С. В. АНИКЕЕВ, А. В. МАРКИН

РАЗРАБОТКА ПРИЛОЖЕНИЙ БАЗ ДАННЫХ В DELPHI

Самоучитель

МОСКВА • ДИАЛОГ-МИФИ • 2013

УДК 004.6
ББК 32.973.26-018.2
А-67

Рецензент
кафедра автоматизированных систем управления
Рязанского государственного радиотехнического университета
зав. кафедрой д-р техн. наук., проф. Г. И. Нечаев

Аникеев С. В., Маркин А. В.
А-67 Разработка приложений баз данных в Delphi: Самоучитель. –
М.: Диалог-МИФИ, 2013. – 160 с. Табл. 12. Ил. 44. Библиогр.: 16 назв.

ISBN ISBN 978-5-86404-243-4

Представляет собой практическое руководство по разработке приложений БД в среде Delphi. Приведено описание общей структуры приложений БД, обзор различных технологий доступа к серверам БД, рассмотрены компоненты, отвечающие за доступ к данным, и компоненты, отвечающие за отображение и редактирование данных.

Весь теоретический материал самоучителя проиллюстрирован множеством примеров демонстрации отдельных свойств и методов компонентов в среде CodeGear Delphi 2007 на учебной базе данных, являющейся упрощенной моделью БД реальной информационной системы. Также приведен пример разработки демонстрационного приложения для работы с базой данных.

Предназначен для студентов вузов, обучающихся по направлениям специалитета, бакалавриата и магистратуры, связанным с разработкой, эксплуатацией и сопровождением информационных систем, желающих самостоятельно изучать основы разработки приложений БД в Delphi.

Самоучитель
Аникеев Сергей Владимирович, Маркин Александр Васильевич
Разработка приложений баз данных в Delphi

Редактор О. А. Голубев. Макет Н. В. Дмитриевой
Подписано в печать 5.03.2013.
Формат 60x84/16. Бум. офс. Печать офс. Гарнитура Таймс.
Усл. печ. л. 9,3. Уч.-изд. л. 6,95. Тираж 500 экз. Заказ

ООО «Издательство Диалог-МИФИ”
115409, Москва, ул. Москворечье, 31, корп. 2. Тел.: 8-905-769-16-61
Http: www.dialog-mifi.ru. E-mail: zakaz@dialog-mifi.ru

Отпечатано ООО «ИНСОФТ»
117105, г. Москва, Варшавское ш., д. 37А

ISBN 978-5-86404-243-4

Оглавление

Предисловие

Настоящее издание написано на основе многолетнего опыта авторов, приобретенного в результате разработки, внедрения, эксплуатации и сопровождения информационных систем для подразделений ОАО «Газпром», жилищно-коммунальных и других организаций, а также чтения лекций, проведения лабораторных практикумов, руководства курсовым и дипломным проектированием в Рязанском государственном радиотехническом университете.

Данное пособие является продолжением серии учебных пособий [1, 2, 3] по работе с реляционными базами данных (БД) и содержит сведения об особенностях разработки приложений БД в среде Delphi.

Для работы с пособием желательно знание теории реляционных БД и знакомство с основными принципами построения SQL-запросов. SQL-запросы в пособии подробно не поясняются, а делается акцент на особенности их использования в Delphi. При необходимости информацию по построению запросов и программированию на SQL можно найти в учебном пособии [1]. Кроме того, на сайте http://rgrty.ru размещен дистанционный авторский курс, посвященный основам построения запросов и программированию на SQL, а на сайте http://abonentplus.ru/sqltest/ можно проверить полученные теоретические знания. Также предполагается, что читатель имеет начальный уровень знаний по разработке в интегрированной среде Delphi. Множество элементарных действий, не связанных непосредственно с разработкой приложений БД в среде Delphi, также подробно не поясняется.

Самоучитель адресован в первую очередь студентам и начинающим разработчикам приложений реляционных БД в Delphi, но, возможно, будет интересным и полезным более широкому кругу читателей.

Все примеры выполнены в среде CodeGear Delphi 2007, запросы выполнены на SQL системы управления БД (СУБД) Firebird 2.5 (последняя версия).

Firebird (FirebirdSQL) – компактная, кроссплатформенная, свободная СУБД, работающая на Linux, Microsoft Windows и разнообразых Unix платформах с 32- и 64-битовой архитектурой. В настоящем пособии эта СУБД выбрана прежде всего потому, что ее последние версии без всяких материальных затрат и особых трудностей можно получить с сайта http://www.firebirdsql.org, установить на личном компьютере и использовать для самостоятельного изучения. Несмотря на то, что примеры разработаны для реализации FirebirdSQL, они также, за некоторым исключением, будут работать в любой из современных промышленных реляционных СУБД в архитектуре «клиент-сервер».

Авторы выражают искреннюю благодарность Дубининой В. Л. за содействие и помощь в подготовке настоящего пособия. Авторы глубоко признательны рецензентам, чьи ценные замечания позволили улучшить качество пособия.

Введение

Настоящий самоучитель посвящен рассмотрению технологий разработки приложений для работы с БД в среде Delphi.

Система визуального программирования Delphi обладает большой популярностью среди широкого круга пользователей: от неспециалистов до системных программистов, занимающихся разработкой сложных приложений и информационных систем [4].

Delphi позволяет быстро и удобно разрабатывать эффективные приложения, включая приложения для работы с БД. Система имеет развитые возможности по созданию пользовательского интерфейса, широкий набор функций, методов и свойств для решения прикладных расчетно-вычислительных задач. В системе имеются развитые средства отладки, облегчающие разработку приложений.

Традиционно Delphi относят к системам быстрой разработки приложений. Вместе с тем эта система обладает некоторыми возможностями современных СУБД: позволяет визуально подготавливать запросы к БД, непосредственно писать SQL-запросы к БД и т. д.

Delphi позволяет создавать приложения для работы с локальными и удаленными БД, включая публикацию БД в Интернете. Применительно к работе с БД Delphi обеспечивает широкий набор инструментальных средств, поддерживает современные технологии, в том числе многоуровневую архитектуру "клиент-сервер".

В рамках настоящего пособия описывается структура приложений БД, технологии доступа к серверам БД, основные понятия, компоненты, свойства, методы и события для работы с БД. Основное внимание уделено технологии IBX, позволяющей работать с БД Interbase/Firebird.

В гл. 1 рассматриваются общие подходы к разработке приложений БД в Delphi, которые не зависят от выбранной технологии доступа к данным. Приведена схема взаимодействия компонентов в приложении БД.

Гл. 2 поясняет, как получить доступ к различным СУБД, управляющим данными различных БД. Приводится обзор различных технологий доступа к данным, подробное описание компонентов технологии IBX, особенности использования компонентов доступа к данным в Delphi. Приведена схема взаимосвязи приложения и таблиц БД при использовании компонентов IBX и модуля данных. Описаны различные особенности наборов данных.

В гл. 3 кратко рассмотрен компонент-источник данных TDataSource, который используется для связи между наборами данных и визуальными компонентами.

В гл. 4 подробно описываются визуальные компоненты, отвечающие за отображение и редактирование данных в приложениях. Представлена схема,

отражающая классификацию визуальных компонентов. Подробно, на примерах, рассмотрена разработка приложения с использованием различных типов визуальных компонентов.

В гл. 5 приведен пример разработки демонстрационного приложения на учебной базе данных.

Материал всех глав сопровождается большим количеством как простых, так и достаточно сложных примеров на учебной БД, являющейся очень сокращенным вариантом БД расчетно-аналитической информационной системы «Абонент», одними из разработчиков которой являются авторы настоящего пособия. Система «Абонент» достаточно эффективно используется в нескольких регионах Российской Федерации для информационного обеспечения деятельности газораспределительных организаций, региональных компаний по реализации газа и других жилищно-коммунальных предприятий [5].

В пособии приводятся примеры реально работающих процедур, которые могут быть использованы читателями при разработке своих собственных приложений.

При использовании программного кода знаки переноса недопустимы. Строчки, которые не умещаются целиком, можно разбивать по словам (переносить на другую строку после точки или перед скобкой), такой код работать будет.

Пример разработки клиентского приложения выполнен в CodeGear Delphi 2007 с использованием компонентов IBX. Запросы, используемые для получения данных, выполнены на SQL СУБД Firebird.

В приложении А приведено описание учебной БД, в приложении Б – скрипт для ее создания, в приложении В – коды хранимых процедур, использованных при разработке демонстрационного приложения, в приложении Г – код программы.

1. Структура приложений БД

Приложение БД предназначено для взаимодействия с некоторым источником данных – базой данных. Взаимодействие подразумевает получение данных, их представление в определенном формате для просмотра пользователем, редактирование в соответствии с реализованными в программе бизнес-алгоритмами и возврат обработанных данных обратно в БД [6].

Базы данных обслуживаются СУБД. СУБД делятся на локальные (преимущественно однопользовательские), предназначенные для настольных приложений, и серверные (сетевые, часто удаленные, многопользовательские), функционирующие на выделенных компьютерах – серверах.

Тем не менее, несмотря на разнообразие реализаций, общая архитектура приложения БД остается неизменной.

В Delphi реализовано достаточно большое число разнообразных технологий доступа к данным. Тем не менее, последовательность операций при конструировании приложений БД остается почти одинаковой, и в работе используются по сути одни и те же компоненты, доработанные для применения с той или иной технологией доступа к данным.

Набор базовых компонентов и способов разработки является единой основой, на которой базируются технологии доступа к данным. Это позволило унифицировать процесс разработки приложений БД.

В настоящей главе рассматриваются общие подходы к разработке приложений БД в Delphi, базовые классы и механизмы, которые не зависят от выбранной технологии доступа к данным. При разнообразии способов реализации и обилии технических деталей общая архитектура приложений БД в Delphi реализует следующую общую схему.

Приложение БД включает в себя:

- механизм получения и отправки данных;
- механизм внутреннего представления данных в том или ином виде;
- пользовательский интерфейс для отображения и редактирования данных;
- бизнес-логику для обработки данных.

Механизм получения и отправки данных обеспечивает соединение с источником данных (часто опосредованно). Источник данных представляет собой хранилище данных (саму базу данных) и СУБД, управляющую данными, обеспечивающую целостность и непротиворечивость данных.

Необходимо знать, куда обращаться и какой протокол обмена использовать для обеспечения двунаправленного потока данных.

Между приложением и собственно базой данных находится специальное программное обеспечение (ПО), связывающее программу и источник данных

и управляющее процессом обмена данными. Это ПО может быть реализовано самыми разнообразными способами, в зависимости от объема БД, решаемых системой задач, числа пользователей, способами соединения приложения и БД. Промежуточное ПО может быть реализовано как окружение приложения, без которого оно вообще не будет работать, как набор драйверов и динамических библиотек, к которым обращается приложение, может быть интегрировано в само приложение. Наконец, это может быть отдельный удаленный сервер, обслуживающий тысячи приложений.

Механизм внутреннего представления данных является ядром приложения БД. Он обеспечивает хранение полученных данных в приложении и предоставляет их по запросу других частей приложения.

Пользовательский интерфейс обеспечивает просмотр и редактирование данных, а также управление данными и приложением в целом.

Бизнес-логика приложения представляет собой набор реализованных в программе алгоритмов обработки данных.

Средства Delphi, предназначенные для работы с БД, можно разделить на два вида [4]: ***инструменты*** и ***компоненты***.

К *инструментам* относятся специальные программы и пакеты, обеспечивающие обслуживание БД вне разрабатываемых приложений (например, SQL Builder, SQL Monitor и многие другие). Инструменты Delphi для работы с БД подробно не рассматриваются в рамках настоящего пособия.

Компоненты предназначены для создания приложений, осуществляющих операции с БД. Среди них можно выделить:

- *Визуальные компоненты* (компоненты отображения и редактирования данных). Используются для создания интерфейсной части приложения.
- *Невизуальные компоненты.* Данные компоненты отображаются во время разработки приложения, но не видны на этапе выполнения программы. Предназначены для организации доступа к данным, содержащимся в таблицах БД. Невизуальные компоненты представляют собой промежуточное звено между данными таблиц БД и визуальными компонентами.

Невизуальные компоненты представлены *компонентами доступа к данным*, отвечающими за передачу данных из БД в приложение, и специальным *компонентом-источником данных TDataSource*, отвечающим за последующую передачу этих данных в визуальные компоненты.

Существует несколько наборов *компонентов доступа к данным*, которые используют разные технологии (DBExpress, IBX и др.) и отличаются по своим возможностям. Каждый набор хорошо подходит для решения определенного круга задач, что дает громадные преимущества, а главное – свободу выбора [7].

Таким образом, базовый механизм доступа к данным создается ***триадой компонентов*** [6]:

- компоненты доступа к данным;
- компоненты TDataSource – источники данных;
- визуальные компоненты отображения данных.

Каждое приложение, использующее БД, обычно имеет по одному компоненту из этих трех типов [8]. Кратко рассмотрим особенности различных типов компонентов (подробнее компоненты будут описаны в гл. 2–4), чтобы дать общее представление о структуре взаимодействия данных компонентов в приложении БД.

База данных, с которой работает приложение Delphi, физически располагается на диске и содержит определенное количество таблиц. Совокупность записей, выделенных по определенным правилам из одной или нескольких таблиц БД, называется ***набором данных***. Набор данных формируется при помощи SQL-запроса на выборку данных (при необходимости информацию по построению запросов и программированию на SQL можно найти в учебном пособии [1]). Например, применительно к учебной БД (описание приведено в приложении А), набором данных может являться выборка из таблицы Abonent абонентов, проживающих на определенной улице. Фактически, набор данных – это тоже таблица, но логическая.

В контексте разработки приложений БД *набор данных* представляет собой совокупность записей, переданных из БД в приложение для просмотра и редактирования. В приложении БД каждый набор данных представлен специальным *компонентом доступа к данным*. Например, в приложении представлять (инкапсулировать) выборку абонентов, проживающих на конкретной улице, может компонент TIBDataSet либо компонент TIBQuery технологии IBX. В VCL (Visual Component Library – библиотека визуальных компонентов) Delphi реализован набор базовых классов (потомки класса TDataSet), поддерживающих функциональность наборов данных, и практически идентичные по составу наборы дочерних компонентов для различных технологий доступа к данным. Каждый компонент-потомок TDataSet представляет собой "образ" набора данных в приложении, в связи с этим далее по тексту самоучителя понятия набора данных и компонента, инкапсулирующего этот набор данных, в большинстве случаев отождествляются. Общее число компонентов-наборов данных в приложении не ограничено.

Для обеспечения связи набора данных с визуальными компонентами отображения данных используется специальный компонент *TDataSource* [2]. Данный компонент является общим для всех технологий доступа [7]. Его роль заключается в управлении потоками данных между набором данных и связанными с ним компонентами отображения данных. Этот компонент об-

ращается к функциям соответствующей технологии доступа к данным для выполнения различных операций. С каждым компонентом доступа к данным может быть связан как минимум один компонент TDataSource. В его обязанности входит соединение набора данных с визуальными компонентами отображения данных. Компонент TDataSource обеспечивает передачу в эти компоненты текущих значений полей из набора данных и возврат в него сделанных изменений.

Еще одна функция компонента TDataSource заключается в синхронизации поведения компонентов отображения данных с состоянием набора данных. Например, если набор данных не активен, то компонент TDataSource обеспечивает удаление данных из компонентов отображения данных и их перевод в неактивное состояние. Или, если набор данных работает в режиме "только для чтения", то компонент TDataSource обязан передать в компоненты отображения данных запрещение на изменение данных.

С одним компонентом TDataSource могут быть связаны один или несколько *визуальных компонентов* отображения данных [6]. В большинстве своем такие компоненты представляют собой модификации стандартных элементов управления, приспособленных для работы с набором данных. Например, в приложении для отображения абонентов, проживающих на определенной улице, можно использовать компонент TDBGrid, а для отображения наименования улицы в отдельном поле – компонент TDBEdit.

При открытии набора данных компонент доступа к данным обеспечивает передачу записей из требуемых таблиц БД в приложение. Курсор набора данных устанавливается на первую запись. Компонент-источник TDataSource организует передачу в компоненты отображения данных значений необходимых полей из текущей записи. При перемещении по записям набора данных текущие значения полей в компонентах отображения данных автоматически обновляются, что также обеспечивается компонентом TDataSource.

Пользователь при помощи компонентов отображения данных может не только просматривать, но и редактировать данные. Измененные значения сразу же передаются из визуального компонента в набор данных при помощи компонента TDataSource. Затем изменения могут быть переданы в базу данных или отменены.

Схема взаимодействия описанных компонентов в приложении БД приведена на рис. 1.1.

Таким образом, мы рассмотрели общую структуру и особенности взаимодействия различных компонентов в приложениях БД.

