions; IgnoreFilter: boolean = false): integer; override; | Позволяет найти строку по значению Value поля Fld с учетом опция поиска, а также с учетом или без учета фильтрации, возвразает номер найденной строки (от 1). Если строка не найдена, функция вернет минус единицу. |

# **TSqlMem**

Иерархия классов приведена на .



Рис. Иерархия классов

Свойства компонента представлены в .

Таб. Свойства компонента TSqlMem

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | false | published |
|  | AutoActive | false | published |
|  | AutoClose | true | published |
|  | AutoOpen | true | published |
|  | BaseName | - | published |
|  | CalcFieldsOnly | false | published |
|  | CalcNames | - | published |
|  | CloseTables | false | published |
|  | DataSource | - | published |
|  | DwordAsLarge | false | published |
|  | FirstAfterSort | false | published |
|  | FullLookup | false | published |
|  | LogChanges | false | published |
|  | MacroList | - | published |
|  | MaxRecords | -1 | published |
|  | MemoLimit | 1024 | published |
|  | Name | - | published |
|  | NoParse | false | published |
|  | OverflowError | true | published |
|  | Params | - | published |
|  | ParamsInMacro | false | published |
|  | Parser | - | published |
|  | ReadOnly | false | published |
|  | SQL | - | published |
|  | Tag | - | published |

Рассмотрим свойства компонента. Свойства Active, AutoActive, BaseName, CalcFieldsOnly, CalcNames, CloseTables, DataSource, DwordAsLarge, MacroList, NoParse, Params, ParamsInMacro, Parser, ReadOnly, SQL аналогичны одноименным свойствам компонента THtQuery.

AutoClose: показывает, закрывать ли компонент с закрытием сессии.

AutoOpen: показывает, активизируется ли компонент автоматически при первом обращении к функциям поиска/позиционирования Удобно при использовании в качестве lookup DataSet: компонент изначально не активен и активизируется автоматически при необходимости.

FirstAfterSort: показывает, устанавливать указатель на первую запись после выполнения сортировки.

FullLookup: показывает, игнорировать ли фильтр при выполнении операции поиска в результате (LookUp).

LogChanges: показывает, надо ли вести историю изменений.

При работе с компонентом можно ввести некоторые ограничения на количество записей или объем памяти, для этого предназначены свойства MaxRecords и MemoLimit.

MaxRecords: содержит максимальное число записей, которое может быть получено с помощью компонента. Отрицательное значение здесь означает неограниченное количество.

MemoLimit: в случае, если значение MaxRecords отрицательно, учитывается значение свойства MemoLimit.

OverflowError: показывает, вызывать ли исключение, если обьем данных в курсоре больше MemoLimit. На приведено сообщение об ошибке, возникающей при превышении допустимого объема памяти.

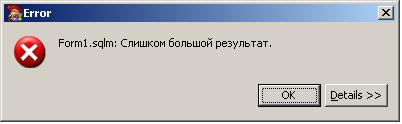


Рис. Ошибка превышения допустимого объема памяти

Рассмотрим события компонента. Будем останавливаться подробно на событиях, специфичных для HyTech.

1. AfterCancel (TDataSetNotifyEvent). Вызывается после отмены операции (вставки, редактирования или удаления).
2. AfterClose (TDataSetNotifyEvent). Вызывается после закрытия компонента.
3. AfterDelete (TDataSetNotifyEvent). Вызывается после операции удаления.
4. AfterEdit (TDataSetNotifyEvent). Вызывается после выполнения операции редактирования.
5. AfterFilter (TNotifyEvent). Вызывается после фильтрации.
6. AfterInsert (TDataSetNotifyEvent). Вызывается после выполнения операции вставки.
7. AfterOpen (TDataSetNotifyEvent). Вызывается после выполнения операции открытия.
8. AfterPost (TDataSetNotifyEvent). Вызывается после выполнения операции post.
9. AfterScroll (TDataSetNotifyEvent). Вызывается после перехода на следующую запись.
10. AfterSort(TNotifyEvent). Вызывает после сортировки.
11. BeforeCancel BeforeCancel (TDataSetNotifyEvent). Вызывается до отмены операции (вставки, удаления или редактирования).
12. BeforeClose (TDataSetNotifyEvent). Вызывается до закрытия компонента.
13. BeforeDelete (TDataSetNotifyEvent). Вызывается перед выполнением операции удаления.
14. BeforeDsOpen (TDataSetNotifyEvent ). Вызывается перед выполнением открытия создаваемого компонентом класса THtQuery (событие THtQuery.BeforeOpen)
15. BeforeEdit (TDataSetNotifyEvent). Вызывается перед выполнением операции редактирования.
16. BeforeFilter(TNotifyEvent). Вызвается перед выполнением операции фильтрации
17. BeforeInsert (TDataSetNotifyEvent). Вызывается перед выполнением операции вставки.
18. BeforeOpen (TDataSetNotifyEvent). Вызывается перед выполнением открытия компонента.
19. BeforeParse (TNotifyEvent). Вызывается перед выполнением разбора sql кода.
20. BeforePost (TDataSetNotifyEvent). Вызывается перед выполнением операции post.
21. BeforeScroll (TDataSetNotifyEvent). Вызывается перед выполнением перемещения на следующую запись.
22. OnCalcFields (TDataSetNotifyEvent). Вызывается при расчете вычисляемых полей.
23. OnCallBack Вызывается при обработке сообщений, полученных от сервера.
24. OnCancelUpdates (TUpdateEvent). Вызывается при отмене сделанных изменений. Принимает два параметра: Sender (TCustomMemTable) и Before (boolean), событие вызвается дважды – до отмены изменений и после. Второй параметр показывает, в какой момент вызвано событие.
25. OnCompare (TCompareEvent). Вызывается при сравнении двух строк в процессе сортировки. Сравниваются значения полей - текущее и OldValue. Результат сравнения возвращается в Cmp: больше нуля, если Value больше OldValue; меньше нуля, если Value меньше OldValue и нуль, если значения равны. В этом случае сортировка производится по возрастанию значений. Принимает ряд параметров:
    * + - * Sender (TCustomMemTable);
          * Cmp (integer) содержит результат сравнения;
          * Param (integer) – используется для передачи параметра в функцию сравнения.
26. OnDbStateChanged (TNotifyEvent). Вызывается при изменении состояния базы данных.
27. OnDeleteError (TDataSetErrorEvent). Вызывается при возникновении ошибки удаления.
28. OnDSFieldsCreated (TNotifyEvent). Вызывается при создании полей создаваемого компонентом класса THtQuery (событие THtQuery. OnFieldsCreated)
29. OnDSFieldTypeConv (TOnFieldTypeConv ). Вызывается при преобразовании типов полей (из поля hytech в поле delphi) создаваемого компонентом класса THtQuery(событие THtQuery.OnFieldTypeConv).
30. OnEditError (TDataSetErrorEvent). Вызывается при возникновении ошибки редактирования.
31. OnFieldsCreated (TNotifyEvent). Вызывается при создании полей.
32. OnFieldTypeConv (TOnFieldTypeConv). Вызывается при преобразовании типов полей (из поля HyTech в поле Delphi). Назначив обработчик на это событие можно изменить ппреобразование по-умолчанию. Принимает ряд параметров:
    * + - * Sender (THtDataSet);
          * Name (PChar) – содержит название поля;
          * aHtType (integer ) – содержит тип поля HyTech;
          * Order (integer) – содержит порядковый номер поля в результате;
          * FldType (TFieldType) – содержит тип поля Delphi.
33. OnGetMacro (TOnGetMacro). Вызывается при получении тела макроса. Принимает ряд параметров:
    * + - * Sender( TObject)
          * S(string) - содержит на входе имя макроса (без лидирующих сиволов #);
          * Found(boolean) -если замена произведена, параметр должен будет установлен в true, иначе - false.
34. OnNewRecord (TDataSetNotifyEvent). Вызывается при операции добавления записи.
35. OnOverflow(TOverflowEvent). Вызывается, если обьем начитываемых данных больше MemoLimit. Принимает ряд параметров:
    * + - * Sender (TObject);
          * DS ( THtDataSet) – набор данных, в котором произошло превышение;
          * Cnt (integer) - при вызове содержит количество строк результата, вмещающихся в MemoLimit. Можно заказать в Cnt чтение любого другого количества строк или вызвать исключение для отмены операции.
36. OnPostError (TDataSetErrorEvent). Вызывается при возникновении ошибки операции post.
37. OnQueryError (TNotifyEvent ). Вызывается при возникновении ошибки при выполнении запроса в создаваемом THtQuery.
38. OnSqlPrepared(TOnSqlPrepared). Вызывается после разбора sql кода препроцессором и перед отправкой его серверу (событие THtQuery. OnSqlPrepared).
39. OnUndo (OnUndo). OnUndo (TUndoEvent ). Вызывается при отмене одного последнего изменения. Принимает ряд параметров:
    * + - * Sender (TCustomMemTable);
          * Before (boolean) - показывает, до или после изменения срабатывает событие;
          * Row (integer) – показывает номер строки, для которой отменяется изменение.

# **THtPump**

Компонент предназначен для буферизованного заполнения таблицы HyTech, используется вместо устаревшего компонента TResultsTable. Иерархия классов приведена на рисунке .



Рис. Иерархия классов

Активация THtPump подразумевает создание принимающей таблицы и запроса информации из неё. SQL запрос для этого может быть предварительно записан в поле Sql. Если SQL не содержит информации, запрос создаётся автоматически согласно списку полей THtPump.Fields. Имя таблицы передаётся как параметр метода Open. Поля THtPump могут быть созданы "вручную". Если список полей пуст, THtPump генерирует исключение. После активации можно приступать к передаче данных: заполнив поля значениями, вызвать метод Post. По мере заполнения буфера данные будут пересылаться в таблицу HyTech.

Свойства компонента представлены в Таб. 33.

Таб. Свойства компонента THtPump

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | - | public |
|  | BaseName | - | published |
|  | BufCount | 3 | published |
|  | BufSize | 65520 | published |
|  | CanInt64 | false | published |
|  | DataBase | - | public |
|  | DateTimeAsDate | false | published |
|  | Fields | - | published |
|  | Int64AsArray9 | - | published |
|  | Name | - | published |
|  | Nullable | - | published |
|  | Posted | - | public |
|  | RetCode | - | public |
|  | Sql | - | published |
|  | TableName | - | published |
|  | Tag | - | published |
|  | TmpTable | true | published |
|  | Transferred | - | public |

Рассмотрим свойства компонента:

Active: свойству присваивается «истина» в момент активизации компонента.BaseName:

BaseName: свойство содержит ссылку на БД.

DataBase: содержит ссылку на THtDataBase, с которым работает компонент. Можно проставить вручную.