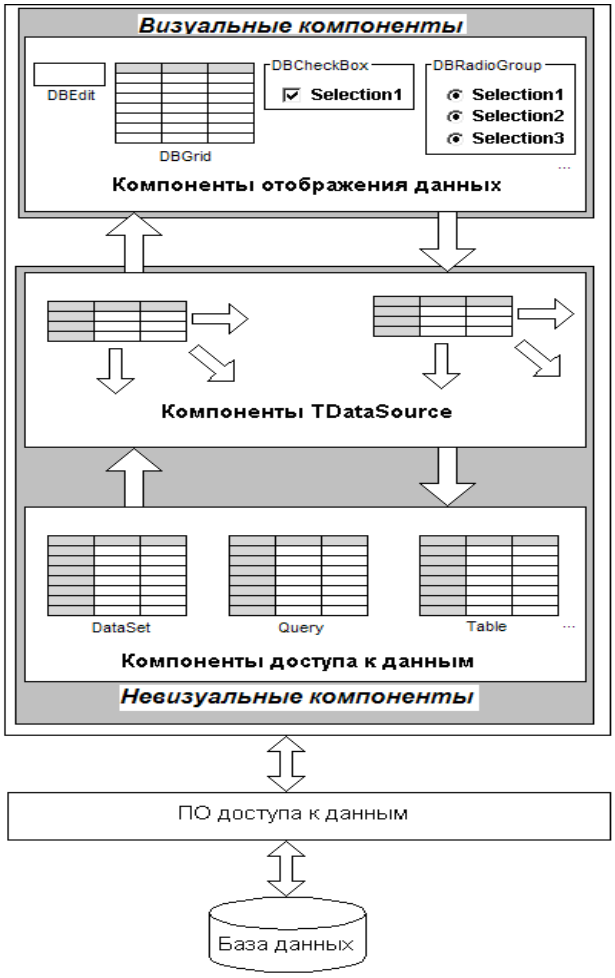


Рис. 1.1. *Механизм доступа к данным приложения БД*

Далее будут более подробно рассмотрены различные технологии доступа к серверам БД, наборы данных, компонент-источник данных, визуальные компоненты, а также пример разработки приложения БД.

Контрольные вопросы

1. Что представляет собой приложение БД?
2. Для каких БД можно создавать приложения в среде Delphi?
3. Какие механизмы включает приложение БД?
4. Что представляет собой специальное программное обеспечение, которое находится между приложением и базой данных?
5. Что такое бизнес-логика приложения?
6. На какие виды можно разделить средства Delphi, предназначенные для работы с БД? Кратко охарактеризуйте каждый вид средств.
7. На какие группы разделяются компоненты, предназначенные для создания приложений БД?
8. Кратко опишите особенности базового механизма доступа к данным в Delphi.
9. Что представляет собой набор данных? Приведите примеры наборов данных в широком смысле и применительно к приложению БД.
10. Нарисуйте схему взаимодействия различных компонентов в приложении БД.

2. Доступ к серверам БД из приложений Delphi

В гл. 1 кратко описаны механизмы и компоненты, входящие в состав любого приложения БД. В настоящей главе рассматривается, как получить доступ к различным СУБД, управляющим данными различных БД.

За соединение с источником данных (базой данных) отвечает *механизм получения и отправки данных*, реализованный с помощью компонентов доступа к данным.

В Delphi используются различные технологии доступа к данным и соответствующие им различные библиотеки компонентов доступа. В любом случае, какая бы технология доступа не использовалась, организуется доступ приложения к данным таким образом, чтобы полностью отстранить разработчика от специфики обслуживания конкретной БД. Любое приложение БД должно «уметь» выполнять как минимум две операции [6].

Во-первых, иметь информацию о местонахождении БД, подключаться к ней и считывать имеющиеся в таблицах данные.

Во-вторых, обеспечивать представление и использование полученных данных. Множество записей данных, переданных в приложение в результате в результате активизации компонента доступа к данным, называется набором данных.

В настоящей главе рассматриваются различные технологии доступа к данным, особенности использования компонентов доступа к данным в Delphi и различные особенности наборов данных.

2.1. Технологии доступа к данным

Технологии доступа к данным отвечают за взаимодействие между приложением и базой данных. В данном разделе приведен обзор различных технологий доступа к данным, используемых в Delphi, и подробное описание компонентов технологии IBX.

2.1.1. Обзор технологий доступа

В Delphi реализовано большое число разнообразных технологий доступа к данным. В зависимости от технологии используется подключение к БД через *ODBC* (Open Database Connectivity, интерфейс кратко рассмотрен в учебном пособии [3]) и через компоненты, работающие с СУБД напрямую (что более эффективно).

Компоненты каждой технологии доступа расположены в Delphi на отдельной одноименной вкладке.

1. ***Технология BDE*** (Borland Database Engine) – технология доступа к базам данных, разработанная фирмой Borland. BDE представляет собой совокупность динамических библиотек и драйверов, обеспечивающих доступ к данным. Механизм BDE до 7-й версии системы Delphi получил самое широкое распространение ввиду широкого спектра предоставляемых им возможностей. Эта технология устарела, но поставляется для учета совместимости со старыми версиями, так как накоплено большое количество приложений с использованием этого подхода. Хорошо работает со старыми типами БД, например, такими как Paradox и dBase [4, 7].

2. ***Технология dbExpress*** – это новая технология доступа к данным фирмы Borland. dbExpress обеспечивает быстрый доступ к информации в базах данных с помощью набора драйверов [4]. Она отличается большей гибкостью и хорошо подходит для программирования клиент-серверных приложений, использующих БД. Компоненты с одноименной вкладки желательно использовать с БД, построенными по серверной технологии, например, Oracle, DB2 или MySQL [7].

3. ***Технология ADO*** (ActiveX Data Objects) – технология доступа к данным, разработанная корпорацией Microsoft. Осуществляет доступ к информации с помощью OLE DB (Object Linking and Embedding Data Base – связывание и внедрение объектов БД). Эта технология основана на стандартных интерфейсах COM, являющихся системным механизмом Windows. Это позволяет удобно распространять приложения БД без вспомогательных библиотек [4]. Данную библиотеку использовать желательно с БД Microsoft, а именно MS Access или MS SQL Server. Ее также можно использовать, если есть специфичный сервер БД, который может работать только через ODBC [7].

4. ***Технология IBX*** реализует непосредственный доступ к базам данных InterBase [4].

Вышеперечисленные технологии и компоненты доступа являются стандартными, поставляемыми с Delphi.

Следует сказать, что, помимо встроенных, существуют также и сторонние наборы компонентов доступа. Для БД InterBase и Firebird таким набором компонентов является FibPlus фирмы Devrace [9]. Библиотека компонентов FibPlus – платная альтернатива IBX, имеющая расширенные редакторы свойств и более высокую функциональность [10]. FibPlus заявляется, как обладающая многими преимуществами по сравнению с другими компонентами доступа к данным, однако, так как данная библиотека компонентов является

платной, в рамках настоящего пособия мы не будем рассматривать компоненты FibPlus. Всю информацию о FIBPIus можно найти на сайте [9].

В рамках настоящего пособия подробно рассматривается технология IBX, используемая для БД Interbase и Firebird.

Все примеры пособия и демонстрационное приложение разработаны с использованием компонентов библиотеки IBX.

2.1.2. Компоненты доступа IBX

Множество приложений на Delphi для работы с БД используют BDE. Однако исторически BDE ориентирован на максимальное упрощение работы с SQL-серверами для тех разработчиков, кто раньше работал только с настольными БД (dBase, FoxPro, Access и т. п.).

В случае, когда в качестве SQL-сервера используется Interbase или Firebird, функцию организации доступа к данным целесообразно выполнять с помощью компонентов InterBase Express (IBX). Набор компонентов IBX – это компоненты для работы с БД InterBase/Firebird, которые используют прямой интерфейс программирования приложений (API -Application Programming Interface), т. е. обращаются к СУБД непосредственно, без каких-то промежуточных средств [11].

Приложение БД, разработанное с использованием IBX, не требует для работы установки дополнительных драйверов, нужно лишь наличие библиотеки GDS32.DLL. Для работы с Firebird необходимо при помощи входящей в комплект утилиты instclient создать gds32.dll из fbclient.dll [12].

За счет "прямого" обращения к СУБД достигается большая производительность и предоставляется доступ к продвинутым функциям сервера, недоступным при использовании стандартных компонентов.

Универсальность BDE многие годы не давала возможности воспользоваться всем потенциалом многих SQL-серверов. BDE скрывает массу особенностей работы с SQL-серверами от разработчиков (что в определенной степени упрощает разработку), но и не дает воспользоваться богатством средств управления транзакциями. В настоящее время можно говорить о том, что технология BDE устарела.

Все примеры в настоящем пособии разработаны с использованием компонентов IBX.

В табл. 2.1 представлены компоненты IBX с закладки InterBase Палитры компонентов Delphi. В настоящем пособии компоненты описываются в порядке, удобном для понимания работы приложения БД, на самой закладке в Delphi компоненты располагаются в другом порядке.

Таблица 2.1. Компоненты IBX

Компонент	*Изображение*	*Описание*
TIBDatabase		Компонент для осуществления соединения с базой данных
TIBTransaction		Компонент для явной работы с транзакциями
TIBDataSet		Основной компонент доступа к данным для получения и редактирования данных
TIBTable		Компонент доступа к данным таблицы БД
TIBQuery		Компонент для получения данных на основе SQL-запроса
TIBUpdateSQL		Компонент для создания модифицируемых наборов данных (в паре с TIBQuery)
TIBStoredProc		Компонент для выполнения хранимых процедур и получения набора данных на основе результатов выполнения процедуры
TIBSQL		Компонент для быстрого выполнения SQL-запросов
TIBEvents		Компонент для отслеживания приложением Delphi событий, происходящих в БД и вызываемых другими процессами или приложениями
TIBDatabaseInfo		Компонент для получения информации о параметрах БД и статистике выполнения запросов
TIBSQLMonitor		Компонент для мониторинга действий клиентского приложения
TIBExtract		Компонент для извлечения метаданных БД в виде скрипта
TIBScript		Компонент для выполнения скриптов (для парсинга использует компонент TIBSQLParser)
TIBSQLParser		Компонент для парсинга скриптов
TIBDatabaseINI		Компонент для записи и считывания настроек TIBDatabase в ini-файле
TIBFilterDialog		Компонент-диалог фильтрации данных
TIBConnectionBroker		Компонент для создания разделяемого соединения в многоуровневых приложениях

Рассмотрим основные компоненты IBX, многие из которых потребуются далее для создания приложений БД [6, 11, 12]. Практически каждый компонент имеет свой собственный редактор свойств.

TIBDatabase. Компонент предназначен для осуществления соединения с базой данных. Основные методы компонента: Open, Close [11].

Основные свойства:

DatabaseName – имя сервера и путь к БД (алиас, если поддерживается сервером). Например, localhost: C:\SQLLAB.FDB.

Params – параметры соединения: имя пользователя, пароль, чарсет и т. п.;

LoginPrompt – свойство определяет, выдавать ли диалог подключения к базе. Если свойство установлено в False и заполнены соответствующие параметры соединения, то запрос имени пользователя и пароля не будет выполняться каждый раз при подключении.

Connected – управление подсоединением к БД или проверка состояния соединения.

DefaultTransaction – компонент TIBTransaction, который используется по умолчанию для выполнения различных операций TIBDatabase. Если это свойство не назначено явно, TIBDatabase создаст себе экземпляр TIBTransaction самостоятельно.

IdleTimer – по умолчанию 0. Определяет время, в течение которого при отсутствии активных действий соединение с БД закроется автоматически.

SQLDialect – 1 или 3, по умолчанию 3. Определяет диалект, в котором будет работать клиентская часть. Когда TIBDatabase подсоединяется к серверу и БД, из БД считывается номер диалекта и устанавливается SQLDialect.

TraceFlags – перечень действий между клиентом и сервером, которые будет отслеживать компонент TIBSQLMonitor или внешние приложения, поддерживающие такую функциональность.

Можно воспользоваться редактором свойств TIBDatabase (вызывается двойным кликом на компоненте, рис. 2.1) для заполнения первых трех основных свойств из упомянутых.

Соединиться с базой можно, установив один раз параметры с помощью вышеописанного диалогового окна. На диалоге редактора свойств есть кнопка Test, которая позволяет проверить соединение с указанным сервером и базой данных. Если проверка прошла успешно, то свойство Connected может быть установлено в True.

Компонент TiBDatabase позволяет выполнять некоторые операции с метаданными БД с помощью определенных методов [6].

CreateDatabase – создание новой БД (параметры БД должны быть включены в список свойства Params).

DropDatabase – удаление существующей БД, путь к которой указан свойством DatabaseName.

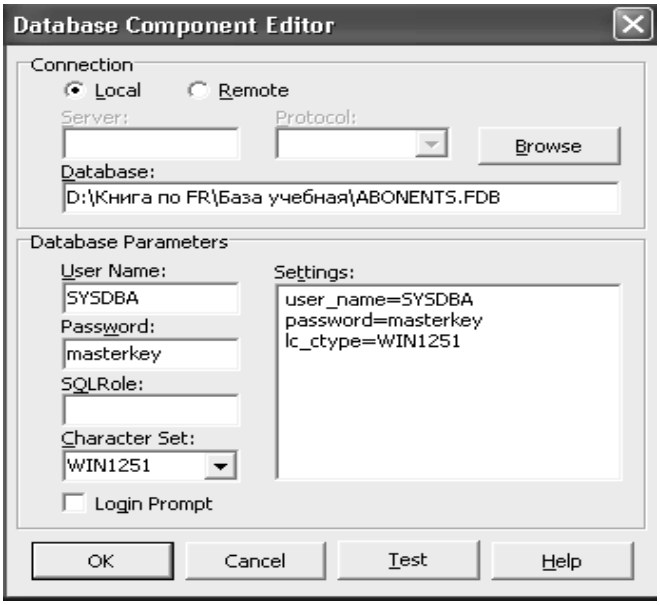


Рис. 2.1. *Редактор свойств компонента TIBDatabase*

GetTableNames(List: TStrings; SystemTables: Boolean = False) – возвращает список List имен таблиц, имеющихся в БД, при этом параметр SystemTables управляет включением в список имен системных таблиц.

GetFieldNames(const TableName: string; List: TStrings) – аналогичным образом возвращает список полей для таблицы, заданной параметром TableName.

TIBTransaction. Компонент для явной работы с транзакциями.

Транзакция – набор логически связанных операций, работающих с данными БД, и либо переводящий базу данных из одного целостного состояния в другое, либо оставляющий базу данных в целостном состоянии, существовавшем до начала транзакции [13].

Транзакция обладает такими свойствами, как *упорядочиваемость, неделимость, постоянство*.

Упорядочиваемость гарантирует, что если две или более транзакции выполняются в одно и то же время, то конечный результат выглядит так, как если бы все транзакции выполнялись последовательно в некотором порядке.

Неделимость означает, что когда транзакция находится в процессе выполнения, то никакой другой процесс не видит ее промежуточные результаты.

Постоянство означает, что после фиксации транзакции никакой сбой не может отменить результатов ее выполнения.

Основные методы компонента TIBTransaction: StartTransaction, Commit, Rollback, CommitRetaining, RollbackRetaining [11].

Существование TIBTransaction дает возможность управлять параметрами транзакций целиком и полностью, а также полноценно управлять стартом и завершением транзакций.

Следует особо отметить, что в BDE практически никакого управления транзакциями нет (одна транзакция на коннект). Отсутствие возможности явного управления транзакциями часто приводит к тому, что приложения BDE будут чаще попадать на блокировки записей, сервер будет страдать от большого числа одновременно активных транзакций или в базе будет накапливаться мусор из-за того, что какая-то транзакция активна с самого утра рабочего дня.

Клиентская часть InterBase и Firebird допускает выполнение любых действий только в контексте транзакции. Даже если можно получить доступ к данным без явного вызова IBTransaction.StartTransaction, то все равно где-то в IBX этот вызов происходит автоматически [12]. Для корректной работы приложений с базой данных желательно управлять транзакциями вручную, т. е. явно вызывать методы StartTransaction, Commit и Rollback компонента TIBTransaction.

Основные свойства:

Active – управление стартом или завершением транзакции, а также проверка состояния транзакции;

DefaultDatabase – компонент TIBDatabase;

Params – параметры транзакции;

AllowAutoStart – если False, то любые попытки автоматического старта транзакций будут пресекаться. По умолчанию True;

IdleTimer – если не 0, то время, через которое транзакция будет завершена по DefaultAction;

DefaultAction – результат автоматического завершения транзакции в случае окончания IdleTimer – taCommit, taRollback, taCommitretaining, taRollbackretaining.Также именно этим способом будут завершены все открытые транзакции в момент вызова IBDatabase.Close (Connected:=False). Значение по умолчанию зависит от версии компонента и может быть как taRollback, так и taCommit;

AutoStopAction – свойство, аналогичное DefaultAction по значениям (плюс saNone по умолчанию), указывает на метод завершения транзакции, когда все наборы данных, подключенные к ней, закрываются.

На рис. 2.2 показан редактор параметров компонента TIBTransaction (вызывается по двойному клику на компоненте или при нажатии кнопки … в свойстве Params). Если параметры не указаны (пусто), то IBX использует параметры транзакции по умолчанию, которые соответствуют уровню изолирован-

ности snapshot+wait+write. В транзакции с такими параметрами не будут видны никакие изменения БД кроме тех, которые выполнены в этой транзакции.

Рис. 2.2. *Редактор параметров транзакции*

Для обычной работы (показ данных в гриде и т. п.) обычно используются параметры ReadCommitted, так как они позволяют транзакции видеть чужие подтвержденные (committed) изменения БД просто путем повторного выполнения запросов (перечитывания данных), однако, например, для отчетов read_committed транзакции использовать нельзя.

В приложении можно использовать сколько угодно компонентов TIBTransaction. Худшими случаями являются как один компонент на все приложение, так и по компоненту на каждый выполняемый запрос. Количество необходимых экземпляров TIBTransaction определяется в соответствии с задачами приложения.

TIBDataSet. Основной компонент доступа к данным. Остальные компоненты доступа к данным (TIBTable, TIBQuery, TIBUpdateSQL…) – это модификации компонента TIBDataSet, либо чуть расширенные, либо усеченные [12].

TIBDataSet предназначен для получения и редактирования данных, является потомком стандартного класса TDataSet и полностью совместим со всеми визуальными компонентами [11]. Основные методы: Prepare, Open, Close, Insert, Append, Edit, Delete, Refresh.

Основные свойства:

Database – связь с компонентом TIBDatabase.

DataSource – ссылка на Master – источник данных (TDataSource) для TIBDataSet, используемого в качестве Detail.

ForcedRefresh – при False перечитывание данных (текущей записи) производится только при явном вызове Refresh. При True – происходит автоматически при Post. Вызов метода Refresh будет перечитывать только одну, теку-

щую запись. Для перечитывания всего набора нужно закрыть и открыть датасет, т. е. повторно выполнить запрос, хранимый в свойстве SelectSQL.

GeneratorField – по нажатию кнопки "..." выводится диалог для конфигурирования "автоинкремента" – какому столбцу, какой генератор, с каким шагом и по какому действию (onNewRecord, onPost, onServer). Это автоматический вызов инкремента значения генератора и помещения его в столбец.

ParamCheck – проверять или нет наличие в запросе параметров (:param). По умолчанию True. Для выполнения операторов DLL (create procedure, alter, drop и т. п.) нужно выставить в False.

Transaction – указывает к какой транзакции привязан компонент.

Запросы:

SelectSQL – основной запрос, который возвращает данные.

RefreshSQL – запрос для обновления текущей строки. Должен содержать условие отбора по первичному ключу или подобное, для выборки одной записи.

InsertSQL – запрос для вставки записи.

UpdateSQL – запрос для обновления записи.

DeleteSQL – запрос для удаления записи.

Для активации TIBDataSet необходимо выполнить ряд действий:

- Соединить его с нужным TIBDatabase и с TIBTransaction (если это не тот TIBTransaction, который указан для IBDatabase.DefaultTransaction).
- Определить запрос, выбирающий данные [для выполняемой хранимой процедуры (ХП) использовать TIBStoredProc или TIBQuery]. В этот момент подсоединенные TIBDatabase и TIBTransaction должны быть активны. Запрос можно указать, щелкнув по кнопке "..." свойства SelectSQL или нажав на компоненте правую кнопку мыши, выбрать в меню Edit SQL.

Если редактирования данных, возвращаемых запросом, записанным в свойстве SelectSQL, не предполагается, то на этом можно остановиться – после Active:=True компонент выберет данные. Иначе можно задать запросы Insert/Update/Delete/Refresh. Это можно сделать вручную (прописав запрос в окне редактора, открывающемся при щелчке по кнопке "..." у соответствующего свойства) или автоматически (с помощью DataSet Editor).

TIBTable. Данный компонент был включен в состав IBX фактически для облегчения перехода на IBX пользователей, которые ранее использовали BDE (как заменитель компонента TTable). Для работы требует указания имени таблицы, после чего запрос на считывание данных формируется автоматически.

Основные методы: Open, Close. Набор данных, полученный при помощи TIBTable, является редактируемым, если речь идет о таблице БД или обнов-

ляемом представлении [11]. Компонент совместим с визуальными компонентами. Основные свойства:

TableName – имя таблицы.

TableTypes – типы таблиц, отображаемые в выпадающем списке TableName – ttSystem включает выборку системных таблиц (rdb$, tmp$), а ttView – включает выборку VIEW.

IndexName – имя индекса для сортировки. Так же, как и в BDE, имя индекса используется косвенно, для получения столбцов индекса и добавления фразы ORDER BY INDEXFIELD1, INDEXFIELD2 к автоматически конструируемому запросу. Поэтому для сортировки по любому столбцу, даже непроиндексированному, следует использовать свойство IndexFieldNames. Явно сортировку записей в обратном порядке (DESC) указать невозможно.

Filter – условие фильтрации в виде Поле [Оператор сравнения] 'Значение' [7]. Текст свойства Filter добавляется как условие where к автоматически конструируемому запросу.

Поскольку свойство ReadOnly по умолчанию False, IBTable сразу готов к редактированию данных. Так как запрос на чтение формируется из одной таблицы, то запросы insert/update/delete могут быть легко сформированы автоматически. Сортировка записей по умолчанию (order by) производится по столбцам первичного ключа (primary key).

Следует отметить, что ни один специалист не рекомендует использование компонента TIBTable в реальных программных проектах, предназначенных для управления серьезными БД в многопользовательской среде [11]. TIBTable формирует SQL-запросы к БД самостоятельно, а это при работе с SQL-серверами считается некорректным (запросы необходимо контролировать, для того чтобы знать, что именно делается для выборки данных) [12]. Более того, программисты, ранее использовавшие BDE, часто используют TIBTable для выборки из таблиц с большим числом записей. Допустим, что в таблице несколько миллионов записей и TIBTable пытается получить их в полном объеме на клиента, а только потом выбирать нужную запись с помощью фильтрации, например. Очевидно что это вызовет колоссальную нагрузку на SQL-сервер и клиента, особенно в многопользовательской среде [11].

В основном, конечно, TIBTable предназначен для облегчения переноса приложений BDE на компоненты IBX. Иногда удобно использовать TIBTable для осуществления доступа к справочникам локальных или небольших БД.

Следует отметить, что в настоящем пособии не рассматриваются подробно свойства и методы, характерные только для табличных компонентов. Например, свойства и методы для работы с индексами присутствуют только в табличных компонентах, так как в компонентах запросов работа с индексами осуществляется средствами SQL, поэтому не рассматривают-

ся в пособии. Также, например, фильтрация характерна только для табличных компонентов и т. д.

TIBQuery. Компонент предназначен для получения данных на основе SQL-запроса [11]. Компонент TIBQuery совместим с визуальными компонентами.

Основные методы: Open, Close, ExecSQL.

Дополнительно к свойствам TIBDataSet (кроме отсутствующих *Refresh/Insert/Update/DeleteSQL*) TIBQuery имеет свойство *Constraints*.

Для запросов, возвращающих набор записей, можно установить свойство Active в True или вызвать метод Open. Для остальных запросов, таких как insert/update/delete, execute или операторов DDL (CREATE TABLE, ALTER, DROP...), нужно вызывать метод ExecSQL, так как эти запросы не возвращают записи, а возвращают только результат выполнения оператора SQL.

Компоненты TIBDataSet и TIBQuery могут выполнять как статический, так и динамический SQL. Динамический SQL отличается от статического наличием параметров. Для параметризированного запроса свойство ParamCheck должно быть True.

Если запрос предполагает задание параметра, то текст обработчика выглядит примерно таким образом:

```
IBQuery.SQL.Clear; // очистить текст sql
// задать текст запроса
IBQuery.SQL.Add ('select * from abonent A');
IBQuery.SQL.Add ('where a.accountCD = :param');
IBQuery.Prepare; // отправить запрос на сервер, проверить его корректность,
// задать значение параметра
IBQuery.ParamByName('param').asString:='005488';
IBQuery.Open; // или IBQuery.ExecSQL
```

Для статических запросов вызов Prepare необязателен – компонент сам его выполнит автоматически, если Prepare не был вызван. Prepare очень удобен при повторяющемся выполнении одного и того же запроса, с разными значениями параметра. При этом Prepare вызывается один раз, а установка параметров и вызов ExecSQL производятся столько раз, сколько нужно. Чаще всего такой способ используется для запросов Insert и Update.

Данный компонент, как и TIBTable, фактически предназначен для облегчения перехода на IBX пользователей, которые ранее использовали BDE (как заменитель компонента TQuery). Аналогично TIBTable компонент TIBQuery скрывает запросы для получения и редактирования данных [11]. Вместо скрытого в этом компоненте свойства SelectSQL используется свойство SQL (на самом деле после присвоения свойства SQL компонент присваивает его значение свойству SelectSQL).

Поскольку свойства DeleteSQL, InsertSQL и ModifySQL спрятаны, то TIBQuery сам по себе не может предоставить разработчику редактируемые

запросы. Если требуется выполнять редактирование запроса, к нему может быть подключен компонент TIBUpdateSQL, который содержит собственные свойства DeleteSQL, InsertSQL и ModifySQL.

Фактически, в данной связке имеется смысл, только если речь идет о миграции готовых BDE-приложений на IBX. При разработке нового приложения, которое с самого начала базируется на IBX, рекомендуется использовать TIBDataSet как вместо TIBTable, так и вместо TIBQuery [11]. Для выполнения запросов, не возвращающих результирующий набор данных, в TIBQuery, как уже было отмечено, предусмотрен метод ExecSQL. Однако часто такие запросы не требуют использования компонентов, предназначенные для взаимодействия с визуальными компонентами. Для запросов, не возвращающих набор строк, можно использовать компонент TIBSQL, который не буферизует получаемые данные и предназначен именно для простого и быстрого выполнения SQL-запросов (будет описан далее).

TIBUpdateSQL. Используется в паре с TIBQuery и предназначен для создания модифицируемых наборов данных. Основные свойства: *DeleteSQL*, *InsertSQL*, *ModifySQL* и *RefreshSQL* [11].

TIBStoredProc. Компонент предназначен для выполнения хранимых процедур и получения набора данных на основе результатов выполнения процедуры. Получаемый набор данных является нередактируемым. Компонент совместим с визуальными компонентами. Основное свойство – StoredProc Name. Основной метод – ExecProc [11]. Собственно, выполнять процедуры можно, вызывая EXECUTE PROCEDURE в компоненте TIBQuery или TIBDataSet. Единственным преимуществом использования TIBStoredProc по сравнению с TIBDataSet является тот факт, что TIBStoredProc самостоятельно формирует список параметров процедуры по ее имени, обращаясь к системным таблицам [11].

Для выборки из селективных процедур (select * from proc) следует использовать обычные TIBDataSet, TIBQuery (TIBStoredProc принципиально не умеет работать как TIBDataSet по простой причине – выполнение процедур в нем предусмотрено только как execute procedure).

В общем случае TIBStoredProc не рекомендуется к использованию, так как имеет проблему с обработкой ошибок.

TIBSQL. Предназначен для быстрого выполнения SQL-запросов. В отличие от TIBQuery или TIBDataSet, TIBSQL не имеет локального буфера для набора данных и несовместим с визуальными компонентами. [11].

Для обеспечения скорости выполнения запроса из компонента удалены все дополнительные механизмы, обслуживающие набор данных. Фактически компонент TIBSQL не имеет отношения к обычным компонентам доступа к данным, его непосредственным предком является класс icomponent, а не

TDataSet. Поэтому он только передает через компонент соединения TiBDatabase запрос серверу и получает назад результат выполнения запроса.

Для повышения скорости компонент не обеспечивает полноценной навигации по набору данных. Перемещение по набору данных возможно только в прямом направлении (однонаправленный курсор) [6].

Для доступа к текущей записи существует только свойство *Current*. Есть свойство *GoToFirstRecordOnExecute* – если True (по умолчанию), то в случае Select компонент запросит у сервера первую запись и заполнит ее данными Current. Если False – Current будет пустым, и для считывания данных с сервера необходимо выполнить IBSQL.Next.

Предназначен для выполнения операторов Insert/update/delete, execute procedure, или select для небуферизированного перебора записей (например, при импорте/экспорте данных или при массовой обработке записей на клиенте) [12].

TIBEvents. Предназначен для отслеживания клиентским приложением Delphi, работающим с СУБД, событий, происходящих в БД и вызываемых другими процессами или приложениями [6]. Для этого используется компонент TiBEvents. Он позволяет определить список необходимых событий и предоставляет разработчику простой механизм отслеживания возникающих на сервере событий. Основное свойство – Events. Основные методы: RegisterEvents, UnresisterEvents [11].

TIBDatabaselnfo. Компонент для получения информации о параметрах БД и статистике выполнения запросов [6]. Подсоединяется к TIBDatabase. Наиболее часто используемые свойства:

UserNames – список пользователей, подключенных к серверу архитектуры SuperServer. Для Classic возвращает только текущего пользователя.

Reads, Writes, Fetches – информация о числе чтений, записи и обращений к страницам БД. Аналог информации, выводимой IBExpert и др. инструментами после выполнения запроса.

PageSize, ODSxxx, NumBuffers, SweepInterval... – информация о свойствах БД.

ReadSeqCount, UpdateCount, InsertCount... – счетчики соответствующих операций, выводятся для всех таблиц (поштучно). Аналог информации, выводимой в закладке Performance Info IBExpert.

TIBSQLMonitor. Компонент для мониторинга действий клиентского приложения [12]. Для мониторинга необходимо в событии *OnSQL* сохранять EventText и EventTime либо в файле, либо в компоненте TMemo на какой-либо форме (например, Memo.Lines.Add(EventText)).

В компоненте соединения с БД можно установить перечень событий сервера, на которые будет реагировать компонент TIBSQLMonitor [11]. Это делается при помощи свойства *TraceFiags*. Вероятные значения множества означают контроль над операциями:

tfQPrepare – подготовка запроса к выполнению (вызов метода Prepare);
tfQExecute – выполнение запроса (вызов метода ExecSQL);
tfQFetch – вызов запроса (вызов методов Open, Close);
tfError – возникновение ошибки;
tfstmt – все операции с запросами;
tfconnect – подключение и отключение БД;
tfTransact – выполнение транзакций;
tfBlob – операции с данными BLOB;
tfService – вспомогательные операции;
tfMisc – любые операции, не учтенные вышеперечисленными значениями.

TIBExtract. Компонент для извлечения метаданных БД (таблиц, процедур, триггеров...) в виде скрипта [12].

TIBScript. Компонент для выполнения скриптов [12]. Для парсинга скриптов компонент использует компонент TIBSQLParser.

TIBDatabaseINI. Компонент для записи и считывания настроек TIBDatabase в ini-файле [12].

Следует отметить, что в рамках настоящего пособия не ставится цели подробно рассмотреть все компоненты IBX. Упор будет сделан на основные компоненты, без использования которых невозможно создать работоспособное приложение БД.

2.2. Размещение невизуальных компонентов в модуле данных

При разработке сложных приложений принято разделять логику работы с данными и пользовательский интерфейс [8]. В Delphi это помогают сделать модули данных – компоненты-контейнеры типа TDataModule. Это невидимые формы, на которых можно размещать такие невизуальные компоненты, как наборы данных, источники данных, провайдеры и иные элементы, осуществляющие связь с БД. Визуальные компоненты в модуле данных разместить невозможно.

Модуль данных не является обязательной составляющей приложения БД, можно работать и без него, размещая невизуальные компоненты на обычных формах. Однако в большинстве реальных приложений БД используются модули данных.

Модуль данных доступен разработчику, как и любой другой модуль проекта, на этапе разработки. Пользователь приложения не может увидеть модуль данных во время выполнения [6].

Есть три типа модулей данных [4]:

- простой модуль данных;

- удаленный модуль данных;
- Web-модуль.

Удаленный модуль данных предназначен для работы с удаленными БД в трехуровневой архитектуре "клиент-сервер" и используется для создания сервера приложения – промежуточного уровня между приложением и сервером БД.

Web-модуль предназначен для работы с БД в сети Интернет и является посредником между обозревателем (программой просмотра Web-документов) и сервером БД.

В рамках настоящего пособия рассматривается только *простой модуль данных*, который представлен объектом DataModule.

На рис. 2.3 приведена взаимосвязь приложения и таблицы БД при использовании компонентов IBX и простого модуля данных.

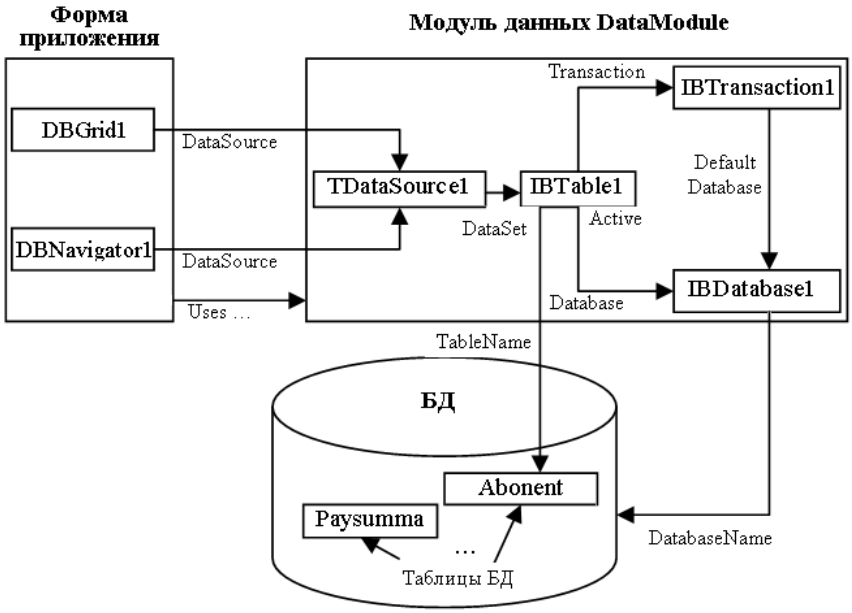


Рис. 2.3. *Взаимосвязь компонентов при использовании модуля данных*

Добавление модуля данных к проекту выполняется командой File/New /Other/Delphi Files/Data Module главного меню Delphi. Модуль данных, как и форма, является контейнером для своих невизуальных компонентов, и для него создается модуль кода с расширением pas [4]. Размещение компонентов в окне модуля, просмотр и редактирование их свойств аналогично действиям с обычной формой.