DateTimeAsDate: показывает, создавать ли поле Delphi типа ftDate, если атрибут в таблице имеет тип DateTime (или ftDateTime).

BufCount: показывает количество буферов , использовать и изменять не рекомендуется, зарезервировано для будущих версий.

BufSize : содержит размер буфера в байтах, по умолчанию макс размер пакета hytech 65520.

CanInt64: использовать и изменять не рекомендуется, зарезервировано для будущих версий.

Fields: содержит список полей компонента.

Int64AsArray9: показывает, надо ли записывать информацию из поля типа int64 в поле таблицы типа array(9) (можно в компоненте создать поле типа int64, а в базу оно будет записываться как array(9)).

Nullable: показывает, могут ли поля содержать пустые значения, если SQL запрос. генерируется автоматически, то все поля смогут содержать пустые значения.

Posted: показывает количество записей, которое поместили в компонент, равен сумме значения. Transfered и количества записей, находящихся в буфере сейчас.

RetCode: содержит значение retcode(), установленное последним выполненным запросом. Если retcode() в последнем запросе не выполнялся, возвращается нуль.Sql : содержит текст SQL запроса.

TableName : содержит имя таблицы, с которой работает компонент.

TmpTable : показывает, является ли таблица временной.

Transferred: показывает количество записей, переданных в таблицу с момента активизации компонента.

Рассмотрим события компонента. Будем останавливаться подробно на событиях, специфичных для HyTech.

1. BeforeOpen(TNotifyEvent). Вызывается перед открытием компонента ( перед установкой связи с БД).
2. OnException (TNotifyEvent). Вызывается при возникновении ошибки.
3. OnProgress (TNotifyEvent ). Вызывается при выполнении каждой операции записи (после post в методе afterpost).

Основные методы компонента представлены в .

Таб. Основные методы компонента THtPump

|  |  |  |
| --- | --- | --- |
| № | Вызов метода | Описание |
|  | function AddField(FieldName: string; htType: THtFieldType; Size: word; aTag: integer = 0; Key: THtKeyAttr = hkNone ): THtPumpField; | Позволяет вручную создать поле с названием FieldName типа htType размера Size, указать дополнительные параметры поля в Key(возможные значения hkNone, hkKey, hkUnique, hkSurrogate, hkAutoinc), проставить свойство Tag. Функция возвращает созданное поле. |
|  | function AddFieldByField(Field: TField; aTag: integer; FieldName: string = ''; Key: THtKeyAttr = hkNone ): THtPumpField; | Позволяет создать поле компонента по полю Delphi типа TField с названием FieldName, указать дополнительные параметры поля в Key(возможные значения hkNone, hkKey, hkUnique, hkSurrogate, hkAutoinc), проставить свойство Tag. |
|  | procedure Close; virtual; | Позволяет закрыть компонент: записать все записи из буфера в таблицу, обнулить счетчики, освободить память, занимаемую набором данных. |
|  | procedure DestroyFields; | Позволяет уничтожить поля. |
|  | procedure ClearRec; | Позволяет заполнить нулями буфер записей. |
|  | function FieldByName(const FieldName: string): THtPumpField; | Позволяет получить поле по его названию FieldName, в случае, если поиск не дал результатов, генерируется исключение. |
|  | function FindField(const FieldName: string): THtPumpField; | Позволяет получить поле по его названию FieldName, в случае, если поиск не дал результатов, функция возвращает nil. |
|  | procedure Flush; | Позволяет добавить записи из буфера в таблицу ( при этом не срабатывают триггеры) до заполнения буфера( добавление записей происходит автоматически при достижении максимального размера буфера). |
|  | procedure HandleException(E: TObject = nil); virtual; | Позволяет вызвать обработчик события OnException, если он назначен, в противном случае пытается вызвать обработчик события OnException экземпляра THtBase. В параметре можно передавать само исключение. |
|  | procedure Open(aTableName: string = ''); | Позволяет активизировать компонент, если поле SQL содержит запрос, то он будет выполнен. Имя таблицы можно указать в качестве параметра. |
|  | procedure Post; | Позволяет выполнить операцию Post, но в отличие от стандартного набора данных, компонент не надо переводить в режим dsEdit или dsIsert, так как компонент всё время находится в состоянии dsIsert. |
|  | function RecordSize: integer; | Позволяет получить размер записи набора данных. |

# **THtDsPump**

Иерархия классов приведена на .



Рис. Иерархия классов

Поля компонента могут быть созданы методом CopyFields. Если список полей пуст, компонент пытается копировать поля из назначенного источника (property Src или источник, указанный в одном из методов: Copy, Move или StartBatch). Компонент поддерживает несколько дополнительных свойств и методов для перекачки данных из различных наборов данных и компонентов, поддерждивающих интерфейс IPumpSrc.

Если источник (DataSet или IPumpSrc) не содержит сразу всё множество данных для передачи (или реально источников несколько), можно воспользоваться следующей последовательностью:

|  |
| --- |
| StartBatch()  Append()  Append()  ......  StopBatch() |

Свойства компонента представлены в .

Таб. Свойства компонента THtDsPump

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | InBatch | - | public |
|  | Success | - | public |
|  | Src | - | published |

Остановимся более подробно на каждом свойстве.

InBatch: показывает, была ли выполнена процедура StartBatch.

Success: показывает, проведена ли успешно процедура StopBatch.

Src: содержит ссылку на TDataSet.

Рассмотрим события компонента. Будем останавливаться подробно на событиях, специфичных для HyTech.

1. OnNext (TPumpNextEvent). Вызывается при перемещении на следующую запись в источнике (наборе данных). С помощью обработчика события можно организовать фильтрацию данных источника или выгрузить каждую запись нужное количество раз. Принимает ряд параметров: Sender (THtDSPump); и ISrc( IPumpSrc) – источник данных.
2. OnStartProgress (TNotifyEvent). Выполняется до начала перекачки информации из источника.
3. OnStopProgress (TNotifyEvent). Выполняется после окончания перекачки информации из источника.

Основные методы компонента представлены в таблице .

Таб. Основные методы компонента THtDsPump

|  |  |  |
| --- | --- | --- |
| № | Вызов метода | Описание |
|  | function Append(Count: integer = -1; NextAndPost: boolean = true): integer; | Позволяет добавить не больше Count записей из источника, указанного в свойстве Src компонента, если Count < 0, то добавляет все, возвращает количество добавленных записей. |
|  | procedure BindSrcFields(SrcDS: TDataSet = nil); | Позволяет найти и запомнить соответствие полей компонента и набора данных SrcDS. |
|  | procedure Close; override; | Позволяет деактивировать компонент. |
|  | procedure CopyFields(DataSet: TDataSet = nil); | Позволяет скопировать поля из набора данных DataSet или вызвать BindSrcFields, если поля уже созданы и не требуется создать их заново (DefaultFields=false). |
|  | procedure CopyRow(NextAndPost: boolean = true); | Позволяет скопировать значения полей из источника, указанного в свойстве Src компонента. Если значение параметра NextAndPost= true, то будет выполнен post, а у источника выполнить переход на следующую запись. |
|  | procedure Open(ISrc: IPumpSrc; aTableName: string = ''); overload; | Позволяет активировать компонент. Ссылка на источник - набор данных передается в первом параметре. Если параметр aTableName не заполнен, то TableName берется из соответствующего свойства. |
|  | procedure Open(DS: TDataSet; aTableName: string = ''); overload; |
|  | function Move(StartRow: integer = 0; Count: integer = -1): integer; overload; | Позволяет осуществить наиболее простой способ получения информации из DataSet (перед этим необходимо очистить список полей у THtDSPump, назначить property TableName), при выполнении метода компонент будет автоматически активирован, при этом в компоненте будут созданы поля, отвечающие полям DS (если он назначен), создан и выполнен SQL запрос, создающий и открывающий принимающую таблицу, установлено соответствие между полями DataSet и HtDSPump, выполнится перекачка данных, HtDSPump деактивируется (Close) и разрушатся поля HtDSPump. |
|  | function Move(ISrc: IPumpSrc; StartRow: integer = 0; Count: integer = -1): integer; overload |
|  | function Move(DS: TDataSet; StartRow: integer = 0; Count: integer = -1): integer; overload; |
|  | procedure StartBatch(StartRow: integer = 0); overload; | Метод предназначен инициирования начала загрузки информации из нескольких источников или для загрузки информации в том случае, если источник Src не содержит сразу всё множество необходимых данных. В параметре StartRow можно передать номер строки в источнике, с которой начинать передачу. Если StartRow <= 0, передача будет осуществляться с текущей строки. |
|  | procedure StartBatch(ISrc: IPumpSrc; StartRow: integer = 0); overload; |
|  | procedure StartBatch(DS: TDataSet; StartRow: integer = 0); overload |
|  | procedure StopBatch(ErrorCode: integer; DoClose: boolean = true); | Метод предназначен инициирования окончания загрузки данных из нескольких источников. Параметр ErrorCode используется для передачи кода ошибки. Если DoClose присвоено значение true, то компонент будет закрыт. |

# **TDataMove**

Компонент предназначен выгрузки информации из БД Hytech в формат dbf на рабочей станции пользователя, при этом необходимо построить соответствия полей источника и полей пункта назначения, достоинством компонента является возможность использования выражений.

Правила перекодировки полей указываются как строки вида:

|  |
| --- |
| <NAME>:<TYPE><WIDTH>[.<DIGIT>]=<EXPRESSION> |

где <NAME> - имя поля приемника (не более 10 символов),

<TYPE> - тип поля

<WIDTH> - длина поля

<DIGIT> - кол-во знаков после запятой (0 можно не указывать)

<EXPRESSION> - выражение

Допустимые значения для TYPE:

* + - * + C - строка,
        + L - логический,
        + D - дата,
        + N - число.

Допустимые значения для WIDTH:

* + - * + [1..255] - для 'C',
        + 1 - для 'L',
        + 8 - для 'D',
        + [1..20] - для 'N'.

Допустимые значения для DIGIT:

* + - * + >= 0 - для 'C',
        + 0 - для 'L',
        + 0 - для 'D',
        + [0..15] - для 'N'.

Выражения могут содержать имена полей, арифметические операции +,-,\* и / , а так же вызовы некоторых функций.

Иерархия классов приведена на .



Рис. Иерархия классов

Свойства компонента представлены в .

Таб. Свойства компонента TDataMove

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Busy | - | public |
|  | CountRow | - | public |
|  | Functions | - | public |
|  | Mappings | - | public |
|  | StartRow | - | public |
|  | Source | - | public |
|  | Values | - | public |

Остановимся более подробно на каждом свойстве.

Busy: содержит значение true, если осуществляется процесс выгрузки информации.

CountRow: количество записей, которые необходимо выгрузить.

Functions: содержит список функций.

Mappings: содержит правила перекодировки.

StartRow: содержит номер записи с которой надо начать выгружать.

Source: содержит ссылку на источник данных.

Values: содержит список переменных.

Рассмотрим событие компонента:

OnFilter (TDataFilterProc). Вызывается перед выгрузкой каждой строки источника в приемник данных. Принимает ряд параметров:

* + - * + Sender (TDataMove);
        + RowNo(integer) – содержит номер строки, которая выгружается;
        + Skip (boolean) – показывает, прошла строка фильтрацию (false) или не прошла (true).

Основные методы компонента представлены в .

Таб. Основные методы компонента TDataMove

|  |  |  |
| --- | --- | --- |
| № | Вызов метода | Описание |
|  | function MoveTo(dst: TStream ): integer; overload; virtual; abstract; | Позволяет выгрузить информацию в поток dst или сразу в файл FileName (в любом случае в формате dbf). Возвращает количество выгруженных записей. |
|  | function MoveTo( FileName: string ): integer; overload; virtual; abstract; |

# **TDBFMove**

Компонент является визуальным классом- наследником TDataMove. Иерархия классов приведена на .



Рис. Иерархия классов

Свойства компонента представлены в .

Таб. Свойства компонента TDBFMove

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | AutoGen | false | published |
|  | CountRow | - | published |
|  | Description | - | published |
|  | FDescription | - | public |
|  | Functions | - | published |
|  | Mappings | - | published |
|  | Name | - | published |
|  | Source | - | published |
|  | StartRow | - | published |
|  | Tag | - | published |
|  | Values | - | published |

Рассмотрим некоторые свойства компонента:

AutoGen: показывает, необходимо ли автоматически генерировать правила перекодировки.

CountRow: количество записей, которые необходимо выгрузить.

Description: содержит строку, вписываемая в заголовок файла

FDescription: является полем класса, на которое ссылается свойство Description.

Functions: содержит список функций

Mappings: содержит правила перекодировки.

Source: содержит ссылку на источник данных.

StartRow: содержит номер записи с которой надо начать выгружать.

Values: содержит список переменных.

# **TTXTMove**

TXTMove перекачивает данные из DataSet`a в текстовый файл. Правила выгрузки полей указываются как строки вида: < выражение >. Разделители полей необходимо указывать в выражении. Например: FIELD + ','.

Для не строковых полей необходимо явно преобразовывать поле к строке, иначе разделитель будет проигнорирован. Например: char(FIELD) + ','. Поля, содержащие дату необходимо всегда форматировать. Например: FormatDate('dd.mm.yyyy',DOCDATE) + ','.

Иерархия классов приведена на рисунке .



Рис. Иерархия классов

Свойства компонента представлены в .

Таб. Свойства компонента TTXTMove

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | AutoGen | false | published |
|  | CountRow | - | published |
|  | Functions | - | published |
|  | Mappings | - | published |
|  | Name | - | published |
|  | OemText | false | published |
|  | Source | - | published |
|  | StartRow | - | published |
|  | Tag | - | published |
|  | UnixText | false | published |
|  | Values | - | published |

Рассмотрим свойства компонента, отличные от свойств предка:

OemText : показывает, надо ли выгружать файл в Oem кодировке.

UnixTest : показывает, надо ли завершать строку одним символом \n.

События и методы компонента совпадают с событиями и методами предка за исключением того, что оба метода MoveTo уже не являются абстрактными, но имеют реализацию.

# **THTDBFTable**

Компонент является набором данных для БД в формате DBF, не использующим BDE. Иерархия классов приведена на рисунке .

****

Рис. Иерархия классов

Свойства компонента представлены в таблице .

Таб. Свойства компонента THTDBFTable

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | - | published |
|  | AnsiTable | false | published |
|  | DateTimeAsDate | false | published |
|  | DbfType | - | public |
|  | DbfTypeName | - | public |
|  | DesignOnlyFields | True | published |
|  | ForceStringFields | - | published |
|  | InBatch | - | public |
|  | Name | - | published |
|  | ReadOnly | false | published |
|  | RecordDeleted | - | public |
|  | ShowDeleted | false | published |
|  | TableName | - | published |

Рассмотрим некоторые свойства компонента:

Active: свойству присваивается «истина» в момент активизации компонента.

AnsiTable: свойство необходимо для реализации возможности работы с таблицам, кодировка которых ANSI.

DateTimeAsDate: показывает, создавать ли поле Delphi типа ftDate, если атрибут в таблице имеет тип DateTime (или ftDateTime).

DbfType: содержит код (типа byte) – признак типа базы данных.

DbfTypeName: содержит название типа базы данных, расшифрованное по коду.

DesignOnlyFields - показывает, надо ли создавать заново поля при активизации компонента в случае, если поля были созданы в неактивном состоянии компонента. (если DesignOnlyFields= true, то никакие больше поля при активизации не создадутся, иначе после выполнения запроса будут созданы TField для всех отобранных столбцов).

ForceStringFields: содержит список атрибутов, создаваемые поля для которых будут иметь тип fkString.

InBatch: показывает, идет ли процесс пакетного добавления.

ShowDeleted: показывает, надо ли «показывать» удаленные записи. В случае если ShowDeleted, например, операцию вставки информации можно осуществлять непосредственно по физическому номер записи, в противном случае необходимо будет вычислять физический номер записи по логическому номеру и только потом осуществлять манипуляции с информацией.

TableName: содержит путь к файлу и таблице.

Рассмотрим события компонента. Более подробно будем останавливаться на событиях, специфичных для HyTech.

1. AfterCancel (TDataSetNotifyEvent). Вызывается после отмены операции (вставки, редактирования или удаления).
2. AfterClose (TDataSetNotifyEvent). Вызывается после закрытия компонента.
3. AfterDelete (TDataSetNotifyEvent). Вызывается после операции удаления.
4. AfterEdit (TDataSetNotifyEvent). Вызывается после выполнения операции редактирования.
5. AfterInsert (TDataSetNotifyEvent). Вызывается после выполнения операции вставки.
6. AfterOpen (TDataSetNotifyEvent). Вызывается после выполнения операции открытия.
7. AfterPost (TDataSetNotifyEvent). Вызывается после выполнения операции post.
8. AfterScroll (TDataSetNotifyEvent). Вызывается после перехода на следующую запись.
9. BeforeCancel (TDataSetNotifyEvent). Вызывается до отмены операции (вставки, удаления или редактирования).
10. BeforeClose (TDataSetNotifyEvent). Вызывается до закрытия компонента.
11. BeforeDelete (TDataSetNotifyEvent). Вызывается перед выполнением операции удаления.
12. BeforeEdit (TDataSetNotifyEvent). Вызывается перед выполнением операции редактирования.
13. BeforeInsert (TDataSetNotifyEvent). Вызывается перед выполнением операции вставки.
14. BeforeOpen (TDataSetNotifyEvent). Вызывается перед выполнением открытия компонента.
15. BeforePost (TDataSetNotifyEvent). Вызывается перед выполнением операции post.
16. BeforeScroll (TDataSetNotifyEvent). Вызывается перед выполнением перемещения на следующую запись.
17. OnCalcFields (TDataSetNotifyEvent). Вызывается при расчете вычисляемых полей.
18. OnDeleteError (TDataSetErrorEvent). Вызывается при возникновении ошибки удаления.
19. OnEditError (TDataSetErrorEvent). Вызывается при возникновении ошибки редактирования.
20. OnFieldAsString (TOnFieldAsString). Вызывается при обращении к полю как к полю типа строка. Принимает ряд параметров:
    * + - * Sender (THtDbfTable);
          * FieldName (string) – содержит название поля;
          * IsString (boolean) – показывает, успешно ли обращение к полю как к строковому.
21. OnFieldsCreated (TNotifyEvent). Вызывается при создании полей.
22. OnGetFieldDigit (TOnGetFieldDigit). Вызывается при создании поля типа TFloatField. Принимает ряд параметров:
    * + - * Sender (THtDbfTable);
          * Fld (TField) – содержит поле;
          * Digit (integer) - содержит количество цифр после запятой, разрешенных для данного типа поля.
23. OnNewRecord (TDataSetNotifyEvent). Вызывается при операции добавления записи.
24. OnPostError (TDataSetErrorEvent). Вызывается при возникновении ошибки операции post.

Основные методы компонента представлены в таблице .

Таб. 21 Основные методы компонента THtBase

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | procedure AddRecords(const | Позволяет добавить в таблицу буфер с записями, |

Таб. Основные методы компонента THTDBFTable

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | function CompareBookmarks(Bookmark1, Bookmark2: TBookmark): Integer; override; | Позволяет сравнить позиции закладок Bookmark1 и Bookmark2, возвращает разность между позициями. |
|  | function CopyData(Copier: IDbfCopier; RecreateTable: boolean = true; Fst: integer = 0; Cnt: integer = -1): integer; overload; | Позволяет скопировать информацию из источника Copier, поддерживающего интерфейс IDbfCopier. Параметр RecreateTable показывает, надо ли пересоздавать таблицу, если такая уже есть. В параметре Fst передается номер строки, с которой начинать копировать информацию, в параметре Cnt передается количество строк, которое необходимо скопировать. Если Cnt < 0, копируются все строки. Функция возвращает количество добавленных записей. |
|  | function CopyData(DS: TDataSet; RecreateTable: boolean = true; Fst: integer = 0; Cnt: integer = -1): integer; | Позволяет скопировать информацию из набора данных DS. Параметр RecreateTable показывает, надо ли пересоздавать таблицу, если такая уже есть. В параметре Fst передается номер строки, с которой начинать копировать информацию, в параметре Cnt передается количество строк, которое необходимо скопировать. Если Cnt < 0, копируются все строки. Функция возвращает количество добавленных записей. |
|  | procedure CopyFields(DS: TDataSet); overload; | Позволяет скопировать поля из источника DS – набора данных. |
|  | procedure CopyFields(Copier: IDbfCopier); overload; | Позволяет скопировать поля из источника Copier – класса, поддерживающего интерфейс IDbfCopier. |
|  | procedure CreateFields; override; | Позволяет создать поля. |
|  | procedure CreateTable(Descript: string = ''); | Позволяет создать таблицу, в параметре можно указать описание (заголовок) таблицы. |
|  | procedure DestroyFields; override | Позволяет удалить поля. |
|  | function FieldDeclaration(index: integer): string; | Позволяет получить описание типа поля по его номеру index. |
|  | function GetDescript: string; | Позволяет получить описание (заголовок) DBF. |
|  | function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; | Позволяет получить содержимое поля Field в буфер Buffer. Возвращает true в случае, если содержимое получить удалось. |
|  | procedure OpenAsPart(Offset, Bytes: Int64; CanAppend: boolean = false; hFile: integer = -1); | Позволяет работать с файлом DBF, если он составляет только часть таблицы TableName (указанной в свойстве компонента). Параметр Offset показывает смещение первого байта, Bytes - размер DBF; CanAppend - возможность добавления записей в этом режиме. В последнем параметре можно передать хэндл ранее открытого файла, если он известен. |
|  | procedure OpenStream(Stream: TStream; OwnStream, CanAppend: boolean); | Позволяет работать с потоком Stream как с физической таблицей. В параметре CanAppend указывается возможность добавления записей в этом режиме. OwnStream показывает, является ли вызвавший поток его монопольным собственником и надо ли поток уничтожать после окончания всех работ. |
|  | procedure PackTable(Descript: string = ''; ClearDescript: boolean = false); | Позволяет упаковать базу. ClearDescript = true - сбрасывает длину Description в нуль независимо от значения параметра Descript. ClearDescript = false - установить Description в Descript, если Descript не пустая строка. |
|  | procedure SortTable(SortFields : array of string); | Позволяет осуществить сортировку данных согласно списку названий полей SortFields. |
|  | procedure BatchAppend; | Позволяет добавить запись в том случае, если компонент находится в режиме пакетной записи. |
|  | procedure StartBatchAppend; | Метод предназначен для инициирования пакетной записи. |
|  | function StopBatchAppend(Commit: boolean): integer; | Метод предназначен для окончания пакетной записи. Если параметр Commit устанавливается в true, все добавленные в ходе пакетные записи данные сохраняются и функция возвращает количество добавленных записей, в противном случае (commit = false) функция возврает размер DBF-файла перед пакетным добавлением. |
|  | function Translate(Src, Dest: PChar; ToOem: Boolean): Integer; override; | Позволяет перевести текст запроса из кодировки OEM в ANSI и наоборот, источник текста приводитсяв Src, текст записываеся в Dest, если ToOem = true, то перевод осуществляется в направлении ANSI->OEM, иначе наоборот. Возвращает размерность строки Src. |
|  | procedure UnsortTable; | Позволяет отменить сортировку данных. |