На рис. 2.4 приведена форма модуля данных с размещенными на ней компонентами.

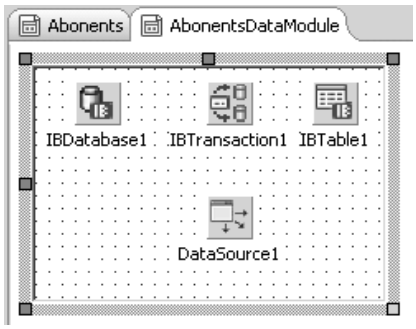


Рис. 2.4. *Форма модуля данных*

Чтобы обеспечить возможность доступа к компонентам модуля данных в модуле формы, в список uses раздела implementation необходимо включить ссылку на модуль данных:

```
uses DataModule1;
```

Ссылку на модуль можно вставить автоматически с помощью команды *File\Use Unit*.

При обращении к содержащимся в модуле данных компонентам для них указывается составное имя, в которое, кроме имени компонента, входит имя модуля данных:

```
<Имя модуля данных>.<Имя компонента>.
```

Например: DataModule1.Tablel.DatabaseName.

Модуль данных имеет события *OnCreate* и *OnDestroy*, возникающие при его создании и уничтожении [8]. В обработчиках этих событий имеет смысл предусмотреть соответственно операторы открытия соединений с таблицами БД и закрытия этих соединений. Примеры таких функций, а также некоторых других, которые полезно использовать в модулях данных, будут приведены при описании примера разработки приложения БД (в гл. 5).

Таким образом, модуль данных позволяет [4]:

- отделить управление БД от обработки данных;
- создать модуль, совместно используемый несколькими приложениями.

Основным назначением модуля данных является централизованное хранение компонентов доступа к данным, а также кода для этих компонентов, в частности, обработчиков событий. В модуле данных удобно размещать код, выполняющий управление БД, например, реализацию бизнес-правил. Изме-

нение значения любого свойства проявится сразу же во всех обычных модулях, к которым подключен модуль данных [6].

Использование простого модуля данных несколькими приложениями позволяет ускорить разработку приложений, так как готовый модуль данных впоследствии можно включать в новые приложения. Создав и сохранив в файле модуль данных, можно подключать его к любым приложениям, работающим с соответствующими таблицами, и не заботиться в дальнейшем о соединении с БД. Достаточно просто подсоединить компоненты отображения и редактирования данных к соответствующим источникам, созданным в модуле данных.

Кроме того, управление БД через общий модуль дает возможность определить для всех пользователей одинаковые режимы и правила работы с базой, а также делает более простым изменение этих режимов и правил.

Однако для небольших приложений использование простого модуля данных не всегда оправданно, так как может затруднить, а не облегчить разработку приложения.

2.3. НАБОР ДАННЫХ

Набор данных (НД) является тем базовым понятием, вокруг которого "вращается" любая деятельность приложения БД [6].

В первом разделе настоящего пособия уже было дано определение набора данных, здесь рассмотрим подробно особенности работы с наборами данных в приложении через невизуальные компоненты.

В объектно-ориентированной среде для представления какой-либо группы записей приложение должно использовать возможности некоторого класса. Этот класс должен инкапсулировать набор данных и обладать методами для управления записями и полями. Разработчики VCL уделили особое внимание созданию максимально эффективной иерархии классов, обеспечивающих использование наборов данных. На рис. 2.5 приведена структура иерархии классов наборов данных.

Класс TDataset является базовым классом иерархии, он инкапсулирует абстрактный набор данных и реализует максимально общие методы работы с ним. На основе базового класса реализованы специальные компоненты VCL для различных технологий доступа к данным, которые позволяют разработчику конструировать приложения БД, используя одни и те же приемы и настраивая одинаковые свойства.

Компоненты-наборы данных (потомки TDataset) входят в состав невизуальных компонентов доступа к данным, и, как видно на рис. 2.5, набор компонентов для каждой из представленных технологий доступа к данным примерно одина-

ков. Везде есть компонент, инкапсулирующий табличные функции, компонент запроса SQL и компонент хранимой процедуры. Функциональность подобных компонентов в различных технологиях почти одинакова.

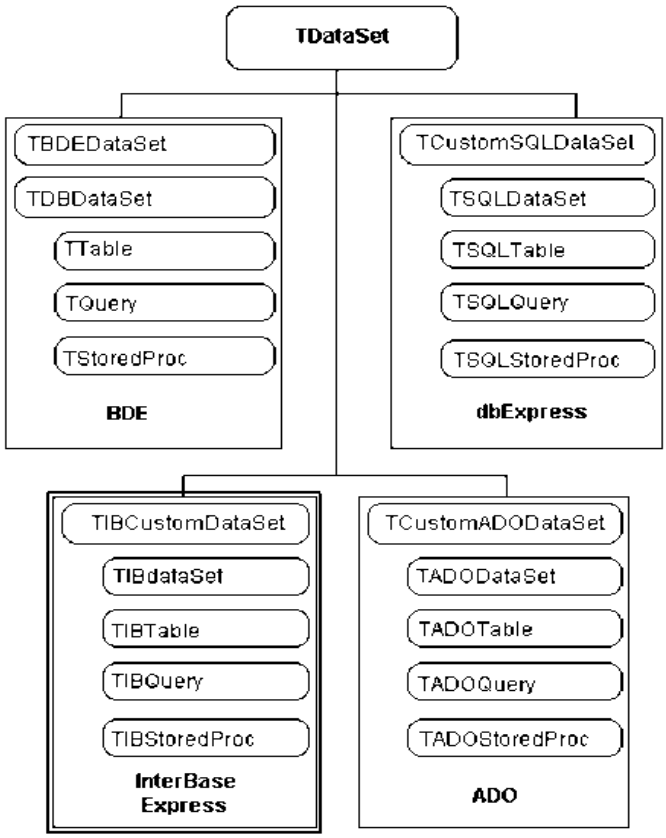


Рис. 2.5. *Иерархия классов, обеспечивающих функционирование набора данных*

Отличительные особенности и основные свойства различных компонентов-потомков Dataset были рассмотрены при описании технологий доступа к данным. Рассмотрим свойства и методы компонентов наборов данных, общие для всех наборов данных, как потомков класса TDataset. Все приведенные в настоящем пособии примеры основаны на компонентах IBX, однако могут быть легко модифицированы для использования с другими технологиями доступа.

2.3.1. Навигационный и реляционный способы доступа к данным

Для выполнения операций с наборами данных используются два способа доступа к данным: *навигационный и реляционный* [4].

Навигационный способ доступа заключается в обработке каждой отдельной записи набора данных. Этот способ обычно используется в локальных БД или в удаленных БД небольшого размера. При навигационном способе доступа каждый набор данных имеет невидимый указатель текущей записи. Поля текущей записи доступны для просмотра.

Указатель определяет запись, с которой могут выполняться такие операции, как редактирование или удаление. При этом стоит сразу отметить, что не каждый набор данных можно редактировать в принципе (признаки редактируемых наборов данных будут описаны далее).

На навигационном способе доступа основана работа визуальных компонентов из вкладки «Data Controls» в Delphi (визуальные компоненты будут рассмотрены в гл. 4).

Достоинством навигационного способа является простота кодирования операций, а основной недостаток состоит в том, что приложение получает все записи набора независимо от того, сколько их требуется обработать на самом деле. Это приводит к большой нагрузке на сеть, особенно при интенсивном обмене данными. При работе с удаленными БД можно использовать навигационный способ доступа, но только для небольших сетей, чтобы не создавать большой загрузки [4].

Реляционный способ доступа основан на обработке группы записей. Если требуется обработать одну запись, все равно обрабатывается группа, состоящая из одной записи. При реляционном способе доступа используются SQL-запросы, поэтому его называют также SQL-ориентированным.

Средства SQL применимы для выполнения операций с локальными и удаленными БД. Наиболее полно преимущества реляционного способа доступа и языка SQL проявляются при работе с удаленными БД. Основным достоинством реляционного способа доступа является небольшая загрузка сети, поскольку передаются только запросы и результат их выполнения.

Применение реляционного способа доступа для локальных БД не дает существенного преимущества, но и в этом случае с помощью SQL-запроса можно выполнять следующие действия [4]:

- формировать состав полей набора данных при выполнении приложения;
- включать в набор данных поля и записи из нескольких таблиц;
- отбирать записи по сложным критериям;
- сортировать набор данных по любому полю, в том числе неиндексированному;
- осуществлять поиск данных по частичному совпадению со значениями в поле.

При использовании компонентов IBX для компонента TIBTable всегда реализуется только навигационный способ доступа к данным. Как уже было отмечено ранее, использование компонента TIBTable не рекомендуется для доступа к таблицам с большим количеством записей (причины пояснялись подробно в п. 2.1.2 на стр.23).

TIBQuery поддерживает два способа доступа. Реляционный способ доступа реализуется в случае, когда используются только средства SQL-запросов. Если дополнительно используются методы, ориентированные на операции с отдельными записями (Next или Edit), визуальные компоненты, ориентированные на операции с текущими записями (DBGrid, который указывает текущую запись с помощью специального маркера, или DBEdit и DBText, отображающие содержимое соответствующих полей именно текущей записи), то будет реализован навигационный способ доступа со всеми его недостатками.

Реляционный способ доступа к данным также реализуется с помощью компонентов TIBDataSet и TIBStoredProc.

2.3.2. Состояния наборов данных

Наборы данных могут находиться в открытом или закрытом состояниях, на что указывает свойство *Active* типа Boolean [4]. Если свойству Active установлено значение True, то набор данных открыт.

Набор данных может быть открыт на этапе разработки приложения. Если при этом к набору данных через источник данных DataSource подключены визуальные компоненты, например, DBGrid или DBEdit, то они отображают данные соответствующего набора данных.

Например, открытый компонент TIBTable содержит набор данных, соответствующий данным таблицы, связанной с ним через свойство TableName. Для открытого компонента TIBQuery набор данных соответствует результатам выполнения SQL-запроса, содержащегося в свойстве SQL этого компонента.

Если свойство Active имеет значение False (по умолчанию), то набор данных закрыт, и его связь с БД разорвана.

На этапе проектирования свойство Active наборов данных автоматически устанавливается в значение False при изменении значения свойств DataBaseName, TableName или SQL.

Следует сразу отметить, что заранее выставлять Active = true допустимо только в процессе настройки и отладки приложения, работающего с локальными БД [8].

В законченном приложении во всех наборах данных сначала должно быть установлено Active = false, затем при событии формы OnCreate эти свойства могут быть установлены в true (или вызван метод Open), а при событии формы OnDestroy эти свойства опять должны быть установлены в false (или вызван метод Close). Это исключит неоправданное поддержание связи с базой

данных, которое занимает ресурсы, а при работе в сети мешает доступу к БД других пользователей.

Управлять состоянием набора данных можно также программно, с помощью методов *Open* и *Close*.

Процедура Open открывает набор данных, ее вызов эквивалентен установке свойства Active в значение True. При вызове метода Open генерируются события BeforeOpen и AfterOpen, а также вызываются процедуры-обработчики этих событий.

Событие *BeforeOpen* возникает непосредственно перед открытием набора данных. В обработчике этого события можно выполнить проверку определенных условий, и если они не соблюдаются, то открытие набора данных может быть запрещено. При этом обработчик события не содержит специального параметра, с помощью которого можно запретить открытие набора данных.

Событие *AfterOpen* генерируется сразу после открытия набора данных. Это событие можно использовать, например, для выдачи пользователю сообщения о возможности работы с данными.

Процедура *Close* закрывает набор данных, ее вызов эквивалентен установке свойства Active в значение False. При вызове метода close генерируются события *BeforeClose* и *AfterClose*, а также вызываются процедуры-обработчики этих событий.

Следует отметить, что закрытие набора данных автоматически не сохраняет текущую запись, т. е. если набор данных при закрытии находился в режимах редактирования или вставки, то произведенные изменения данных в текущей записи будут потеряны. Поэтому перед закрытием набора данных нужно проверять его режим (режимы наборо данных будут описаны далее) и при необходимости принудительно вызывать метод *Post*, сохраняющий сделанные изменения. Одним из вариантов сохранения изменений является вызов метода Post в обработчике события BeforeClose, возникающего непосредственно перед закрытием набора данных. При этом следует помнить, что при закрытии приложения событие BeforeClose не генерируется, и несохраненные изменения теряются.

2.3.3. Перемещение по записям набора данных

Набор данных представляет собой множество записей данных, как уже неоднократно было отмечено. Во время выполнения программы после открытия набора данных по записям можно перемещаться, получать доступ к отдельным полям и т. д. [6].

Рассмотрим основные свойства, ответственные за навигацию по НД, за получение доступа к отдельным записям НД.

Один и тот же набор данных в разные моменты времени может содержать различные записи в зависимости от ограничений и критерия фильтрации [4],

т.е. управление числом записей в наборе данных осуществляется косвенно – путем отбора записей тем и иными способами.

Общее число записей, составляющих набор данных, возвращает свойство *RecordCount* типа Longint. Это свойство доступно для чтения при выполнении приложения. Однако следует помнить, что каждое обращение к этому свойству приводит к обновлению набора данных, что может вызвать проблемы для больших таблиц или сложных запросов [6].

Признаком того, что достигнута последняя запись набора, является свойство *Eof* типа Boolean, которое в этом случае имеет значение True.

Аналогичную функцию для первой записи выполняет свойство *Bof* типа Boolean.

Частой задачей при программной работе с наборами данных является определение, не является ли набор данных пустым. Для этого можно использовать метод *IsEmpty* типа Boolean, который возвращает значение True, если набор данных пуст. Также можно использовать свойства Eof и Bof.

Пример определения "пустого" набора данных

```
if IBQuery1.IsEmpty
  then ShowMessage('Набор данных пустой!');
```

или

```
if IBQuery1.Bof and IBQuery1.Eof
  then ShowMessage('Набор данных пустой!'); .
```

Номер текущей записи позволяет узнать свойство *RecNo* типа Integer.

Размер записи в байтах возвращает свойство *RecordSize* типа Word.

Для осуществления навигации по НД используется ряд методов, перечисленных в табл. 2.2.

Таблица 2.2. Основные методы навигации по НД

Метод	*Назначение метода*
procedure Next;	Перемещение курсора на одну запись вперед
procedure Prior;	Перемещение курсора на одну запись назад
procedure First;	Перемещение курсора на первую запись
procedure Last;	Перемещение курсора на последнюю запись
function MoveBy (Distance: Integer): Integer;	Перемещение вперед и назад на заданное число записей. Параметр Distance определяет число записей. Если параметр отрицательный – перемещение осуществляется к началу набора данных, иначе – к концу
procedure DisableControls;	Отключение связанных с набором данных компонентов отображения данных (для ускоренного перемещения по набору данных)
procedure EnableControls;	Подключение связанных с набором данных компонентов отображения данных (отключенных методом DisableControls)

Ниже приведен классический пример навигации по записям набора данных в цикле с помощью метода Next. Такой цикл может быть использован в приложении для выполнения необходимых действий с отдельными записями набора данных (комментарий заменяется программным кодом).

```
with IBDataset1 do
begin
  Open;
  while not Eof do
  begin
    //работа с отдельными записями набора …
    Next;
  end;
end;
```

2.3.4. Доступ к полям записей наборов данных

После получения доступа к конкретной записи набора данных можно получить доступ к конкретным полям текущей записи. В процессе программирования разработчик очень часто обращается к полям набора данных [6].

Каждая запись набора данных в приложении представляет собой совокупность значений полей таблицы исходного набора данных, сформированного в результате выборки данных из БД. Однако состав полей набора данных и их количество в приложении может отличаться от состава и количества полей исходного набора данных. Состав полей (набор объектов-полей) формируется при разработке приложения с помощью Редактора полей набора данных (см. стр. 44). Кроме того, возможно динамическое изменение состава полей во время выполнения приложения.

Каждое поле набора данных представляет собой отдельный столбец, для работы с которым в Delphi служат объект *Field* типа TField и объекты производных от него типов, например, TlntegerField, TFloatField или TStringField [4]. Для доступа к объектам *Field* и, соответственно, к полям записей у набора данных есть специальные методы и свойства, доступные при выполнении приложения.

Количество полей набора данных возвращает свойство *FieldCount* типа Integer, а количество полей типа BLOB содержится в свойстве *BlobFieldCount*. Данные свойства доступны только для чтения. Как уже было отмечено, число полей в наборе данных может изменяться в зависимости от настройки компонента набора данных.

Совокупность полей (столбцов) набора данных инкапсулирует свойство: *Fields [Index: Integer]: TField.*

К отдельному полю можно обратиться, указав его номер index в массиве полей Fields. Номера полей находятся в пределах от нуля до FieldCount-1. В отличие от Редакторов полей и столбцов, применяемых на этапе разработ-

ки приложения, свойство Index можно использовать для определения и изменения порядка полей набора данных во время выполнения приложения.

Ниже приведен пример чтения значений полей текущей записи. Приведенный программный код может быть использован в коде модуля формы приложения, на которой предварительно размещены компоненты IBTable1 и ListBox1.

```
for i := 0 to IBTable1.FieldCount – 1 do
  ListBox1.Items.Add(IBTable1.Fields[i].AsString);
```

В данном примере содержимое каждого поля текущей записи набора данных IBTable1 интерпретируется как строковое значение и добавляется к списку ListBoxl (визуальные компоненты отображения данных будут рассмотрены в гл. 4).