# **TMacList**

Компонент предназначен для работы с макросами, хранит текст макросов в DFM файле. Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента представлены в таблице .

Таб. Свойства компонента TMacList

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Child | false | published |
|  | Macro | - | public |
|  | Name | true | published |
|  | Names | true | public |
|  | SQL | - | published |
|  | Tag | 0 | published |

Остановимся на некоторых свойствах более подробно:

Child: содержит ссылку на дочерний компонент типа TCustomMacrosList, если таковой имеется.

Macro: позволяет получить тело макроса по имени, если макрос не найден, возвращается MacroName.

Names: содержит список имен имеющихся макросов.

SQL: является хранилищем макросов.

Рассмотрим событие компонента:

OnChanged (TNotifyEvent) Вызывается при изменении свойств компонента (изменение имени файла, тела макроса).

# **TMacrosList**

Для удобства хранения и модификации запросов в библиотеке имеется компонента TMacrosList. Этот компонент работает с текстовыми файлами, содержащими SQL запросы. Каждый запрос в файле имеет имя, указываемое в отдельной строке в квадратных скобках. Сам запрос начинается со следующей строки и продолжается до имени следующего запроса или до конца файла. В процессе активизации TMacrosList индексирует файл для быстрого доступа к запросам по их именам. Компоненты THtDataSet и THtBase имеют свойства MacrosList – ссылку на компонент TMacrosList, и, если в запросе встречается идентификатор, начинающийся двумя «диезами» (##), то препроцессор считает этот идентификатор именем запроса (макроса) и обращается к MacrosList за расшифровкой.

Файл должен иметь следующую структуру:

|  |
| --- |
| [Имя макроса]  тело макроса  [Имя макроса]  тело макроса  ..................... |

где имя макроса - произвольный текст, а тело макроса: произвольное количество строк (не менее одной); открывающая квадратная скобка первым символом строки не допускается.

Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента представлены в таблице .

Таб. Свойства компонента TMacrosList

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Active | - | public |
|  | Child | false | published |
|  | ExpandedFileName | - | public |
|  | FileName | - | published |
|  | Macro | - | public |
|  | Name | - | published |
|  | Names | true | public |
|  | SQL | - | public |
|  | Tag | 0 | published |

Остановимся на свойствах компонента, отличных от свойств предка.

Active: устанавливается в true при активизации компонента.

ExpandedFileName: содержит полный путь к файлу.

FileName: содержит идентификатор хранилища.

Рассмотрим событие компонента:

1. OnChanged (TNotifyEvent). Вызывается при изменении свойств компонента (изменение имени файла, тела макроса).

# **TSQLParser**

Компонент является препроцессором SQL и выполняет преобразование получаемого текста в текст на основе имеющихся во входном тексте инструкций. Весь не относящийся к инструкциям препроцессора текст транслируется без изменений (есть режим запрета трансляции комментариев). Все инструкции препроцессора чувствительны к регистру (Case sensitive). Препроцессор может находиться в одном из следующих состояний:

* + - * + pmNothing: ничего не делает.
        + pmParams: составление списка параметров запроса. Входной текст сканируется, выходной не формируется.
        + pmProcess: основной процесс.
        + pmFindError: поиск позиции ошибки.

Иерархия классов приведена на .



Рис. Иерархия классов

Свойства компонента представлены в таблице .

Таб. Свойства компонента TSQLParser

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | AddLastSemi | true | published |
|  | Functions | false | published |
|  | IgnoreParams | false | published |
|  | IgnoreQrySplit | false | published |
|  | IncludeComment | false | published |
|  | MacroList | - | published |
|  | Name | - | published |
|  | ParamsInMacro | false | published |
|  | ParseAsMacro | false | published |
|  | SQL3 | false | published |
|  | TabSize | 8 | published |
|  | Tag | - | published |

Остановимся более подробно на каждом свойстве.

AddLastSemi: свойство указывает, необходимо ли добавлять ли точку с запятой в конец результата, если там таковой нет. HyTech SQL требует наличия в конце каждого оператора терминального символа ";". Парсер может добавлять его автоматически в случае отсутствия.

Functions: показывает, используются ли пользовательские функции. Пользователь может определить произвольное количество своих функций препроцессора, вызовы которых можно использовать во входных текстах препроцессора. Когда препроцессор встретит во входном тексте прагму с именем одной из пользовательских функций, будет вызван обработчик OnSubst этой функции, в котором можно получить формальные и фактические параметры вызова и сформировать выходной текст.

IgnoreParams: показывает, игнорировать ли стандартные параметры в запросе.

IgnoreQrySplit: свойство указывает, надо ли парсеру игнорировать разделитель запросов. (символ "\", единственный в строке). По умолчанию (IgnoreQrySplit = false) парсер вызывает обработчик OnSubQuery, передавая ему обработанную часть, затем сбрасывает свое состояние в исходное и продолжает обработку входного текста. Если же IgnoreQrySplit = true, то разделители запросов рассматриваются как обычные символы и попадают в выходной поток.

IncludeComment: свойство указывает, надо ли передавать комментарии в выходной текст (по умолчанию комментарии в выходной текст не попадают, но при отладке SQL их полезно включить).

MacroList: содержит ссылку на хранилище макросов.

ParamsInMacro: указывает, есть ли необходимость искать параметры (Params) в макросах. Когда препроцессор обрабатывает sql-код с целью извлечения списка параметров, ParamsInMacro = true заставляет его также получать и просматривать тела макросов.

ParseAsMacro: показывает, каким образом интерпретировать знак «%» в запросе. в HT Sql «%» - это оператор взятия остатка от деления. В макросах знак «%» используется , как в \*.bat файлах, для указания параметра: %1, %2 и т.д. В основном запросе (который передан парсеру) «%» - всегда оператор остатка, когда происходит разбор тела макроса, то написанные слитно процент и цифра интерпретируются как номер параметра. Установка свойства ParseAsMacro в true включает такую интерпретацию и для основного запроса.

SQL3: содержит признак того, что разбирается код, удовлетворяющий стандарту SQL93.

TabSize: содержит размер табуляции. Используется функциями определения места ошибки.

# **THtUpdateSQL**

TUpdateSQL предназначен для модификации данных на сервере с помощью заранее подготовленных SQL-предложений. Он позволяет определить различные операторы SQL для удаления, вставки и модификации записи, в том числе отличные от простых операторов delete, insert, update. Эти операторы SQL содержатся в свойствах DeleteSQL, InsertSQL, ModifySQL соответственно.

Компонент позволяет назначить произвольные SQL код, модифицирующий THtDataSet. Компонент выполняет SQL выражения при операциях THtDataSet.Post и THtDataSet.Delete, возвращая результат выполнения THtDataSet'у. Напрямую методы THtUpdateSQL выполнять не следует, так как все необходимые действия производятся его THtDataSet'ом. Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента представлены в таблице .

Таб. Свойства компонента THtUpdateSQL

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | DataSet | - | public |
|  | DeleteSQL | - | published |
|  | InsertSQL | - | published |
|  | ModifySQL | - | published |
|  | Parser | - | published |
|  | RetCode | - | public |
|  | SelRecSQL | - | public |

Остановимся более подробно на свойствах компонента.

DataSet: содержит ссылку на компонент THtDataSet, с которым осуществляется работа.

DeleteSQL: содержит sql код, выполняющийся при вызове метода DataSet.Delete.

InsertSQL: содержит sql код, выполняющийся при вызове метода DataSet.Insert.

ModifySQL: содержит sql код, выполняющийся при вызове метода DataSet.Update.

Parser: содержит ссылку на внешний препроцессор SQL.Если Parser не назначен, создаётся временный экземпляр TSqlParser'а для обработки запроса, иначе используется Parser. Внешний препроцессор позволяет более широко управлять процессом, т.к. доступны все его события и методы.

RetCode: содержит значение retcode(), установленное последним выполненным запросом. Если retcode() в последнем запросе не выполнялся, возвращается нуль.

SelRecSQL: содержит текст sql-запроса к БД.

Рассмотрим события компонента THtUpdateSQL.

1. OnExecuted(TOnExecuted). Вызывается после выполнения SQL кода на изменение данных. Принимает ряд параметров:
   * + - * Sender (THtUpdateSQL);
         * DataSet (THtDataSet) - содержит ссылку на набор данных;
         * UpdateKind (TUpdateKind) - содержит вид изменения;
2. OnGetMacro (TOnGetMacro). Вызывается при получении тела макроса. Принимает ряд параметров:
   * + - * Sender( TObject);
         * S(string) - содержит на входе имя макроса (без лидирующих сиволов #);
         * Found(boolean) -если замена произведена, параметр должен будет установлен в true, иначе - false.
3. OnSetParams (TOnSetParams). Вызывается при установке значений дополнительных параметров запросов на редактирование. В SQL запросах на редактирование в компоненте можно задавать дополнительные параметры, начинающиеся с восклицательного знака. Значения этим параметрам нужно присваивать в этом обработчике. Принимает ряд параметров:
   * + - * Sender (TObject);
         * DataSet (THtDataSet) – содержит ссылку на набор данных;
         * UpdateKind (TUpdateKind) - содержит вид изменения;
4. OnSetParSel (TOnSetParams). Вызывается при установке значений параметров запросов на выбор. Подробнее см. событие OnSetParams.
5. OnSelRec (TNotifyEvent). Вызывается после выполнения запроса на выбор информации.

Методы компонента представлены в таблице .

Таб. Основные методы компонента THtUpdateSQL

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | procedure Apply(UpdateKind: TUpdateKind); | Позволяет обновить (присвоить) значения параметров и выполнить sql код, осуществляющий указанное в UpdateKind изменение |
|  | procedure ExecSQL(UpdateKind: TUpdateKind); | Позволяет выполнить sql код, осуществляющий указанное в UpdateKind изменение |
|  | procedure RefreshRow; | Позволяет обновить текущую строку информацией из базы |
|  | procedure SetParams(UpdateKind: TUpdateKind); | Позволяет обновить (присвоить) значения параметров |

# Закладка HtComp

# **TExcel**

Компонент TExcel предназначен для работы с документами MS Excel. Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента приведены в .

Таб. Свойства компонента TExcel

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | AsNew | - | public |
|  | BatchMax | - | published |
|  | BatchMin | - | published |
|  | BatchOn | - | public |
|  | Connected | - | public |
|  | DDE | - | public |
|  | Decimals | - | published |
|  | ExeName | - | published |
|  | FirstRow | - | public |
|  | LastCol | - | public |
|  | Lines | - | public |
|  | Name | - | published |
|  | Ready | - | public |
|  | Selection | - | public |
|  | Tag | - | published |

Рассмотрим более подробно свойства компонента:

AsNew: свойство доступно только для чтения. Показывает, что Excel не был запущен ранее, а загружен компонентой, т.е., если AsNew false, подключение происходит к ранее запущенному Excel.

BatchMax, BatchMin: задают размер промежуточного буфера при пакетной передаче данных.

BatchOn: содержит признак процесса пакетной загрузки данных в Excel.

DDE: содержит ссылку на экземпляр класса TDdeCLientConv, используемый для взаимодействия с приложением Excel

Connected: свойству присваивается значение true, если связь с Excel установлена успешно.

Decimals: содержит требуемую точность десятичного числа при передаче его значения в Excel, т.е. требуемое количество знаков после запятой

ExeName: содержит полное имя (включая абсолютный путь) запускаемого файла Excel.

FirstRow: содержит номер первой строки.

LastCol: содержит номер последнего столбца.

Lines: является контейнером для хранения данных, пересылаемых в Excel при пакетном заполнении.

Ready: содержит признак готовности приложения Excel.

Selection: содержит адрес выделенной ячейки листа Excel, например '[11]Лист1!R6C1'.

Рассмотрим события компонента TExcel. Более подробно будем останавливаться на событиях, специфичных для HyTech.

1. OnClose (TNotifyEvent ).Вызывается при закрытии приложения Excel.
2. OnGetColumnHeader (TGetColumnHeader).Вызывается при обращении к заголовку поля (при экспорте из TDataSet или TDbGrid). Принимает ряд параметров:
   * + - * Sender (TObject);
         * num (integer) - содержит номер поля;
         * Value (string) – содержит название поля.
3. OnOpen (TNotifyEvent).Вызывается при открытии приложения Excel.
4. OnProgress (TProgressEvent). Вызывается при загузке данных из/в Excel. Перед началом загрузки Position = ppFirst; Value = количество выгружаемых в процессе экспорта записей, в процессе экспорта Position = ppNext; Value = кол-во выгруженных к этому моменту записей, при работе с посдней записью Position = ppLast, Value = количество выгруженных записе. Принимает ряд параметров:
   * + - * Sender (TObject);
         * Position(TProgressPosition) – отражает состояние выгрузки;
         * Value (integer) – содержит количество записей;
         * Stop (boolean) – если содержит значение «true», то процесс выгрузки прекратится (Position будет равен ppLast).

Методы компонента представлены в таблице .

Таб. Основные методы компонента TExcel

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | procedure BatchCancel; | Позволяет закончить пакетное заполнение. |
|  | procedure BatchSend; | Позволяет инциировать пакетное заполнение. |
|  | procedure BatchStart(FirstRow, FirstCol: Integer); | Позволяет подготовить компонент к пакетному заполнению ячеек листа Excel, начиная с FirstRow, FirstCol. |
|  | procedure BatchStartDef(Cols: integer); | Позволяет инициировать пакетное заполнение ячеек листа Excel, начиная с текущей выделенной ячейки. |
|  | procedure CloseMacroFile; | Позволяет закрыть файл макросов. |
|  | procedure Connect; | Позволяют установить связь и разорвать связь между приложением и MS Excel. |
|  | procedure Disconnect; |
|  | procedure DoRect(Top, Left, Bottom, Right: Integer; Data: TStrings; Request: Boolean); | Позволяет заполнить указанную область листа указанными значениями. Принимает 6 входных параметров: индекс строки начала области Top, индекс столбца начала области Left, индекс строки окончания области Bottom, индекс столбца окончания области Right, исходные данные Data. Параметр Request показывает, надо ли отправлять значение (false) или получать значения (true). |
|  | procedure DoExport(DS: TDataSet; const FldList: array of TField; const Header: string; Fst, Lst: integer); | Позволяет экспортировать информацию из источника данных типа TDataSet(DS) или его наследников (например, THtQuery) в Excel, при этом указывается список полей FldList, заголовок Header. В Fst и Lst передаются номера первой строки и последней строки для экспорта. |
|  | procedure Exec (const Cmd : string) | Позволяет выполнить любые процедуры/функции Excel, переданные в Cmd, при этом в процедуру/функцию Excel можно передавать параметры. |
|  | procedure ExportData(DS: TDataSet; const Flds: string; WithHeader: boolean; Fst, Lst: integer); | Позволяет экспортировать информацию из источника данных типа TDataSet или его наследников( например, THtQuery) в Excel. Процедура принимает пять параметров:   * ссылку на источник данных (TDataSet), * строку, содержащую список экспортируемых атрибутов, если задается пустая строка, то по экспортируются все атрибуты, если необходимо указать название нескольких атрибутов, они перечисляются через запятую; * логическое значение, если true, то экспорт осуществляется с заголовками (алиасами атрибутов), в противном случае без заголовков. * номер первой экспортируемой записи; * номер последней экспортируемой записи.   Выполняет ряд проверок и вызывает метод DoExport. |
|  | procedure ExportGrid(Grd: TDbGrid; WithHeader: boolean; Fst, Lst: integer); | Позволяет экспорта информацию из компонента типа TDbGrid или его наследников в Excel. Процедура принимает четыре параметра:   * ссылку на источник данных (TDbGrid), * логическое значение, если true, то экспорт осуществляется с заголовками, в противном случае без заголовков. * номер первой экспортируемой записи; * номер последней экспортируемой записи. |
|  | function GetCell(Row, Col: Integer): string; | Позволяет получить значение указанной ячейки. Индекс строки и столбца указывается в качестве входных параметров Row и Col. |
|  | function GetCurrSel: TPoint | Позволяет получить координаты выделенной ячейки. |
|  | procedure GetRange(R: TRect; Lines: TStrings) | Позволяет получить значение указанной области. Границы области задаются в параметре типа TRect, целевые значения возвращаются во втором параметре Lines типа TStrings. |
|  | procedure Maximize; | Позволяют проводить стандартные операции с окном Excel – развернуть во весь экран, свернуть. |
|  | procedure Minimize; |
|  | procedure OpenMacroFile(const Fn: TFileName; Hide: Boolean); | Позволяет открыть файл макросов Fn. Параметр Hide указывает, отображать ли (hide= false) открытый файл. |
|  | procedure OpenTml(const Fn: TFileName); | Позволяет открыть файл с указанным именем Fn ( при необходимости создать). |
|  | procedure ProcessMessages | Позволяет обрабатывать сообщения, имеющиеся на данный момент в очереди сообещний. |
|  | procedure PutDay(Row, Col: Integer; d: TDateTime);virtual; | Процедуры позволяют заполнить целевую ячейку информацией соответствующего типа. Каждая процедура принимает три параметра: индекс строки R, индекс столбца C, значение соответствующего типа. |
|  | procedure PutExt(Row, Col: Integer; e: Extended); virtual; |
|  | procedure PutInt(Row, Col: Integer; i: Longint); virtual |
|  | procedure PutStr(Row, Col: Integer; const s: string) |
|  | procedure Restore | Позволяют восстановить окно Excel. |
|  | procedure Run (const Mn : string); | Позволяет запустить макрос. Имя макроса необходимо указать при вызове в параметре Mn. |
|  | procedure Select(Row, Col: Integer); | Позволяет выделить ячейку на листе Excel. Принимает два параметра – номер строки Row и номер столбца Col. |
|  | procedure SelRange(R1, C1, R2, C2: integer); | Позволяет установить курсор на область листа Excel. Границы области задаются 4ми выходными параметрами – индекс строки начала области R1, индекс столбца начала области C1, индекс строки окончания области R2, индекс столбца окончания области C2. |
|  | procedure Wait | Метод предназначен для реализации ожидания ответа от Excel. |

# **THtxDbFilter**

Компонент позволяет осуществлять реализовать фильтрацию данных на основе условий. Атрибуты, участвующие в условиях, указываются в процессе разработки приложения, также есть возможность указать справочный набор данных из БД или любой другой набор значений. Более подробная информация об условиях находится в следующем разделе.

Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента приведены в .

Таб. . Свойства компонента THtxDbFilter

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Align | alNone | published |
|  | Anchors | [akLeft,akTop] | published |
|  | BiDiMode | bdLeftToRight | published |
|  | BorderStyle | bsSingle | published |
|  | Canvas | - | public |
|  | Color | clBtnFace | published |
|  | Conditions | - | published |
|  | ConditionTitles | () | published |
|  | Constraints | - | published |
|  | DragCursor | crDrag | published |
|  | DragKind | dkDrag | published |
|  | DragMode | dmManual | published |
|  | EditData | - | public |
|  | Enabled | true | published |
|  | Font | - | published |
|  | ItemData[Index: Integer] | - | public |
|  | ParentBiDiMode | true | published |
|  | ParentColor | false | published |
|  | ParentFont | true | published |
|  | ParentShowHint | true | published |
|  | PopupMenu | - | published |
|  | ScrollBars | ssBoth | published |
|  | ShowHint | false | published |
|  | TabOrder | - | published |
|  | TabStop | true | published |
|  | Titles | () | published |
|  | UseOrBind | false | published |

Рассмотрим более подробно свойства компонента. Свойства Align, Anchors, BiDiMode, BorderStyle, Canvas, Color, Constraints, DragCursor, DragKind, DragMode, Enabled, Font, ParentBiDiMode, ParentFont, ParentColor, ParentShowHint, PopupMenu, ScrollBars, ShowHint, TabOrder, TabStop унаследованы от предков.