Следующий пример демонстрирует задание формата отображения полей. Приведенный программный код может быть использован в коде модуля формы приложения, на которой предварительно размещен компонент IBTable1.

```
for i := 0 to IBTable1.FieldCount – 1 do
  IBTable1.Fields[i].DiplayFormat := '#.###';
```

Форматирование с помощью свойства *DiplayFormat* относится только к отображению значений в визуальных компонентах и не затрагивает вид и размещение данных в таблице БД [4].

Номер поля в наборе данных не является заранее фиксированным числом и зависит от порядка полей в таблице БД, от текущего состава полей набора данных, а также от значений свойств некоторых связанных визуальных компонентов. Так, при изменении порядка расположения столбцов в компоненте DBGrid соответственно изменяется порядок следования полей набора данных. В связи с этим метод Fields можно использовать, только если точно известно, что структура полей набора данных жестко задана и не изменяется. Иначе обращение к какому-либо полю по его номеру в массиве полей может вызвать обращение совсем к другому полю.

Для доступа к полям более предпочтительно использовать методы *FindField* и *FieldByName.*

Метод

FindField (const FieldName: string): TField;

возвращает для набора данных поле, имя которого указывает параметр FieldName [4]. В отличие от номера в массиве полей, имя поля более статично, несет большую смысловую нагрузку и изменяется реже. Если заданное параметром FieldName поле не найдено, то метод FindField возвращает значение Nil. На практике чаще всего используется метод

FieldByName(const FieldName: string): TField;,

который отличается от метода FindField тем, что если заданное поле не найдено, то генерируется исключение.

При использовании указанных методов следует учесть, что имя поля, определяемое параметром FieldName, является именем физического поля таблицы БД, заданным при создании таблицы, а не именем (свойством Name) объекта Field, которое создано для этого поля. При этом для наборов данных, основанных на SQL-запросах, FieldName физического поля можно переопределить в тексте запроса. Имя поля, передаваемое в параметре FieldName, не чувствительно к регистру символов [4].

Свойство Fields и методы FindField и FieldByName наиболее часто используются для доступа к значению поля текущей записи совместно со свойствами объекта Field вида *Asxxx*:

AsVariant типа Variant;
AsString типа String;
Aslnteger типа Longint;
AsFloat типа Double;
AsCurrency типа Currency;
AsBoolean типа Boolean;
AsDateTime типа TdateTime.

При использовании любого из этих свойств выполняется автоматическое преобразование типа значения поля к типу, соответствующему названию свойства. При этом преобразование должно быть допустимо, в противном случае возникает ошибка компиляции по несоответствию типов, например, при попытке прочитать логическое значение как целочисленное.

Ниже приведен пример, где выполняется чтение содержимого поля Name текущей записи как строкового значения и отображение этого значения в надписи Label1.

```
Labe1.Caption := IBTable1.FieldByName('Name').AsString;
```

Аналогичным образом можно присваивать произвольному полю текущей записи новое значение (будет рассмотрено в п. 2.3.6).

Для получения списка имен полей набора данных можно использовать метод *GetFieldNames(List: TStrings)*, который возвращает в параметр List полный список имен полей НД [6].

2.3.5. Поиск данных

Одна из важнейших для пользователя операций с БД – поиск записей по некоторому ключу [8]. Для программиста задача состоит в том, чтобы найти определенную строку внутри текущего набора данных, т. е. позиционировать курсор на определенной строке [7].

Можно перебирать все записи в цикле с помощью метода Next, пока не будет найдена необходимая, но данный способ нерационален. Существует несколько специальных методик поиска записей [7, 8].

Две из них – *SetKey* и *FindKey*, наиболее ранние методики (начиная с Delphi 1), в пособии подробно не рассматриваются, так как применимы только для табличных компонентов.

Два других метода – *Lookup* и *Locate*, появились позднее. Данные методы универсальнее, применимы к любым наборам данных, не требуют предварительной специальной индексации набора данных.

Метод Locate объявлен так [8]:

```
Locate(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions): Boolean;
```

Метод Locate имеет три параметра:

KeyFields – строка, содержащая список имен полей, по которым ведется поиск, разделенных точками с запятой;

KeyValues – массив искомых значений этих полей;

Options – параметры поиска.

В качестве параметров поиска можно указывать сочетания из двух констант:

loPartialKey – допустимость частичного совпадения (искомое значение может совпадать не полностью, т. е. может совпадать часть значения, но с самого начала);

loCaselnsensitive – нечувствительность поиска к регистру, в котором введены символы.

Метод Locate возвращает true, если искомая запись найдена, иначе метод вернет false.

Допустим, есть набор данных IBQuery1, в котором содержаться данные об абонентах, и необходимо найти абонента по ФИО, введенному пользователем в поле ввода Editl. Пример вызова метода Locate для данной задачи приведен ниже.

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  IBQuery1.Locate ('FIO', Edit1.Text,[loCaselnsensitive,loPartialKey]);
end;
```

Поиск по ФИО выполняется в обработчике события OnChange поля ввода Edit1. При вводе пользователем в поле каждого нового символа фамилии курсор будет перемещаться на запись, наиболее близко совпадающую с уже введенными символами. Поиск будет работать, независимо от того, индексирована ли база данных по полю FIO, однако при наличии индексации поиск производится быстрее.

Можно осуществлять поиск по нескольким полям. Например, необходимо найти абонента не просто по ФИО, но и с номером лицевого счета, начинающимся с определенных символов (задаются пользователем в поле ввода Edit2).

При поиске по нескольким полям вторым аргументом в вызове метода Locate должен быть массив Variant. Для задания этого аргумента можно воспользоваться функцией VarArrayOf, которая формирует тип Variant из задаваемого ей массива параметров любого типа. Вызов метода будет выглядеть так:

```
IBQuery1.Locate ('FIO;AccountCD', VarArrayOf([Edit1.Text,Edit1.Text]),
[loCaselnsensitive,loPartialKey]);
```

Рассмотрим метод *Lookup* [8]:

```
Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant;
```

Первые два параметра аналогичны методу Locate. Третий параметр ResultFields – строка, перечисляющая разделенные точками с запятой имена полей, значения которых возвращаются в виде массива Variant. При успешном завершении поиска метод возвращает значение или значения полей, перечисленных в ResultFields, в виде значения Variant или массива Variant. При неудаче возвращается значение Variant типа varNull.

Приведем пример поиска с помощью Lookup.

```
var
  V: Variant;
begin
  V := IBQuery1.Lookup('FIO', Edit1.Text,'Phone');
  if VarType(V) = varNull
  then ShowMessage ('Абонент не найден')
  else  ShowMessage (V);
end;
```

В данном примере производится поиск записи по фамилии абонента, введенной в поле Edit1, причем ФИО проверяется на полное соответствие. Если такое полное соответствие не найдется, метод Lookup возвращает нулевое значение типа Variant.

Метод Lookup не изменяет положения курсора. Он только возвращает значения указанных полей, которые затем могут использоваться любым способом. В частности, они могут использоваться вместо параметра KeyValues в другом операторе Lookup или Locate. Это открывает широкие возможности поиска значений в наборах данных.

2.3.6. Особенности программного редактирования данных

В п. 2.3.4 были рассмотрены методы, используемые для чтения значений полей записей НД. На практике часто возникает задача программного редактирования наборов данных, т.е. необходимо не только считывать значения полей записей НД, но также программно редактировать НД: добавлять и удалять записи, редактировать отдельные поля записей и т. д.

При этом следует помнить, что не каждый набор данных является редактируемым. Для того чтобы набор данных был редактируемым, должны быть выполнены определенные условия [4, 14]:

- данные отбираются только из одной таблицы или просмотра (view);
- таблица или просмотр допускают модификацию;
- в запросе не используется операнд distinct и агрегатные функции;
- в запросе не применяется соединение таблиц;
- в запросе отсутствуют подзапросы и вложенные запросы;

- не используется группирование данных;
- сортировка применяется только к индексированным полям.

Узнать, можно ли редактировать набор данных, можно с помощью свойства *CanModify* типа Boolean, которое принимает значение True для редактируемых наборов.

Класс TDataSet содержит ряд свойств и методов, которые обеспечивают редактирование набора данных.

В табл. 2.3 перечислены методы, используемые для редактирования наборов данных [6].

Таблица 2.3. Методы редактирования НД

Метод	*Назначение метода*
procedure Edit;	Перевод набора данных в режим редактирования (выполняется перед началом редактирования)
procedure Append;	Добавление новой пустой записи в конец набора данных. При использовании метода набор данных переходит в режим редактирования самостоятельно
procedure Insert;	Добавление новой пустой записи на место текущей, при этом текущая запись и все нижеследующие смещаются на одну позицию вниз. При использовании метода набор данных переходит в режим редактирования самостоятельно
procedure AppendRecord (const Values: array of const);	Добавление новой записи уже с заполненными полями Values в конец набора данных
procedure InsertRecord (const Values: array of const);	Добавление новой записи уже с заполненными полями Values на место текущей
procedure SetFields (const Values: array of const);	Заполнение заданными значениями Values всех полей для текущей записи
procedure ClearFields;	Очистка содержимого всех полей текущей записи. После использования метода поля становятся пустыми (NULL), а не сбрасываются в нулевое значение
procedure Delete;	Удаление текущей записи
procedure Refresh;	Обновление набора данных без закрытия и повторного открытия. Метод сработает только для тех таблиц и запросов, которые нельзя редактировать
procedure Post; virtual;	Сохранение сделанных изменений. Метод вызывается разработчиком самостоятельно или же вызывается самим набором данных при переходе на другую запись
procedure Cancel; virtual;	Отмена всех изменений, сделанных после последнего вызова метода Post

Пример добавления новой записи в таблицу абонентов учебной БД (компонент IBTable1 подключен к таблице Abonent).

```
IBTable1.AppendRecord(['445122', 3, 2, 48, 'Петров И. И.', '450865']);
```

После завершения работы метода набор данных автоматически возвращается в состояние просмотра.

При программном редактировании данных чаще всего имеет место не добавление или удаление записей, а изменение записей НД.

Методы FindField и FieldByName (см. п. 2.3.4) могут использоваться не только для чтения значений полей записей НД, но также и для присваивания произвольному полю текущей записи нового значения.

Для того чтобы записать значение в поле, оно должно допускать модификацию, а набор данных должен находиться в соответствующем режиме, например, редактирования или вставки [4]. Возможность модификации набора данных (всех его полей) в целом определяется свойствами *CanModify* и *Readonly* (только для чтения). При этом следует учесть, что даже если набор данных является модифицируемым, для отдельных полей этого набора редактирование может быть запрещено, и любая попытка изменить значение такого поля вызовет исключение. Возможность модификации данных в отдельном поле, также как и для НД в целом, определяется значением свойства *CanModify* типа Boolean.

При необходимости программист может запретить модификацию поля, а также скрыть его, используя свойства *Readonly* и *Visible* типа Boolean. Если поле является невидимым (свойство Visible установлено в False), но разрешено для редактирования (свойство Readonly установлено в False), то можно изменить значения этого поля программно [4].

О том, редактировалась ли текущая запись, можно узнать с помощью свойства *Modified* типа Boolean, если оно имеет значение True.

Рассмотрим пример редактирования записей набора данных, сформированного компонентом IBDataSet.

```
with IBDataSet1 do
begin
  First;
  while not Eof do
  begin
    Edit;
    FieldByName('Phone').AsString := '8(4912)' +
trim(FieldByName('Phone').AsString);
    Post;
    Next ;
  end;
end;
```

В данном примере сначала выполняется позиционирование на первой записи. Затем для каждой записи в цикле: вызывается метод Edit, который подготавливает буфер текущей записи для редактирования; изменяется значение

поля при помощи FieldByName, после чего редактирование завершается вызовом метода Post. В результате IBDataSet1 подставляет значения полей записи в ModifySQL и выполняет запрос.

В реальной практике подобная обработка большого множества записей не является хорошим решением. Правильным решением в данном случае является выполнение запроса UPDATE, который изменил бы все записи набора данных. Пример дан только для демонстрации метода Edit.

Далее рассмотрим программную работу с SQL-запросами, позволяющими реализовать реляционный способ доступа к данным как наиболее эффективный.

2.3.7. Реляционный способ доступа к данным с помощью SQL-запросов

Многие из рассмотренных до настоящего пункта в пособии операций с наборами данных (перемещение по записям набора данных и т. д.) относились к навигационному способу доступа к данным, который, как уже неоднократно было отмечено, не является рациональным для удаленных БД и таблиц с большим количеством записей.

Современное программирование приложений БД ориентировано на использование языка SQL, как для операций считывания данных из БД, так и для выполнения модификации данных.

Поскольку SQL не обладает возможностями полнофункционального языка программирования, а ориентирован на доступ к данным, его обычно интегрируют в средства разработки программ и используют в интерактивном режиме [1]. При этом для работы с SQL в среде разработки предлагаются соответствующие средства и компоненты. В Delphi к таким компонентам относятся наборы данных IBQuery, IBSQL и IBDataset (применительно к технологии доступа IBX).

В настоящем пункте рассматриваются особенности использования языка SQL для доступа к данным БД, как для операций считывания данных из БД, так и для выполнения модификации данных.

Статические и динамические запросы

В зависимости от способа формирования различают два вида SQL-запросов: *статические и динамические*.

Статический SQL-запрос формируется при разработке приложения и в процессе выполнения приложения не изменяется [4]. В статических запросах фиксировано все: имена таблиц, поля, константы в выражениях и т. п. Такой запрос обычно используется в случаях, когда код запроса заранее известен и во время работы приложения не требует модификации. Статические запросы чаще всего используются для выборки данных из таблиц-справочников. Например, статический запрос может быть использован для получения данных

об улицах из таблицы Street учебной БД с целью дальнейшего применения этих данных для подстановочных полей (подробно рассмотрено далее).

Динамический SQL-запрос формируется или изменяется при выполнении приложения. Такой запрос обычно применяется, если его текст зависит от действий пользователя.

Говоря о разделении запросов на статические и динамические, отдельно стоит отметить *запросы с параметрами*. При решении реальных задач очень редко используются полностью статические запросы, чаще для настройки статического SQL-запроса во время выполнения приложения в его тексте используются параметры. Обычно параметры применяются в тексте запроса вместо конкретных значений полей. Например, если необходимо выбрать информацию об одном конкретном абоненте из таблицы Abonent учебной БД, то необходимо использовать запрос с параметром.

Значения параметров передаются из внешних источников (обычно вводятся пользователем на форме приложения) и тем самым, не изменяя текст самого запроса, можно менять возвращаемый им результат [4]. Статические запросы, в которых использованы параметры, иногда также называют *изменяющимися* (динамическими) [4].

В некоторых источниках (например, [7, 8]) SQL-запросы с параметрами относят к разряду динамических, а в некоторых (например, [4]) – статических.

С SQL-запросами при разработке приложения можно работать с помощью разных редакторов, а также программными методами. Рассмотрим способы работы с основными типами SQL-запросов в приложении Delphi.

Разработка SQL-запросов с помощью редакторов

Как уже было отмечено, в Delphi для работы с SQL-запросами предназначены такие компоненты, как IBQuery, IBSQL и IBDataset. Для работы с выполняемыми ХП удобно использовать компонент IBStoredProc. Рассмотрим подробно работу с компонентом IBDataset, как с основным наиболее универсальным компонентом доступа к данным по технологии IBX.

Разработка запроса на выборку данных. В первую очередь, заполняем свойства Database и Transaction компонента IBDataset для установления его связи с БД. Для определения запроса на выборку данных используется свойство *SelectSQL*, имеющее тип TString. Это список строк, содержащих запрос SQL. В процессе проектирования приложения можно сформировать в этом свойстве окончательный запрос SQL, который будет осуществлять выборку необходимых данных, либо сформировать некоторый предварительный запрос SQL, который показал бы, с какими таблицами будет проводиться работа, и затем менять его программным способом в соответствии с алгоритмом работы приложения.

Рассмотрим работу с SQL-запросами в редакторе на примере работы со справочником улиц (таблица STREET) учебной БД.

Для задания запроса с помощью редактора необходимо дважды щелкнуть мышью по свойству SelectSQL, в результате чего откроется окно редактора запросов (рис. 2.6).

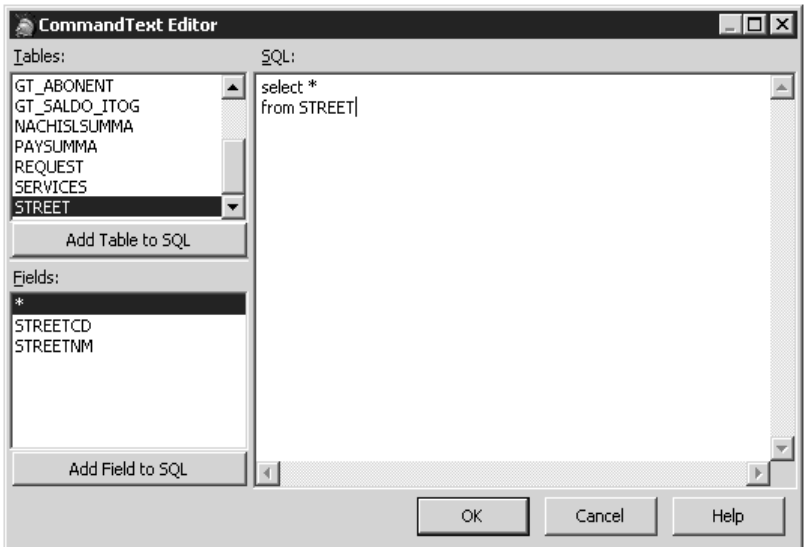


Рис. 2.6. *Окно редактора запросов*

В редакторе запросов доступны все таблицы БД и их поля. В данном случае в редакторе запросов написан простейший статический запрос – выборка данных по всем улицам из таблицы Street.

Для выполнения написанного запроса и отображения его результатов в визуальных компонентах необходимо установить свойство Active := True.

Для управления отображением данных у компонента IBDataset имеется Редактор полей [8]. Вызвать его можно или двойным щелчком на IBDataset, или щелчком правой кнопки мыши на IBDataset и выбором раздела Fields Editor из всплывающего меню. Используя Редактор Полей, можно выполнять следующие действия:

- добавлять имена полей, которые будут присутствовать в наборе данных (в набор данных не обязательно включаются все поля SQL-запроса [4]);
- задавать заголовки полей, отличающиеся от их имен (в основном используется для русификации столбцов, например, заголовок столбца Fio можно изменить на ФИО);
- делать определенные поля невидимыми (свойство Visible), не редактируемыми (свойство ReadOnly);

- задавать тексты, отображаемые при значениях true и false в логических полях;
- задавать формат отображения чисел;
- создавать вычисляемые поля, поля просмотра;
- задавать диапазоны значений и многое другое.

Рассмотрим, как с помощью редактора полей добавить в набор данных новое поле. Для добавления поля необходимо предварительно задать запрос на выборку данных, затем вызвать редактор полей, щелкнуть правой кнопкой мыши, выбрать пункт меню New field… и задать имя и тип поля. Команда New field позволяет создать объект-поле, которое реально не существует в структуре данных исходного набора данных (поля исходного набора добавляются командами Add Fields или Add All Fields). Для выбора типа поля используется группа радиокнопок Field Type:

Data – поле данных;
Calculated – вычисляемое поле;
Lookup – подстановочное поле.

На рис. 2.7 показано, как можно создать подстановочное поле StreetNM для набора данных, выводящего данные из таблицы Abonent, для отображения наименований улиц вместо их кодов.

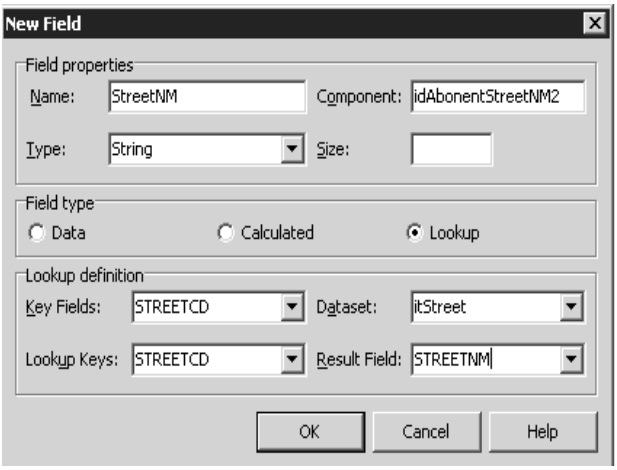


Рис. 2.7. *Окно создания подстановочного поля*

Как уже было отмечено, на практике чаще всего используются запросы с параметрами. *Параметр* – это специальная переменная, перед именем которой ставится двоеточие в тексте запроса [4].

: <имя параметра>.

Двоеточие не является частью имени параметра и ставится только в тексте запроса. Как и для обычных переменных программы, в процессе выполнения приложения вместо параметра подставляется его значение. Параметры удобно использовать для передачи в текст SQL-запроса внешних значений.

Например, в редакторе запросов компонента IBQuery можно записать SQL-запрос с условием:

```
select *
from Street
where StreetNM like :StrName;
```

Система Delphi автоматически учитывает все указанные в SQL-запросе параметры в специальном списке параметров [4]. Это значение свойства *Params*, представляющее собой массив заданных параметров. Свойство позволяет получить доступ к каждому параметру, как при разработке, так и при выполнении приложения.

На этапе разработки приложения можно вызвать Редактор параметров двойным щелчком в Инспекторе Объектов по свойству Params и установить свойства параметров (тип параметра, значение по умолчанию и т. д.) на этапе разработки.

Для компонента IBDataset работа с параметрами производится только программным путем.

Работа с запросами, в том числе с параметризированными запросами, программными методами будет рассмотрена далее.

Мы уже показали заполнение свойства SelectSQL компонента IBDataSet (см. рис. 2.6). Это свойство содержит запрос на выборку записей, однако часто требуется выполнять модификацию данных с помощью SQL-запросов.

Разработка модифицирующих запросов. Помимо выборки данных из БД и представления их пользователю в наглядном виде при решении реальных задач практически всегда необходимо предоставить возможность редактирования данных пользователем и последующее сохранение изменений в БД. Для редактирования данных БД (добавления, изменения, удаления записей) используются модифицирующие запросы (INSERT, UPDATE, DELETE).

TIBDataSet был спроектирован в первую очередь для того, чтобы использоваться совместно со стандартными визуальными компонентами, и он предоставляет средства для автоматического выполнения тех модифицирующих запросов, которые необходимы для изменения данных, полученных при помощи SelectSQL [11]. К таким средствам относятся свойства InsertSQL, ModifySQL, DeleteSQL, RefreshSQL.

Задать свойства можно вручную (прописав запрос в окне для конкретного свойства), или автоматически [12].

Рассмотрим особенности свойств для модификации данных на примере работы со свойством ModifySQL.

Допустим, в визуальном компоненте TDBGrid получен результат выполнения запроса на выборку улиц (код улицы StreetCD и наименование улицы StreetNM) и выполняется попытка редактирования первой записи, например, изменение наименования улицы. В момент изменения записи произойдет следующее: в локальном буфере IBDataSet у текущей записи значение поля StreetNM будет изменено со старого на новое. Однако в БД данное изменение пока никак не отразится. Чтобы произвести фактическое изменение данных на сервере, необходимо выполнить соответствующий запрос UPDATE. Этот запрос необходимо заранее указать у компонента IBDataSet в свойстве *ModifySQL*:

```
update STREET
set
  STREETCD = :STREETCD,
  STREETNM = :STREETNM
where
  STREETCD = :OLD_STREETCD
```

Как видно из примера, вместо реальных значений в этом запросе указаны параметры (:STREETCD и :STREETNM), названия которых совпадают с названием реальных полей. Таким образом, когда пользователь изменит значения полей конкретной записи, IBDataSet сам задаст значения всех параметров, взяв их из соответствующих полей. Запрос из ModifySQL выполнится, и только после этого изменения, сделанные пользователем, окажутся в БД. Аналогичная последовательность действий связана с запросами в свойствах InsertSQL и DeleteSQL – они выполняются при вставке новой записи и удалении записи пользователем.

Следует обратить внимание на префикс OLD_ в названии параметра: OLD_StreetCD.

У IBDataSet есть специальные параметры, которые могут быть использованы в запросах [12]. Параметры именуются автоматически, и соответствуют именам столбцов, предваряемые префиксами OLD_ и NEW_. В столбцах OLD_ хранятся данные текущего столбца до модификации записи, а в NEW_ хранятся те, которые ввел пользователь при редактировании.

В предыдущем запросе UPDATE префикс OLD_ означает, что IBDataSet должен подставить в параметр значение поля до изменения пользователем. Таким образом, если будут сформированы все модифицирующие запросы, то IBDataSet позволит пользователям редактировать данные, т. е. мы фактически получим запрос, который разработчики на Delph обычно называют "живым" [11].

Существует еще одна важная особенность при создании модифицирующих запросов. После выполнения любого модифицирующего действия IBDataSet выполнит запрос, указанный в свойстве *RefreshSQL*. Этот запрос

должен возвращать только одну запись – текущую и нужен для обновления значений полей текущей записи после сделанных исправлений.

Пример запроса для свойства RefreshSQL приведен ниже.

```
select
  STREETCD,
  STREETNM
from STREET
where
  STREETCD = :STREETCD
```

Смысл данного запроса становится очевидным, если допустить существование в БД триггеров для таблицы Street, которые модифицируют значения полей. Поскольку изменения происходят в самой БД сразу после вставки или после изменения записей, то без повторного перечитывания записи (без Select только что вставленной или измененной записи), мы не узнаем о тех изменениях, которые были сделаны в триггерах. Можно, конечно, вообще переоткрыть весь запрос (такой механизм и реализуется в BDE, например), заданный в SelectSQL, однако без этого можно обойтись, используя RefreshSQL и значительно сэкономив при этом сетевой трафик и снизив нагрузку на сервер, поскольку получение всего лишь одной измененной записи гораздо более эффективно, чем переоткрытие запроса целиком [11].

IBX предоставляет возможность быстро сгенерировать необходимые модифицирующие запросы при помощи редактора IBDataSet (рис. 2.8).

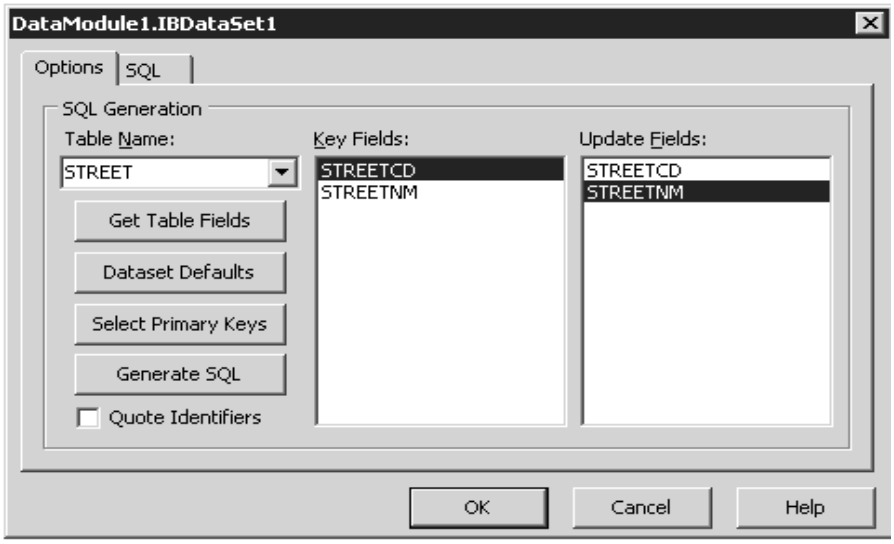


Рис. 2.8. *Генератор модифицирующих запросов*

Редактор вызывается нажатием правой кнопки на компоненте и выбором пункта DataSet Editor [12].

Базовая информация (имя таблицы, столбцы), уже получена из SelectSQL.

В диалоге нужно указать столбцы первичного ключа (key fields) и обновляемые столбцы (update fields).

В списке Key Fields нужно выделить те поля, которые будут формировать условие WHERE в запросах. Очевидно, что это должны быть поля, которые определяют первичный ключ у таблицы. Если такой ключ существует для выбранной таблицы, то можно просто нажать на кнопку Select Primary Keys, чтобы автоматически выделить нужные поля.

В списке Update Fields необходимо выделить те поля, которые потом пользователь сможет редактировать. Как правило, столбцы первичного ключа не обновляются, поэтому их исключают из UpdateFields [12]. CALCULATED-поля не включаются в данный список, так как их значения нельзя менять, при этом изначально разработчик должен сам точно представлять, какие поля необходимо исключать из модифицирующих запросов, так как компоненты IBX не дают такой подсказки [11].

В заключение необходимо нажать кнопку Generate SQL, чтобы получить все запросы lnsertSQL, ModifySQL, DeleteSQL и RefreshSQL. Диалоговое окно с четырьмя запросами, сгенерированными автоматически, представлено на рис. 2.9.

Рис. 2.9. *Окно с модифицирующими запросами*

Далее рассмотрим программные методы работы с запросами, которые включают в себя методы для определения запроса и задания его параметров, открытие/закрытие запроса и т. д.

Работа с SQL-запросами программными методами

При решении реальных практических задач огромное значение имеет работа с SQL-запросами на этапе выполнения приложения. Программным путем формируются динамические запросы, задаются значения параметров и т. д. Рассмотрим подробно работу с запросами программными методами на примере компонента IBDataset.

При работе с запросами практически всегда требуется менять состояние набора данных (закрыт/открыт). Обычно перед вызовом методов работы с запросами выполняется закрытие набора (либо компонент делается неактивным), а после завершения работы с запросами – открытие набора данных (для наборов, возвращающих набор записей).

Все свойства вида *xxxSQL*, описанные выше, имеют тип String и во время выполнения приложения могут формироваться программно методами, обычными для класса TString [7]. Среди таких методов основное значение имеют методы *Clear* (очистка) и *Add* (добавление строки).

Например, программно задать запрос на выборку абонентов из таблицы Abonent учебной БД можно таким образом:

```
IBDataset1.Close;
IBDataset1.SQL.Clear;
IBDataset1.SQL.Add('SELECT *');
IBDataset1.SQL.Add('FROM Abonent');
IBDataset1.SQL.Add('WHERE Phone = ' + Edit1.Text);
IBDataset1.Open;
```

В данном случае организовано формирование и выполнение запроса, который выводит информацию по абоненту с номером телефона, указанным в поле ввода Editl, которое размещено на форме приложения. Текст из поля ввода (Edit1.Text) подставляется в текст запроса.

Как уже было отмечено, для передачи в текст SQL-запроса внешних значений удобно использовать параметры [4]. С помощью параметров задачу поиска абонента по телефону можно решить проще. Для этого необходимо в свойстве SelectSQL на этапе разработки приложения написать запрос:

```
SELECT *
FROM Abonent
WHERE Phone = :PhoneNom
```

В последующем перед выполнением запроса вместо параметра необходимо подставить его значение, в данном случае из редактора Editl.

Программный доступ к параметрам во время выполнения приложения осуществляется аналогично доступу к полям набора данных [8]. Параметры являются объектами типа TParams, образующими массив *Params*, к элементам которого можно обращаться по индексу (номеру). Значения параметров, как и значения полей, определяются такими свойствами объектов-параметров, как Value, AsString, Aslnteger и т. п.

Например, к параметру PhoneNom получить доступ можно так:

```
IBDataset1.Params[0].AsString  := Edit1.Text;
```

Последовательность, в которой располагаются параметры в свойстве Params, определяется последовательностью их упоминания в запросе SQL. Часто, если запрос сложный и имеет множество параметров, использование данного метода может привести к ошибкам из-за неверного определения индекса параметра.

Для доступа к параметру во время выполнения приложения чаще используется метод *ParamByName*, который не требует запоминания индексов, и отличается от аналогичного метода FieldByName только тем, что вместо имени поля указывается имя параметра.

Ниже приведен код, в котором выполняется присваивание значения параметру PhoneNom для реализации поиска абонента по заданному номеру телефона.

```
IBDataset1.Close;
IBDataset1.ParamByName('PhoneNom').AsString := Edit1.Text;
IBDataset1.Open;
```

Следует помнить, что задание нового значения параметру само по себе еще не обеспечивает влияния на возвращаемый из запроса результат. Надо повторно выполнить данный запрос.

Реализация задачи поиска абонента путем использования параметра отличается от реализации путем полного формирования запроса (приведенного в самом начале пункта) не только тем, что код с использованием параметра короче.

При использовании параметров можно не изменять текст SQL-запроса во время выполнения приложения и, тем не менее, передавать в него различные значения. То есть статический запрос как бы превращается в динамический.

Такая работа с динамическими запросами намного легче, особенно, если запросы большие, а изменяется только значение какого-то одного поля, а не общая структура запроса [7]. Использование переменной-параметра в таком случае предпочтительнее и с точки зрения компактности программного кода, и с точки зрения производительности. Серверы БД перед выполнением нового запроса производят его компиляцию, что отнимает время. Если в тексте запроса изменилось что-либо, будет происходить компиляция. При использовании переменных запрос не изменяется, потому что изменилось только значение переменной и лишних затрат на перекомпиляцию не будет.

Обычно текст SQL-запроса проверяется и выполняется при каждом открытии набора данных. Если текст запроса при выполнении приложения не изменяется, то его можно предварительно подготовить, а после этого только использовать такой подготовленный к выполнению запрос. Это позволяет ускорить обработку статических запросов, в том числе имеющих параметры.

Подготовку запроса к выполнению осуществляет метод *Prepare*, который можно вызывать при создании формы. Чтобы определить, была ли произведена предварительная подготовка запроса, необходимо проанализировать свойство *Prepared* типа Boolean, которое после вызова метода Prepare устанавливается в значение True.

Если текст подготовленного к выполнению запроса изменился (к изменению значений параметров это не относится), то автоматически вызывается метод *OnPrepare*, и свойство Prepared устанавливается в False.

Несмотря на предпочтительность параметризации запросов по сравнению с их динамическим определением, при программировании реальных приложений часто имеют место ситуации, когда необходимо выполнять полностью динамическое формирование запроса.

Допустим, необходимо выполнять сортировку данных об абонентах, получаемых из таблицы Abonent, по номеру лицевого счета (поле AccountCD) либо по ФИО (поле FIO) в зависимости от условий, заданных пользователем. Пользователь управляет выбором поля для сортировки с помощью группы переключателей RadioGroupl, а в RadioGroup2 выбирает порядок сортировки. По щелчку на кнопке Button1, расположенной на форме Form1, данные должны быть отсортированы. Обработчик события OnClick кнопки приведен ниже.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  sort: string;
begin
  IBDataSet1.Close;
  IBDataSet1.SelectSQL.Clear;
  IBDataSet1.SelectSQL.Add('SELECT * FROM Abonent');
  case RadioGroup1.ItemIndex of
  0: sort := 'ORDER BY AccountCD';
  1: sort := 'ORDER BY FIO';
  end;
  case RadioGroup2.Itemindex of
  0:;
  1: sort := sort + ' DESC';
  end;
  IBDataSet1.SelectSQL.Add(sort);
  IBDataSet1.Open;
end;
```

В данном примере осуществляется формирование динамического запроса для управления сортировкой данных: в тексте запроса изменяются названия полей и добавляется или исключается слово desc. Конкретный вид запроса формируется в зависимости от условий, заданных пользователем.