Conditions: содержит список возможных условий, информация об условиях приведена в следующем разделе.

ConditionTitles: позволяет задать заголовки четырех столбцов компонента, значения по умолчанию – «Название», «Условие», «Значение», «Значение».

EditData: содержит информацию об условиях в процессе редактирования.

ItemData[Index: Integer]: позволяет получить условие по номеру.

Titles: в свойстве содержатся заголовки стобцов компонента.

UseOrBind: показывает, разрешать ли пользователю применять «ИЛИ» при формировании условий. Не рекомендуется применять, если исходный запрос обращается больше, чем к одной таблице.

Рассмотрим свойства компонента. Свойства OnClick, OnDblClick, OnDragDrop, OnDragOver, OnEndDock, OnEndDrag, OnEnter, OnExit, OnKeyDown, OnKeyPress, OnKeyUp, OnMouseMove, OnMouseUp, OnMouseWheelDown, OnMouseWheelUp, OnStartDock, OnStartDrag, унаследованы от компонентов – предков, на них останаливаться не будем.

Более подробно будем останавливаться на свойствах, специфичных для HyTech.

1. OnEditButtonClick(ThtxFilterButtonClick). Вызывается при нажатии на кнопку выбора значения для сравнения, если стиль кнопки, указанный в данном условии, равен fbEllipsis. Использует ряд параметров:
   * + - * Sender (TObject);
         * Cond (ThtxCondition) – содержит условие, для которого которого выбирается значение;
         * Value (string) – содержит значение, выбранное пользователем;
         * Key (string) – содержит ключ значения, выбранного пользователем ;
         * Accept (boolean) – показывает, принимается ли выбранное значение, если false, значение не будет принято.
2. OnLookupDropDown (ThtxFilterLookupDown); Вызывается при отображении списка возможных значений LookUp для сравнения. Использует ряд параметров:
   * + - * Sender (TObject);
         * Cond (ThtxCondition) – содержит условие, для которого которого выбирается значение;
         * Text (string) - содержит имеющееся в настоящий момент значение( выбранное ранее).
3. OnMakeText (ThtxFilterOnMakeText). Вызывается при формировании строк с условиями. Использует ряд параметров:
   * + - * Sender (TObject);
         * index (integer) – содержит количество сформированных условий;
         * data (PhtxResultData) – содержит описание условия и собственно выбранные значения(вид условия, значения для сравнения);
         * Text (string) – содежит получившийся в результате текст условия.
4. OnSelLookupItem (ThtxFilterLookupClose) Вызывается при выборе значения из списка LookUp. Использует ряд параметров:
   * + - * Sender (TObject);
         * Cond (ThtxCondition) - содержит условие, для которого которого выбирается значение;
         * Locate (boolean) – true, если в выпадающем списке найдено текущее значение поля;
         * Value (string) – содержит значение, выбранное пользователем;
         * Key (string) – содержит ключ значения, выбранного пользователем ;
         * Accept (boolean) – показывает, принимается ли выбранное значение, если false, значение не будет принято.

Основные методы компонента представлены в таблице .

Таб. Основные методы компонента THtxDbFilter

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | procedure AddItem(fBind :TBind; fData :ThtxCondition; fCond :TCond; fVal1 :variant; fVal2 :variant; fKey1 :variant; fKey2 :variant); | Позволяет программно добавить в компонент «шаблон» условия, в параметре fBind можно передать «вид связки» - «и» / «или», в параметре fData – описание условия, в остальных параметрах – вид условия(=, like …), ключевые значения и отображаемые значения. |
|  | procedure Append; | Позволяет добавить условие в конец списка условий. |
|  | procedure Cancel; | Позволяет отменить внесенные в условия правки. |
|  | procedure Clear; | Позволяет удалить сформированные ранее условия. |
|  | function Count: integer; | Позволяет получить количество уже сформированных условий. |
|  | procedure Delete; | Позволяет удалить условия. |
|  | procedure Edit; | Позволяет перевести компонент в режим редактирования условий. |
|  | procedure Insert; | Позволяет добавить условие в список условий. |
|  | function MakeText( nTag: integer = 0 ): string; | Позволяет сформировать строку с условиями. В параметре можно передать Tag условия, которое необходимо сформировать, если параметр оставить равным 0, будут сформированы все условия. |
|  | procedure Post; | Позволяет применить все правки, внесенные в список условий |
|  | function State: ThtxResultState; | Позволяет получить состояние компонента (rsBrowse, rsEdit, rsInsert). |

# THtxCondition

Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента приведены в .

Таб. . Свойства компонента THtxCondition

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | ArrayDecimal | true | published |
|  | ArraySize | 0 | published |
|  | ArrayTerm | #0 | published |
|  | ButtonStyle | fbAuto | published |
|  | CaseSensitive | false | published |
|  | DisplayName | - | published |
|  | DisplayValueType | dtChar | published |
|  | DropDownOnly | false | published |
|  | DropDownRows | 12 | published |
|  | EditMask | - | published |
|  | Enabled | - | published |
|  | FieldName | - | published |
|  | Filter | - | public |
|  | IsReadOnly | - | public |
|  | KeyList | - | published |
|  | KeyValueType | dtChar | published |
|  | LookupDataSet | - | published |
|  | LookupDisplayField | - | published |
|  | LookupKeyField | - | published |
|  | MaxLength | 0 | published |
|  | Multiplier | 1 | published |
|  | OutMask | - | published |
|  | ParsText | - | published |
|  | PickList | - | published |
|  | ShortCut | 32808 ('Alt+Down') | published |
|  | TrackChars | false | published |
|  | UseKeyValue | false | published |

Рассмотрим более подробно некоторые свойства.

ArrayDecimal: показывает, являются числа, хранимые в массиве, десятичными или шестнадцатиричными.

ArraySize: содержит размер массива, влияет на поля типа dtArray.

ArrayTerm: содержит символ, которым будет дополнен массив до его фактической длинны.

ButtonStyle: определяет стиль кнопки в столбце «значение», fbAuto соответствует , fbEllipsis - .

CaseSensitive: определят, учитывать ли регистр строки при сравнении (если сравнивается строка).

DisplayName: позволяет задать название условия, отображаемое пользователю.

DisplayValueType: позволяет задать тип полей, используемых в сравнении.

DropDownOnly: показывает, выбирать ли значения для сравнения из списка PicList, иначе разрешен ввод значений пользователем.

DropDownRows: показывает, сколько строк отображать единовременно в выпадающем списке.

EditMask: содержит маску, ограничивающую значения, которые могут быть выбраны (более подробно см. справку по TEditMask).

Enabled: показывает, использовать ли условие ( при Enabled = false условие можно будет выбрать, но оно не участвовать при вызове MakeText0.

FieldName: содержит название поля основного набора данных, по которому должна осуществляться фильтрация.

Filter: содержит ссылку на компонент THtxDbFilter, к которому принадлежит данное условие.

IsReadOnly: показывает, можно ли пользователю вписывать свои значения для сравнения.

KeyList: хранит список значений, используемых для подстановки и поиска, при этом для отображения используются соответствующие строки из свойства PicList, если UseKeyValue = true, в противном случае и для отображения, и для поиска используются строки из свойства PickList.

KeyValueType: содержит тип ключевого значения.

LookupDataSet: в свойстве можно указать набор данных, используемый для LookUp.

LookupDisplayField: в свойстве можно указать отображаемое поле набора данных, используемого для LookUp.

LookupKeyField: в свойстве можно указать подставляемое поле набора данных, используемого для LookUp.

MaxLength: в свойстве можно указать макисмальную длину значения (например, для строковых полей нет смысла вводить больше символов, чем длина поля).

Multiplier: содержит величину, на которую умножаются значения для сравнения, используется только при участии в сравнении поля типа dtCurrency.

OutMask: содержит маску, которая используется для форматирования при подставновке в скрипт.

ParsText: содержит готовую строку для подстановки в выражение.

PickList: может содержать список возможных значений для сравнения.

ShortCut: позволяет задавать кнопку (или сочетание нескольких), по нажатию на которую будет открываться выпадающий список.

TrackChars: показывает, позиционироваться ли на ближайшую похожую (в выпадающем списке) строку при вводе каждого очередного символа, по умолчанию false.

UseKeyValue: показывает, использовать ли для подстановки и поиска значений значения, содержащиеся в свйостве KeyList.

Рассмотрим событие класса.

1. OnMakeText (ThtxFilterOnMakeText). Вызывается при формировании строк с условиями. Использует ряд параметров:
   * + - * Sender (TObject);
         * index (integer) – содержит количество сформированных условий;
         * data (PhtxResultData) – содержит описание условия и собственно выбранные значения(вид условия, значения для сравнения);
         * Text (string) – содежит получившийся в результате текст условия .

# **TXTreeView**

Первый вариант компонента, предназначенного для работы дерева с базой данных. Недостатком компонента является отсутствие невидимых узлов, то есть если узел есть, то он виден, существующих невидимых узлов не бывает. В настоящий момент устарел.

Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента приведены в .

Таб. Свойства компонента TXTreeView

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | Align | alNone | published |
|  | AlignWithMargins | false | published |
|  | Ancors | [akLeft,akTop] | published |
|  | AutoExpand | false | published |
|  | BiDiMode | bdLeftToRight | published |
|  | BorderStyle | bsSingle | published |
|  | BorderWidth | - | published |
|  | Canvas | - | public |
|  | ChangeDelay | 0 | published |
|  | Changing | false | published |
|  | Contrstraints | - | published |
|  | CollapseDelChild | false | published |
|  | Color | clWindow | published |
|  | Ctl3D | true | published |
|  | Cursor | crDefault | published |
|  | DragCursor | crDrag | published |
|  | DragKind | dkDrag | published |
|  | DragMode | dmManuak | published |
|  | DropTarget | - | public |
|  | Enabled | true | published |
|  | Font | - | published |
|  | Height |  | published |
|  | HelpContext | 0 | published |
|  | HelpKeyword | - | published |
|  | HelpType | htContext | published |
|  | HideSelection | false | published |
|  | Hint | - | published |
|  | HotTrack | false | published |
|  | Images | - | published |
|  | Indent | - | published |
|  | Items | - | public |
|  | Left | - | published |
|  | Link | - | public |
|  | Margins | - | published |
|  | MultiSelect | false | published |
|  | MultiSelectStyle | [msControlSelect] | published |
|  | Name | - | published |
|  | NodeMenuFirst | true | published |
|  | OvlImages | - | published |
|  | OvlOffset | 0 | published |
|  | ParentBiDiMode | true | published |
|  | ParentCtl3D | true | published |
|  | ParentColor | false | published |
|  | ParentFont | true | published |
|  | ParentShowHint | true | published |
|  | PopupMenu | - | published |
|  | PopupNode | - | public |
|  | ReadOnly | false | published |
|  | RightClickSelect | false | published |
|  | RowSelect | false | published |
|  | Selected | - | public |
|  | SelectionCount | - | public |
|  | Selections | - | public |
|  | ShowButtons | true | published |
|  | ShowHint | false | published |
|  | ShowLines | true | published |
|  | ShowRoot | false | published |
|  | SortType | stNone | published |
|  | StateImages | - | published |
|  | TabOrder | - | published |
|  | TabStop | true | published |
|  | Tag | 0 | published |
|  | ToolTips | true | published |
|  | Top | - | published |
|  | TopItem | - | published |
|  | Visible | - | published |
|  | Width | - | published |

Рассмотрим более подробно некоторые свойства.

Свойства Align, AlignWithMargins, Ancors, BiDiMode, BorderWidth, Contrstraints, Color, Ctl3D, Cursor, DragCursor, DragKind, DragMode, Enabled, Font, Height, HelpContext, HelpKeyword, HelpType, Hint, Left, Margins, Name, ParentBiDiMode, ParentCtl3D, ParentColor, ParentFont, ParentShowHint, PopupMenu, ShowHint, TabOrder, TabStop, Tag, Top, Visible, Width аналогичны одноименным свойствам стандартных классов-предков.

AutoExpand: показывает, надо ли автоматически раскрывать (и сворачивать) все дочерние узлы при выделении родительского.

BorderStyle: позволяет задавать тип границы компонента.

Canvas: содержит ссылку на «холст» компонента.

ChangeDelay: свойство содержит величину задержки между моментом, когда узел выбран и срабатыванием события OnChange.

Changing: флаг содержит значение «истина», если в настоящий момент осуществляется переход на другой узел.

CollapseDelChild: показывает, удалять ли дочерние узлы при сворачивании узла.

DropTarget: содержит узел, в который была произведена попытка перенести узел при окончании операции DragDrop.

HideSelection: показывает, отображать ли выделенные узлы при изменении фокуса (с компонента TXTreeView на другой компонент).

HotTrack: показывает, выделять ли узлы в момент, когда над ними проводится мышка.

Images: содержит ссылку на коллекцию картинок, используемых узлами.

Indent: определяет размер отступа в пикселях при разворачивании дочернего узла.

Items: содержит ссылку на экземпляр класса TXTreeNodes, который, в свою очередь является проводником между представлением и узлами.

Link: содержит ссылку на экзмепляр класса TTVLink, предназначенного для связи древовидного представления и управляющего им выпадающего списка.

MultiSelect: показывает, можно ли выделить одновременно несколько узлов представления.

MultiSelectStyle: определяет, каким образом работает множественный выбор.

NodeMenuFirst: свойство показывает, располагать ли элементы контекстного меню дерева в начале, а затем распологать элементы контекстного меню самого узла или наоборот ( одному узлу можно назначить несколько контекстных меню, также контекстное меню можно назначить собственно дереву, при построении контекстного меню собираются все пункты меню в одно большое ).

OvlImages: содержит ссылку на список оверлейных изображений.

OvlOffset: содержит смещение индекса картинки в списке оверлейных картинок, то есть физически отображаться будет картинка под номером OvlOffset+ OverlayImage(свойство узла).

PopupNode: содержит ссылку на узел, на котором было вызвано контекстное меню, вычисляется, когда приходит сообщения WM\_CONTEXTMENU, свойство содержит информацию до тех пор, пока светится конекстное меню.

ReadOnly: показывает, можно редактировать дерево, отображаемое компонентом.

RightClickSelect: показывает, считать ли выделенным узел в момент нажатия на него правой клавишей мыши.

RowSelect: показывает, выделять ли всю строку, когда выделяется узел.

Selected: содержит ссылку на выделенный в настоящий момент узел.

SelectionCount: содержит количество выделенных в настоящий момент узлов.

Selections: содержит выделенные в настоящий момент узлы.

ShowButtons: показывает, отображать ли кнопки «+» и «- ».

ShowLines: определяет, отображать ли линии, связывающие дочерние узлы и родительские.

ShowRoot: показывает, соединяются ли линии и кнопки плюс (+) или минус (-) с корневыми элементами в компоненте.

SortType: определяет вид сортировки.

StateImages: содержит ссылку на коллекцию изображений для вывода на экран состояний узлов.

ToolTips: показывает, имеют ли узлы отступы.

TopItem: содержит самый верхний видимый пользователю узел.

Рассмотрим события компонента. Более подробно будем останавливаться на событиях, специфичных для HyTech.

1. OnAdvancedCustomDrawItem(TXTVAdvancedCustomDrawItemEvent). Вызывается на каждом этапе отрисовки узла представления. Использует ряд параметров:
   * + - * Sender (TXTreeView);
         * Node (TXNode) – содержит узел;
         * State (TCustomDrawState) – отражает состояние отрисовки (подробнее см справку по TCustomDrawState);
         * Stage (TCustomDrawStage) – отражает этап отрисовки отрисовки (подробнее см справку по TCustomDrawStage);
         * PaintImages (Boolean) – показывает, нужно ли перерисовать изображения, связанные с компонентом;
         * DefaultDraw (Boolean) – показывает, нужно ли перерисовать текст узла после обработки события.
2. OnChange(TXTVChangedEvent). Вызывается после того, как изменяется выделенный узел. Использует несколько параметров: Sender (TXTreeView) и Node (TXNode) –содержит текущий узел.
3. OnChanging(TXTVChangingEvent). Вызывается перед тем, как изменяется выделенный узел. Использует ряд параметров:
   * + - * Sender (TXTreeView);
         * Node (TXNode) – содержит текущий узел;
         * AllowChange (Boolean) – показывает, можно ли изменять узел.
4. OnClick (TNotifyEvent). Вызывается при осуществлении нажатия на компоненте.
5. OnCollapsed (TXTVExpandedEvent). Вызывается после того, как сворачивается узел. Использует несколько параметров: Sender (TXTreeView) и Node (TXNode) – содержит текущий узел.
6. OnCollapsing (TXTVCollapsingEvent).Вызывается перед тем, как сворачивается узел. Использует ряд параметров:
   * + - * Sender (TXTreeView);
         * Node (TXNode) – содержит текущий узел;
         * AllowCollapse (Boolean) – показывает, можно ли сворачивать узел.
7. OnCompare (TXTVCompareEvent). Вызывается, когда появляется необходимость сравнить два узла в ходе сортировки. Использует ряд параметров:
   * + - * Sender (TXTreeView);
         * Node1 (TXNode) – содержит первый узел;
         * Node2 (TXNode) – содержит второй узел;
         * Data (Integer) – пользовательский параметр;
         * Compare (Integer)- содержит результат сравнения, больше 0, если первый узел «больше» второго, меньше 0, если «меньше» и равен 0, если узлы равны;
8. OnCustomDraw (TXTVCustomDrawEvent). Вызывается непосредственно перед отрисовкой представления. Использует ряд параметров:
   * + - * Sender (TXTreeView);
         * ARect (TRect)- определяет границы компонента TXTreeView;
         * DefaultDraw(Boolean)– показывает, нужно ли перерисовать текст узла после обработки события.
9. OnCustomDrawItem (TXTVCustomDrawItemEvent). Вызывается непосредственно перед отрисовкой узла представления. Использует ряд параметров:
   * + - * Sender (TXTreeView);
         * Node (TXNode) – содержит узел;
         * State (TCustomDrawState) – отражает состояние отрисовки (подробнее см справку по TCustomDrawState);
         * DefaultDraw (Boolean) – показывает, нужно ли перерисовать текст узла после обработки события.
10. OnDblClick (TNotifyEvent). Вызывается при двойном нажатии на дереве.
11. OnDeletion (TXTVExpandedEvent). Вызывается после удаления узла. Использует несколько параметров: Sender (TXTreeView) и Node (TXNode) – содержит текущий узел.
12. OnDragDrop (TDragDropEvent). Вызывается при завершении операции DragDrop.
13. OnDragOver (TDragDropEvent). Вызывается, когда над узлами выполняется операция DragDrop.
14. OnEdited (TXTVEditedEvent ).Вызывается после того, как узел был отредактирован. Использует ряд параметров:
    * + - * Sender (TXTreeView);
          * Node (TXNode) – содержит текущий узел;
          * OldText (string) – содержит предыдущее значение;
          * Canceled (boolean) - показывает, отменено ли редактирование.
15. OnEditing (TXTVEditingEvent). Вызывается перед редактированием узла. Использует ряд параметров:
    * + - * Sender (TXTreeView);
          * Node (TXNode) – содержит редактируемый узел;
          * AllowEdit (Boolean) – показывает, разрешено ли редактирование.
16. OnEndDock (TEndDockEvent). Вызывается при завершении при окончании стыковки компонента.
17. OnEndDrag (TEndDragEvent). Вызывается при завершении операции DragDrop.
18. OnEnter (TNotifyEvent). Вызывается, когда фокус устанавливается на компонент.
19. OnExit (TNotifyEvent). Вызывается при потере фокуса компонентом.
20. OnExpanded (TXTVExpandedEvent). Вызывается после того, как были открыты дочерние узлы текущего узла. Использует несколько параметров: Sender (TXTreeView) и Node (TXNode) – содержит текущий узел.
21. OnExpanding (TXTVExpandingEvent). Вызывается перед тем, как были открыты дочерние узлы текущего узла. Использует ряд параметров:
    * + - * Sender (TXTreeView);
          * Node (TXNode) – содержит текущий узел;
          * AllowExpansion (Boolean) – показывает, можно ли открывать дочерние узлы.
22. OnGetExpandedIndex (TXTVExpandedEvent). Вызывается, когда представление запрашивает свойство OverlayIndex узла дерева. Использует несколько параметров: Sender (TXTreeView) и Node (TXNode) – содержит текущий узел.
23. OnGetImageIndex (TXTVExpandedEvent) Вызывается, когда представление запрашивает свойство ImageIndex узла дерева. Использует несколько параметров: Sender: (TXTreeView) и Node (TXNode) – содержит текущий узел.
24. OnGetSelectedIndex (TXTVExpandedEvent). Вызывается, когда представление запрашивает свойство SelectedIndex узла дерева. Использует несколько параметров: Sender (TXTreeView) и Node (TXNode) – содержит текущий узел.
25. OnInvalidating (TNotifyEvent). Вызывается при пересоздании узла.
26. OnKeyDown (TKeyEvent). Вызывается при нажимании на кнопку на клавиатуре.
27. OnKeyPress (TKeyPressEvent). Вызывается при нажатии на кнопку на клавиатуре.
28. OnKeyUp (TKeyEvent). Вызывается при отпускании кнопки на клавиатуре.
29. OnMouseDown (TMouseEvent). Вызывается при нажимании на кнопку на мыши.
30. OnMouseMove (TMouseMoveEvent). Вызывается при движении мышью над компонентом.
31. OnMouseUp (TMouseEvent). Вызывается при отпускании кнопки на мыши.
32. OnStartDock (TStartDockEvent). Вызывается при начинании стыковки.
33. OnStartDrag (TStartDragEvent). Вызывается при начинании операции DragDrop.