Все описанные до настоящего момента в пункте примеры демонстрировали работу с запросами на выборку данных. Кратко рассмотрим на примере работы с компонентом IBDataSet, как программными методами можно редактировать данные.

Ранее уже отмечалось, что TIBDataSet порожден классом TDataSet, а значит, наследует основные принципы его работы. У TIBDataSet как наследника TDataSet существует три основных метода для изменения данных: *Delete, Insert (Append)* и *Edit* [11].

Предположим, что есть редактируемый IBDataSetl, возвращающий данные по улицам из таблицы Street, и необходимо вставить новую запись в справочник. Ниже приведен пример кода для вставки новой записи программным методом.

```
with IBDataSet1 do
begin
  Insert;
  FieldByName('StreetCD').AsInteger := 10;
  FieldByName('StreetNM').AsString := 'НОВАЯ УЛИЦА';
  Post;
end;
```

При выполнении данного кода после вызова метода Insert, IBDataSet1 формирует пустой буфер для новой (пока еще не введенной) записи. Затем задаются значения нужных полей при помощи вызовов метода FieldByName. Подтверждается редактирование вызовом метода Post. В этот момент IBDataSet1 выполняет запрос, прописанный в свойстве InsertSQL, подставив вместо параметров те значения полей, которые в приведенном примере заданы непосредственно в коде программы. Если запрос выполнился успешно, то IBDataSet1 автоматически выполняет RefreshSQL для обновления только что вставленной записи – для проверки изменений, внесенных на стороне БД.

Аналогичным образом, но с использованием метода Edit вместо Insert, выполняется программное редактирование записей.

2.3.8. Режимы наборов данных

В процессе своего функционирования (от открытия методом Open и до закрытия методом Close) набор данных, как можно было понять из предыдущих пунктов, выполняет самые разнообразные операции. В любой момент времени набор данных находится в некотором состоянии (режиме), т. е. подготовлен к выполнению действий строго определенного рода [6].

Текущий режим набора данных определяется свойством *State* типа TDataSetState. Оно доступно для чтения во время выполнения приложения и может быть использовано только для текущего режима. Для перевода набора данных в требуемый режим используются специальные методы. Они могут вызываться явно (указанием имени метода) или косвенно (путем управления соответствующими визуальными компонентами, например, навигатором DBNavigator или сеткой DBGrid) [4].

Все режимы набора данных делятся на две группы.

1. ***Режимы, в которые набор данных переходит автоматически***, а также непродолжительные по времени режимы, сопровождающие функционирование полей набора данных:

dsOpening (открытие набора данных). В этот режим НД переходит при открытии методом Open или свойством Active.

dsBlockRead (состояние просмотра по блокам). Включает механизм ускоренного перемещения по большим массивам записей без показа промежуточных записей в компонентах отображения данных и без вызова обработчика события перемещения по записям [6]. Для реализации быстрого перемещения по набору данных можно использовать методы DisableControls и EnableControls.

dsFilter (фильтрация записей). В этот режим набор данных автоматически переходит из режима dsBrowse (описан далее) каждый раз, когда выполняется обработчик события OnFilterRecord. В режиме блокируются все попытки изменения записей. После завершения работы обработчика события OnFilterRecord набор данных автоматически переводится в режим dsBrowse [4].

dsNewValue (обращение к значению свойства TField.NewValue). Включается при обращении к свойству NewValue поля набора данных, которое представляет собой измененное, но еще не сохраненное в базе значение [7].

dsOldValue (обращение к значению свойства TField.OldValue). Включается при обращении к свойству OldValue поля набора данных, которое представляет собой старое значение (до внесения изменений), прочитанное из БД [7].

dsCurValue (обращение к значению свойства TField.CurValue). Включается при обращении к свойству CurValue поля набора данных, которое представляет собой текущее значение поля, включая изменения, сделанные другим пользователем [7].

dsinternalCalc (указание на необходимость вычислять значения полей, свойство TField.FieldKind которых имеет значение fklnternalCalc).

dsCalcFields (расчет вычисляемых полей). Включается при выполнении метода onCalcFields.

Следует отметить, что режимы данной группы относятся к служебным (набор данных использует эти режимы для собственных нужд) и обычно при программировании не требуется их анализ [8, 11]. Практический интерес представляют режимы, которыми можно управлять из приложения.

2. ***Режимы, которыми можно управлять из приложения***. Для управления режимами набора данных используются методы open, Close, Edit, Insert. К данной группе относятся пять режимов:

dsinactive (неактивен). Набор данных закрыт и доступ к его данным невозможен. В этот режим набор данных переходит после своего закрытия методом Close, когда свойство Active установлено в значение False.

dsBrowse (навигация по записям набора данных и просмотр данных, недоступность для редактирования). В этот режим набор данных переходит:

- из режима dsinactive при установке свойства Active в значение True;
- из режима dsEdit при вызове метода Post или Cancel;
- из режима dsInsert при вызове метода Post или Cancel.

dsEdit (редактирование текущей записи). В этот режим набор данных переходит из режима dsBrowse при вызове метода Edit.

dsinsert (вставка новой записи). В этот режим набор данных переходит из режима dsBrowse при вызове методов Insert, insertRecord, Append или AppendRecord.

dsSetKey (поиск записи, удовлетворяющей заданному критерию). В этот режим набор данных переходит из режима dsBrowse при вызове методов SetKey, SetRangeXXX, FindKey, GotoKey, FindNearest или GotoNearest. Данный режим возможен только для компонента IBTable, так как для компонента IBQuery отбор записей осуществляется средствами языка SQL.

Рассмотрим, как изменяется состояние набора данных при выполнении стандартных операций.

Взаимосвязи между основными режимами наборов данных, а также некоторые методы и свойства, с помощью которых набор данных переходит из одного режима в другой, показаны на рис. 2.10.

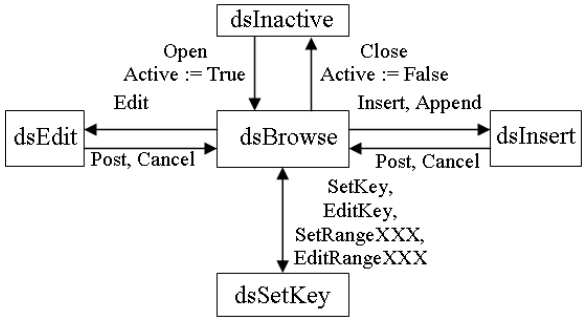


Рис. 2.10. *Взаимосвязи основных режимов набора данных*

Закрытый набор данных всегда имеет неактивное состояние *dsinactive*.

При открытии набор данных переходит в состояние просмотра данных *dsBrowse*. В этом состоянии по записям набора данных можно перемещаться и просматривать их содержимое, но редактировать данные нельзя. Это основное

состояние открытого набора данных, из него можно перейти в другие состояния, но любое изменение состояния происходит через просмотр данных.

При необходимости редактирования данных набор должен быть переведен в состояние редактирования *dsEdit*, для этого используется метод *Edit*. После выполнения метода можно изменять значения полей для текущей записи. При перемещении на следующую запись набор данных автоматически переходит в состояние просмотра.

Для того чтобы вставить в набор данных новую запись, необходимо использовать состояние вставки *dsinsert*. Метод *Insert* переводит набор данных в это состояние и добавляет на месте текущего курсора новую пустую запись. При переходе на другую запись, после проверки на уникальность первичного ключа (если он есть) набор данных возвращается в состояние просмотра.

Иногда при описании операций, выполняемых с записями набора данных, под режимом редактирования подразумевается не только режим *dsEdit* изменения полей текущей записи, но и режим *dsinsert* вставки новой записи. Тем самым режим редактирования понимается в широком смысле слова как режим модификации набора данных [4].

Состояние установки ключа *dsSetKey* используется только в табличных компонентах при необходимости поиска методами *FindKey* и *FindNext*, а также при использовании диапазонов (метод setRange). Это состояние сохраняется до момента вызова одного из методов поиска по ключу или метода отмены диапазона. После этого набор данных возвращается в состояние просмотра. Как уже было отмечено ранее, в пособии подробно не рассматриваются методы, применимые только для табличных компонентов, в том числе методы, переводящие набор данных в режим *dsSetKey*.

При выполнении программы можно определить режим набора данных путем обращения к свойству *State* типа TDataSetState самого набора данных и связанного с ним источника данных DataSource [4]. При изменении режима набора данных для источника данных DataSource генерируется событие *OnStateChange* типа TNotifyEvent.

Ниже приведен пример, где в обработчике события OnStateChange компонента DataSource1 определяется режим набора данных, связанного с источником данных DataSourcel, и информация об этом режиме выводится в надписи Label1 [4].

```
procedure TForml.DataSource1StateChange(Sender: TObject);
begin
  case DataSourcel.State of
    dsinactive: Label1.Caption := 'Набор данных закрыт';
    dsBrowse: Label1.Caption := 'Просмотр набора данных';
    dsEdit: Label1.Caption := 'Редактирование набора данных';
    dsInsert: Label1.Caption := 'Вставка записи в набор данных'
  else Label1.Caption := 'Режим набора данных не определен';
  end;
end;
```

Контрольные вопросы

1. Перечислите и кратко опишите технологии доступа к данным, реализованные в Delphi. Существуют ли не встроенные в Delphi наборы компонентов доступа для работы с БД InterBase и Firebird?

2. Каким образом организуется доступ к данным с помощью компонентов InterBase Express? Перечислите основные компоненты технологии InterBase Express.

3. Для каких целей в Delphi используются модули данных? Какие компоненты можно разместить на модуле данных? Какие существуют типы модулей данных?

4. Как реализуются наборы данных в приложениях БД?

5. Чем отличаются навигационный и реляционный способы доступа к данным?

6. В каких состояниях могут находиться наборы данных в приложении? С помощью каких свойств можно управлять состояниями наборов данных?

7. Какие свойства и методы отвечают за организацию навигационного перемещения по записям набора данных?

8. Какими объектами в приложении представляется поле набора данных? Какие методы отвечают за доступ к полям набора данных?

9. Какие методы отвечают за организацию поиска данных среди записей наборов данных?

10. Перечислите условия, которые должны быть выполнены для того, чтобы набор данных был редактируемым. С помощью какого свойства можно узнать, является ли набор данных редактируемым?

11. Какие методы используются для редактирования наборов данных?

12. Какие компоненты могут использоваться в приложениях для работы с данными с помощью SQL? Как формируются статический и динамический запросы?

13. Опишите особенности разработки запросов с помощью редактора запросов (на примерах разработки запроса с параметрами и модифицирующего запроса).

14. Опишите особенности работы с запросами программными методами (на примере формирования динамического запроса).

15. Опишите режимы, в которых могут находиться наборы данных, и их взаимосвязи.

3. Источники данных TDataSource

В первом разделе настоящего пособия, при рассмотрении общей структуры приложений БД, был описан компонент-источник данных TDataSource и его роль в приложении БД.

В данной главе еще раз повторим некоторые ключевые особенности данного компонента, а также рассмотрим некоторые свойства и события, характерные для компонента TDataSource, приведем примеры использования.

Источник данных TDataSource используется как промежуточное звено между набором данных и визуальными компонентами, с помощью которых пользователь управляет этим набором данных [4]. Компонент расположен на странице Data Access Палитры компонентов Delphi и является общим для всех технологий доступа.

Рассмотрим основные свойства компонента TDataSource.

Для указания набора данных, с которым связан источник данных, служит свойство *DataSet* типа TDataSet последнего. Визуальные компоненты связаны с источником данных через свои свойства DataSource. Обычно связь между источником и набором данных устанавливается на этапе проектирования в Инспекторе объектов, однако при необходимости эту связь можно установить или разорвать динамически.

Пример установки для компонента DataSource1 набора данных IBTable1:

```
DataSource1.DataSet := IBTable1;
```

При смене у компонента DataSource исходного набора данных (свойство DataSet) визуальные компоненты, связанные с этим источником данных DataSource1, автоматически подключаются к новому набору данных.

Для анализа состояния, в котором находится набор данных, можно использовать свойство *State* типа TDataSetState. При каждом изменении состояния набора данных для связанного с ним источника данных DataSource генерируется событие *OnStateChange* типа TNotifyEvent. Пример использования данного свойства был рассмотрен ранее в пункте 2.3.8 при описании режимов наборов данных.

Если компонент DataSource связан с редактируемым набором данных, то может использоваться свойство *AutoEdit* типа Boolean, которое определяет, может ли набор данных автоматически переводиться в режим модификации при выполнении пользователем определенных действий [4]. При использовании визуальных компонентов DBGrid и DBEdit таким действием, например, является нажатие алфавитно-цифровой клавиши, когда компонент находится в фокусе ввода. По умолчанию свойство AutoEdit = True, и автоматический переход в режим модификации разрешен. Если необходимо защитить данные от случайного изменения, то одной из предпринимаемых мер является установка свойства AutoEdit в значение False.

При этом следует учесть, что значение свойства AutoEdit влияет на возможность редактирования набора данных только со стороны пользователя. Программно можно изменять записи независимо от значения этого свойства.

Без учета значения свойства AutoEdit пользователь может переводить набор данных в режим модификации путем нажатия кнопок компонента DBNavigator.

Иногда в визуальных компонентах требуется отключать отображение полей записей набора данных, например, при переборе записей в цикле их обработки, поскольку при этом возникает мелькание данных вследствие их быстрой смены. Для управления отображением записей можно использовать свойство *Enabled* типа Boolean источника данных. Назначение свойства фактически аналогично назначению методов *DisableControls* и *EnableControls* набора данных, которые были рассмотрены ранее.

Ниже приведен пример использования свойства Enabled.

```
DataSource1.Enabled := False;
IBQuery1.First;
for n := 1 to IBQuery1.RecordCount do
begin
  // Обработка записи набора данных IBQuery1
  IBQuery1.Next;
end;
DataSource1.Enabled := True;
```

В приведенном примере перед началом цикла, в котором перебираются записи набора данных, отображение записей в визуальных компонентах отключается, а после цикла – включается.

При использовании компонента TDataSource генерируются следующие события [7]:

OnStateChange – событие возникает, когда состояние связанного с TDataSource набора данных изменилось;

onDataChange – возникает, когда данные в наборе данных были изменены или при переходе к другой записи, а также при открытии набора данных. Событие можно использовать, например, для контроля положения указателя текущей записи и выполнения действий, связанных с его перемещением;

onUpdateData – возникает непосредственно перед записью данных в БД, когда данные в текущей строке должны быть обновлены. В обработчике события можно предусмотреть дополнительный контроль и обработку введенных в поля значений, а также некоторые другие действия, например, отказ от изменения записи.

Рассмотрим пример использования события OnDataChange компонента DataSource. Предположим, что компонент связан с некоторым редактируемым набором данных IBTable1, и на форме установлен компонент для редактирования данных, например DBGrid1, и кнопка Button1 сохранения изменений. Необходимо менять доступность кнопки в зависимости от того, были ли изменены данные пользователем. Пользователь может изменять данные не-

посредственно в визуальном компоненте DBGrid1, и при изменении данных кнопка должна стать доступной. Создаем обработчик события OnDataChange:

```
procedure TDataModule1.DataSource1DataChange(Sender: TObject; Field:
TField);
begin
  Button1.Enabled:=IBTable1.Modified;
end;
```

Если набор данных изменен, то свойство Modified набора данных IBTable1 вернет True, и кнопка станет доступной, иначе будет недоступной.

В обработчике события нажатия кнопки пишем следующий код:

```
IBTable1.Post;
Button1.Enabled: = IBTable1.Modified;
```

В данном обработчике сначала сохраняются данные, а затем меняется доступность кнопки.

Таким образом, как уже было отмечено ранее, основная роль компонента TDataSource заключается в управлении потоками данных между набором данных и связанными с ним компонентами отображения данных.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие задачи в приложениях БД выполняет источник данных TDataSource?
2. Каким образом компонент TDataSource осуществляет синхронизацию поведения компонентов отображения данных с состоянием набора данных?
3. К какой технологии доступа к данным относится компонент TDataSource? На какой закладке в Delphi он расположен?
4. Какое свойство компонента TDataSource отвечает за связь с набором данных? Что происходит при смене у компонента исходного набора данных?
5. Какое свойство компонента TDataSource отвечает за возможность редактирования связанного набора данных пользователем?
6. Какое свойство компонента TDataSource управляет отображением записей набора данных?
7. Какие события генерируются при использовании компонента TDataSource?
8. Приведите пример программного анализа состояния, в котором находится набор данных, в обработчике события компонента TDataSource.
9. В каком случае возникает событие onDataChange компонента TDataSource? Приведите пример обработчика события onDataChange.
10. Приведите пример обработчика события компонента TDataSource, которое возникает непосредственно перед записью данных в базу данных.