Методы компонента представлены в таблице .

Таб. Основные методы компонента TXTreeView

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | function AlphaSort: Boolean; | Позволяет отсортировать узлы в алфавитном порядке. Возвращает true в случае успешной сортировки. |
|  | procedure Change(Node: TXNode); dynamic; | Вызывается автоматически при смене выбранного узла. В параметре передается новый узел. |
|  | procedure ClearSelection(KeepPrimary: Boolean = False); virtual; | Позволяет отменить выделение узлов в дереве, параметр показывает, сохранять ли выделение узла, выделенного первым. |
|  | function CustomSort(SortProc: TTVCompare; Data: Longint): Boolean; | Позволяет отсортировать узлы в дереве согласно SortProc процедуре сортировки, Data предназначен для передачи пользовательского параметра в функцию сортировки. |
|  | procedure Deselect(Node: TXNode); virtual; | Позволяет отменить выделение узла Node. |
|  | function FillFirst(NodeClass: TXNodeClass; const Names: string; const Values: array of Variant): TXNode; overload; | Позволяет создать (заполнить) корневой узел, при этом указывается тип узла, который надо создать, имена свойств, значения свойств. Возвращает созданный узел. |
|  | function FillFirst(NodeClass: TXNodeClass; Source: TPersistent; const Names: string; const Values: array of Variant): TXNode; overload; | Позволяет создать (заполнить) корневой узел, при этом указывается тип узла, который надо создать, имена свойств, а также источник source , в соответствии с полями которого нужно создавать узел. Возвращает созданный узел. |
|  | function FindNextToSelect: TXNode; virtual; | Позволяет получить следующий, после выделенного узла, узел. |
|  | procedure FullCollapse; | Позволяет свернуть все узлы. |
|  | procedure FullExpand; | Позволяет раскрыть все узлы. |
|  | function GetHitTestInfoAt(X, Y: Integer): THitTests; | Возвращает положение точки с координатами X, Y относительно древовидного представления. Возвращает одно из значений типа THitTests. |
|  | function GetNodeAt(X, Y: Integer): TXNode; | Возвращает узел типа TXNode по координатам. |
|  | function GetSelections(AList: TList): TXNode; | Позволяет получить список выделенных записей. Возвращает значение, содержащееся в свойстве Selected. |
|  | function GetTextPath: string; | Позволяет получить путь к текущему узлу в текстовом виде. |
|  | function IsEditing: Boolean; | Показывает, имеет ли место редактирование текста узла в настоящий момент. |
|  | procedure LoadFromFile(const FileName: string); | Позволяет создать узлы в соответствии с информацией из файла FileName. |
|  | procedure LoadFromStream(Stream: TStream); | Позволяет создать узлы в соответствии с информацией из потока Stream. |
|  | procedure Lock; | Позволяет увеличить на 1 количество блокировок представления. |
|  | procedure LockUpdate; | Позволяет увеличить на 1 количество блокировок представления и выполнить процедуру BeginUpdate, что предотвращает перерисовку компонента в ходе серии изменений. |
|  | procedure RefreshNode(Node: TXNode; ExpandIt: boolean); | Позволяет пересоздать узел Node. Если второй параметр true, узел разворачивается. |
|  | procedure SaveToFile(const FileName: string); | Позволяет сохранить информацию об узлах в файл FileName. |
|  | procedure SaveToStream(Stream: TStream); | Позволяет сохранить информацию об узлах в поток Stream. |
|  | procedure Select(Node: TXNode; ShiftState: TShiftState = []); overload; virtual; | Позволяет выделить узел Node так, как будто на узел нажали мышкой, во втором параметре при этом можно передать состояние служебных клавиш в момент «нажатия». |
|  | procedure Select(const Nodes: array of TXNode); overload; virtual; | Позволяет выделить несколько узлов Nodes. |
|  | procedure Select(Nodes: TList); overload; virtual; | Позволяет выделить узлы согласно списку Nodes. |
|  | procedure Subselect(Node: TXNode; Validate: Boolean = False); virtual; | Позволяет изменить выделение узла Node (выделить, если не был выделен, снять выделение, если был). Предварительно необходимо разрешить множественное выделение. Если второй параметр true, в методе также проверяется актуальность информации о выделении в дереве, отражает ли список выделенных узлов действительно выделенные узлы. |
|  | procedure Unlock; | Позволяет уменьшить на 1 количество блокировок представления. |
|  | procedure UnlockUpdate(CallChange: boolean); | Позволяет уменьшить на 1 количество блокировок представления и выполнить процедуру EndUpdate, что предотвращает перерисовку компонента в ходе серии изменений. Если параметр CallChange true, в конце вызывается метод Change. |

# **THTree**

Компонент используется вместо TXTreeView, однако не является визуальным. Кеширует в узлах информацию; при подключении нескольких THTreeView (наследник стандартного TTreeView, предназначенный для работы с THTree), обеспечивает их взаимодействие и идентичность отображаемой информации. Предназначен для хранения древовидной структуры узлов. Используется в паре с THTReeView(для отображения узлов). Узлами THTree могут быть различного типа наследники THNode. Иерархия классов представлена на .



Рис. Иерархия классов

Свойства компонента приведены в .

Таб. Свойства компонента THTree

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | DragNode | - | public |
|  | Items | - | public |

Рассмотрим более подробно свойства компонента.

DragNode: содержит ссылку на узел, который перетаскивается.

Items: позволяет обратиться к узлу.

Основные методы компонента представлены в таблице .

Таб. Основные методы компонента THTree

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | function FillFirst(NeedClass: THNodeClass; const PropNames: string; const PropValues: array of Variant): THNode; overload; | Позволяет создать (заполнить) корневой узел, при этом указывается тип узла NeedClass, который надо создать, имена свойств PropNames, значения свойств PropValues, возвращается ссылка на класс. |
|  | function FillFirst(NeedClass: THNodeClass; Source: TPersistent; const PropNames: string; const PropValues: array of Variant): THNode; overload; | Позволяет создать (заполнить) корневой узел и вернуть ссылку на него, при этом указывается тип узла NeedClass, который надо создать, имена свойств PropNames, значения свойств PropValues, а также источник source , в соответствии с полями которого нужно создавать узел. |
|  | function GetFirstNode: THNode; | Позволяет получить корневой узел. |
|  | procedure HandleException(ExceptObject: TObject); virtual; | Позволяет обработать исключительную ситуацию ExceptObject. |
|  | function IsClient(Client: Pointer): boolean; | Позволяет проверить, является ли CLient получателем уведомительных сообщений. |
|  | procedure RegisterNotifier(Notifier: IHTNotifier); | Позволяет зарегистрировать получателя сообщений Notifier. |
|  | procedure UnregisterNotifier(Notifier: IHTNotifier); | Позволяет отменить регистраицю получателя сообщений Notifier. |

# **THTreeView**

Иерархия классов представлена на рисунке .



Рис. Иерархия классов

Свойства компонента, отличные от свойств предка, представлены в .

Таб. Свойства компонента THTreeView

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Название | Значение по умолчанию | Тип |
|  | HNodeOver | - | public |
|  | HRoot | - | public |
|  | Locked | - | public |
|  | Tree | - | public |
|  | PopupNode | - | public |

Остановимся более подробно на свойствах компонента.

HNodeOver : содержит узел, надо которым осуществляется DragDrop.

HRoot : содержит корневой узел представления.

Locked: содержит количество блокировок компонента.

Tree: содержит ссылку на соответствующий компонент типа THTree.

PopupNode: содержит узел, на котором вызвано контекстное меню.

Рассмотрим события компонента.

1. OnNotify (THTreeNotifierEvent). Предоставляет возможность дописать специфическую обработку событий. Использует ряд параметров:
   * + - * Sender (TObject);
         * HNode (IHNode) – содержит узел;
         * Events (THNEvents) – содержит набор типов событий, для которых дописывается обработка.

Методы компонента представлены в таблице .

Таб. Основные методы компонента THTreeView

|  |  |  |
| --- | --- | --- |
| №п/п | Название | Описание |
|  | procedure Clear; | Позволяет очистить узлы представления и отменить регистрацию представления на события соответствующего компонента THTree |
|  | procedure DefaultNotify(HNode: IHNode; Events: THNEvents); | Является обработчиком по умолчанию. При необходимости применить специфическую обработку событий используется OnNotify, для всех остальных событий используется этот метод. |
|  | procedure Lock; | Позволяет увеличить на 1 количество блокировок представления |
|  | function NodeByHNode(out Node: TTreeNode; HNode: IHNode): boolean; | Позволяет найти узел компонента по его hnode (идентификатору узла) в представлении. Узел возвращается в параметре Node. Метод возвращает true, если узел найден. |
|  | function NodeByHPath(out Node: TTreeNode; Path: THPath): boolean; | Позволяет найти в дереве узел, соответствующий пути Path, если узел не найден, Node содержит последний соответствующий пути узел. Метод возвращает true, если узел найден. |
|  | function PopupHNode(NeedClass: THNodeClass; out HNode): boolean; | Позволяет по идентификатору узла, на котором в древовидном представлении вызвано контекстное меню представления, найти узел компонента, при этом класс найденного узла должен быть не младше, чем NeedClass, параметр HNode сделан безтиповым, чтобы при вызове метода можно было передавать переменную уже нужного типа, не выполняя преобразований. Метод возвращает true, если узел найден. |
|  | procedure RefreshView; virtual; | Позволяет обновить все узлы соответствующего древовиного представления |
|  | function SelectedHNode(NeedClass: THNodeClass; out HNode): boolean; | Позволяет по идентификатору выделенного в представлении узла найти узел компонента Tree, при этом класс найденного узла должен быть не младше, чем NeedClass, параметр HNode сделан безтиповым, чтобы при вызове метода можно было передавать переменную уже нужного типа, не выполняя преобразований |
|  | function SetHRoot(const Value: IHNode): TTreeNode; | Позволяет очистить представление, создать корневой узел, поставить ему в соответствие IHNode из текущего компонента. |
|  | procedure Unlock(DoRepaint: boolean = true); | Позволяет уменьшить на 1 количество блокировок представления. Параметр показывает, требуется ли перерисовка. |

# Приложение



Рис. Общая иерархия классов