4. Визуальные компоненты отображения и редактирования данных

До настоящей главы в пособии были рассмотрены вопросы организации доступа к данным в приложениях БД, создание наборов данных и т. д. Рассмотрим более подробно компоненты, отвечающие за отображение данных в приложениях (визуальные компоненты).

Отображение данных обеспечивает достаточно представительный набор компонентов VCL Delphi [6]. Визуальные компоненты для работы с данными расположены на странице Data Controls Палитры компонентов и предназначены для построения интерфейсной части приложения. Они, также как и компоненты вкладки Data Access, используются независимо от используемой технологии доступа к данным [7].

4.1. Обзор визуальных компонентов

Визуальные компоненты используются для навигации по набору данных, а также для отображения и редактирования записей. Для построения графиков и диаграмм используется компонент TDBChart с закладки TeeChart. Визуальные компоненты для работы с данными представлены в табл. 4.1.

Таблица 4.1. Визуальные компоненты

Компонент	***Изображение***	***Описание***
TDBGrid		Компонент-таблица (сетка) для представления записей набора данных
TDBNavigator		Компонент для перемещения (навигации) по записям набора данных
TDBText		Компонент для представления значения текстового поля
TDBEdit		Компонент представляет собой стандартный однострочный текстовый редактор для доступа к данным поля набора данных
TDBMemo		Компонент для представления значений мемо-полей (многострочный редактор)

Окончание табл. 4.1

TDBImage		Компонент для представления графических изображений
TDBListBox		Компонент для выбора значения поля БД из списка
TDBComboBox		Компонент для выбора значения поля БД из списка и произвольного ввода значения (комбинированный список)
TDBCheckBox		Компонент-флажок для представления значения логического поля набора данных
TDBRadioGroup		Компонент для предоставления фиксированного набора возможных значений поля при помощи группы зависимых переключателей
TDBLookupListBox		Компонент-список, формируемый по полю другого набора данных
TDBLookupComboBox		Компонент-список (комбинированный), формируемый по полю другого набора данных
TDBRichEdit		Компонент предоставляет возможности полноценного текстового редактора для просмотра и изменения текстовых данных поля набора данных
TDBCtrlGrid		Компонент для представления записей набора данных на наборе панелей (модифицированная сетка)
TDBChart		Компонент для построения графиков и диаграмм

Все визуальные компоненты для работы с данными можно разделить на три группы [6]:

- компоненты для работы с набором записей;
- компоненты для работы с отдельными полями;
- компоненты для навигации по данным.

На рис. 4.1 представлена схема, отражающая классификацию визуальных компонентов для работы с данными.

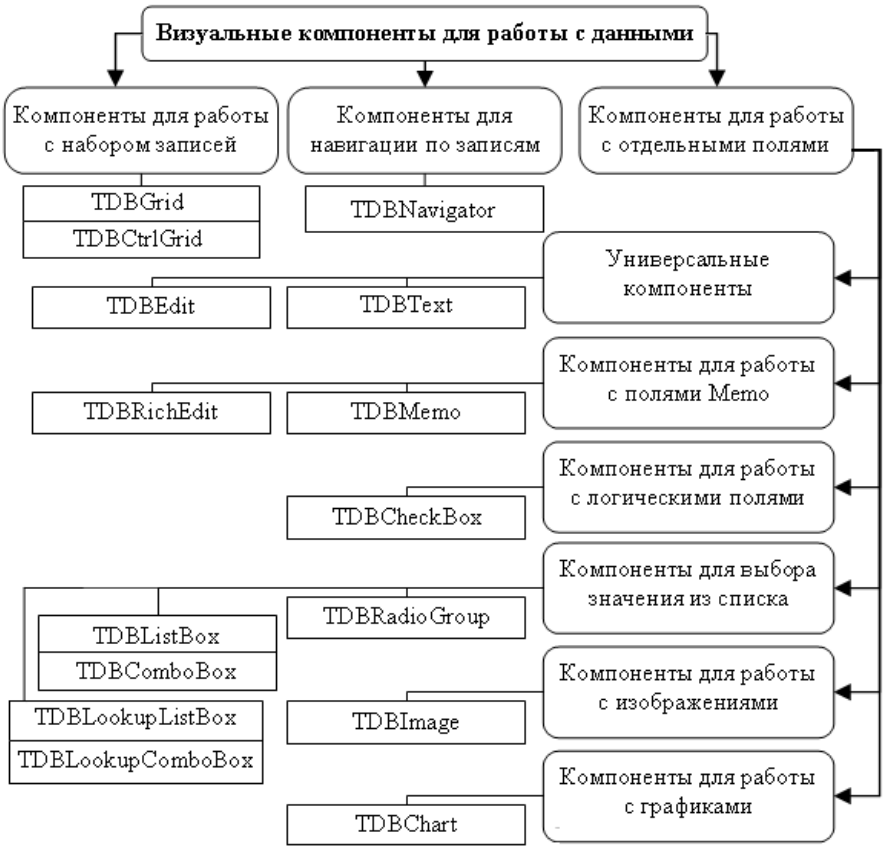


Рис. 4.1. *Классификация компонентов отображения данных*

Визуальные компоненты похожи на соответствующие компоненты страниц Standard, Additional и Win32 и отличаются, в основном, тем, что ориентированы на работу с БД и имеют дополнительные свойства для связи с наборами данных. Часто визуальные компоненты называют элементами, чувствительными к данным [4].

Компоненты для представления наборов данных обеспечивают просмотр наборов данных целиком или в произвольном сочетании полей. К таким компонентам относятся TDBGrid и TDBCtrlGrid, выводящие записи набора данных в табличном виде. Основное свойство данных компонентов, характерное и для всех остальных визуальных компонентов работы с данными, – свойство *DataSource*. Свойство предназначено для соединения с набором данных через источник данных – компонент TDataSource.

Большинство визуальных компонентов предназначены для работы с отдельным полем, т. е. при перемещении по записям набора данных такие компоненты отображают и позволяют редактировать значение только одного поля текущей записи [4, 6]. Помимо свойства DataSource для таких компонентов характерно свойство *DataField,* которым задается поле связанного с компонентом набора данных. Свойство Field обеспечивает доступ к классу TField поля, заданного свойством DataField.

К компонентам для работы с отдельным полем относятся, например, однострочный редактор DBEdit, графическое изображение DBimage и др.

Следует отметить, что для всех визуальных компонентов, предназначенных для отображения и редактирования значений полей, при изменении пользователем их содержимого набор данных автоматически переводится в режим редактирования. Произведенные с помощью этих компонентов изменения также автоматически сохраняются в связанных с ними полях при наступлении определенных событий, таких, например, как потеря фокуса визуальным компонентом или переход к другой записи набора данных [4].

При программном изменении содержимого этих визуальных компонентов набор данных не переводится в режим редактирования автоматически. Для этой цели в коде должен предварительно вызываться метод Edit набора данных. Чтобы сохранить изменения в поле (полях) текущей записи, программист также должен предусмотреть соответствующие действия, например, вызов метода Post или переход к другой записи набора данных.

Все замечания относительно редактирования набора данных имеют смысл, если для набора данных установлено свойство *Readonly* (режим "только для чтения") = False.

Особенную роль среди компонентов отображения данных играет компонент TDBNavigator. Он не показывает данные и не предназначен для их редактирования, зато обеспечивает навигацию по набору данных.

Отдельную группу составляют компоненты синхронного просмотра данных – TDBLookupListBox и TDBLookupComboBox. Они обеспечивают показ значений поля из одной таблицы в соответствии со значениями поля из другой таблицы.

Следует отметить, что наиболее часто при программировании приложений БД используются компоненты TDBGrid, TDBEdit и TDBNavigator [6].

Далее рассмотрим особенности отдельных визуальных компонентов, предназначенных для работы с данными. Следует учесть, что при описании компонентов в пособии будут рассмотрены только основные, наиболее часто используемые свойства, методы и обработчики событий.

4.2. Визуальные компоненты для табличного представления данных

Для табличного представления данных в Delphi используются два компонента: TDBGrid и TDBCtrlGrid. Оба компонента выводят записи набора данных в табличном виде, при этом компонент TDBGrid выводит записи в виде стандартной таблицы, а компонент TDBCtrlGrid каждую ячейку такой таблицы отображает в виде формы.

4.2.1. Компонент TDBGrid

Как уже было отмечено выше, компонент TDBGrid – один из наиболее часто используемых визуальных компонентов при программировании приложений БД.

Компонент представляет собой сетку, двумерную таблицу, которая соответствует внутренней структуре набора данных, при этом строки представляют собой записи, а столбцы – поля набора данных [4, 6]

Основные особенности компонента

По своему виду сетка TDBGrid похожа на стандартную таблицу TStringGrid, но между ними есть значительные различия. Так, для TStringGrid можно устанавливать через соответствующие свойства число строк и столбцов. У TDBGrid числом строк управлять нельзя, так как отображаются все записи, имеющиеся в связанном наборе данных [4].

На рис. 4.2 приведена форма, на которой размещен компонент TDBGrid для отображения данных об абонентах в табличном виде. Предварительно создан и активизирован набор данных IBDataset2, который возвращает все поля таблицы Abonent, и связанный с ним источник данных DataSource2, затем для компонента TDBGrid заполнено свойство DataSource. Компонент TDBGrid (см. рис. 4.2) отображает все столбцы связанного набора данных в исходном виде. (Как можно с помощью средств компонента TDBGrid отобразить данные связанного набора данных в более удобном и понятном виде см. стр. 71.)

С помощью таблицы TDBGrid пользователь управляет набором данных, поля которого в ней отображаются.

Для навигации по записям и их просмотра используются полосы прокрутки и клавиши перемещения курсора.

На этапе выполнения программы для перехода в режим редактирования поля определенной записи достаточно установить на него курсор в компоненте TDBGrid и нажать любую алфавитно-цифровую клавишу.

Переход в режим вставки новой записи выполняется нажатием клавиши *<Insert>*, после чего можно заполнять поля. Вставка записи происходит в том месте, где находится указатель текущей записи.

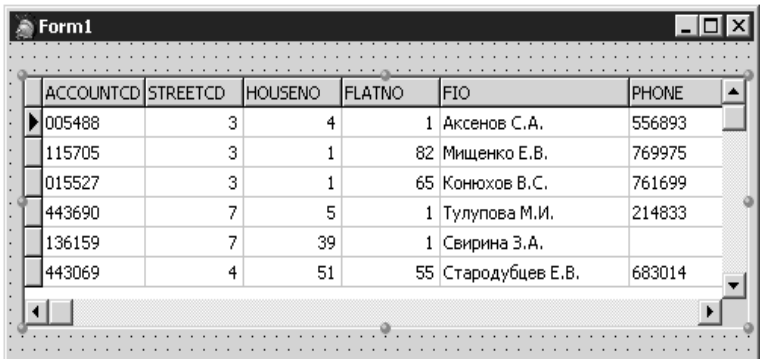


Рис. 4.2. *Компонент TDBGrid*

Изменения, сделанные при редактировании или добавлении записи, подтверждаются нажатием клавиши <*Enter*>, или переходом к другой записи, или отменяются нажатием клавиши <*Esc*>.

Для удаления записи следует нажать комбинацию клавиш <*Ctrl*>+<*Delete*>.

Большинство свойств компонента TDBGrid доступно как на этапе разработки приложения через инспектора объектов, так и на этапе выполнения программы. Ряд свойств доступен только на этапе выполнения программы.

Ниже перечислены основные свойства компонента.

1. ***Свойства, доступные на этапе разработки и на этапе выполнения:***

- *Columns* типа TDBGridColumns – одно из самых важных свойств таблицы. Содержит коллекцию объектов TColumn, описывающих колонки компонента (будет подробно рассмотрено далее);
- *DefaultDrawing* типа Boolean. Определяет способ визуализации данных в сетке. При значении True данные отображаются автоматически, а при значении False используется метод-обработчик OnDrawColumnCell;
- *Options* типа TDBGridOptions. С помощью данного свойства осуществляется настройка параметров компонента TDBGrid, от которых зависит его внешний вид и некоторые функции. Свойство представляет собой множество (TDBGridOptions = set of TDBGridOption) и принимает комбинации значений, каждое из которых определяет особенности визуализации и поведения компонента [4, 6]:
  - dgEditing – данные можно редактировать;
  - dgAlwaysShowEditor – данные в сетке всегда в режиме редактирования;
  - dgTitles – видны заголовки колонок;
  - dglndicator – в начале строки виден номер текущей колонки;
  - dgColumnResize – колонки можно перемещать и менять их ширину;

- dgColLines – видны линии между колонками;
- dgRowLines – видны линии между строками;
- dgTabs – для перемещения по строкам можно использовать клавиши <Tab> и <Shift>+<Tab>;
- dgRowSelect – можно выделять целые строки, при этом игнорируются установки dgEditing И dgAlwaysShowEditor;
- dgAlwaysShowSelection – выделение текущей ячейки сохраняется, даже если сетка не активна;
- dgConfirmDelete – при удалении строк появляется запрос о подтверждении операции;
- dgCancelOnExit – созданные пустые строки при уходе из сетки не сохраняются;
- dgMultiSelect – можно выделять несколько строк одновременно.

По умолчанию свойство Options содержит комбинацию значений [dgEditing, dgTitles, dglndicator, dgColumnResize, dgColLines, dgRowLines, dgTabs, dgConfirmDelete, dgCancelOnExit];

- *TitleFont* – шрифт заголовков колонок;
- *Color* и FixedColor типа TColor. Свойства задают цвета сетки и ее фиксированных ячеек соответственно.

2. ***Свойства, доступные на этапе выполнения программы.*** Обращение к данным свойствам выполняется внутри программного кода, результат доступен при выполнении программы. Основные свойства:

- *FieldCount* типа Integer. Возвращает число видимых колонок таблицы;
- *Fields [Index: Integer]* типа TField. Массив объектов полей набора данных, отображаемых в компоненте. Индекс определяет номер столбца в массиве столбцов и принимает значения в интервале 0..FieldCount-1;
- *SelectedField* типа TField. Содержит объект текущего поля;
- *SelectedRows* типа TBookmarkList. Набор закладок на записи набора данных, соответствующих выделенным строкам сетки;
- *Selectedlndex* типа Integer. Содержит номер текущей колонки в массиве Columns. Свойства SelectedField, SelectedRows и Selectedlndex совместно определяют текущую позицию в двумерной структуре данных;
- *EditorMode* типа Boolean. Показывает, можно ли редактировать текущую ячейку, или переводит TDBGrid в режим редактора данных ячейки;
- *Canvas* типа TCanvas. Канва компонента с ее свойствами и методами.

Для компонента TDBGrid доступны методы-обработчики событий, среди которых есть как стандартные методы, присущие всем элементам управления, так и специфические.

К специфическим ***методам*** относятся:

- *OnCellclick* и *OnTitleClick*. Событие OnCellclick генерируется при щелчке на ячейке с данными, а OnTitleClick – при щелчке на заголовке столбца. Оба события имеют тип TDBGridClickEvent, описываемый так:
  type TDBGridClickEvent = procedure (Column: TColumn) of object;
  где Column – столбец, на котором был произведен щелчок.
- *OnColEnter* и *OnColExit* типа TNotifyEvent. События инициируются при перемещении фокуса между столбцами сетки. OnColEnter возникает при получении столбцом фокуса, а OnColExit – при его потере.
- *OnColumnMoved*. Событие генерируется при перемещении столбцов сетки с помощью мыши (это возможно в случае, если свойство Options содержит значение dgColumnResize). Событие имеет тип TMovedEvent, описываемый как:
  type TMovedEvent = procedure (Sender: TObject; Fromlndex, Tolndex: Longint) of object;,
  где Fromlndex и Tolndex указывают индексы в массиве столбцов сетки, соответствующие предыдущему и новому положению перемещенного столбца соответственно.
- *OnEditButtonClick*. Событие генерируется при щелчке мышью на кнопке в ячейке.
- *OnDrawColumnCell*. Событие возникает при прорисовке любой ячейки и используется для программной реализации отображения сетки. Порядок вызова события OnDrawColumnCell зависит от значения свойства DefaultDrawing типа Boolean. Программная прорисовка сетки может понадобиться в случае, когда желательно выделить ячейку или столбец с помощью цвета или шрифта, а также вывести в ячейке, кроме текстовой, и графическую информацию и т. д. Событие имеет одноименный тип OnDrawColumnCell:
  type TDrawColumnCellEvent = procedure (Sender: TObject; const Rect: TRect; DataCol: Integer; Column: TColumn; State: TGridDrawState) of object;,
  где Rect – координаты ограничивающего ячейку прямоугольника; DataCol – номер прорисовываемого столбца в массиве столбцов сетки; Column – объект прорисовываемого столбца; State – задает состояние ячейки и принимает комбинации следующих значений:
  - gdSelected (ячейка находится в выбранном диапазоне);
  - gdFocused (ячейка имеет фокус ввода);
  - gdFixed (ячейка находится в фиксированном диапазоне).

Совместно с событием OnDrawColumnCell часто два метода компонента TDBGrid:

- *DefaultDrawColumnCell (const Rect: TRect; DataCol: Integer; Column: TColumn; State: TGridDrawState).* Метод перерисовывает текст в ячейке колонки с номером DataCol. Ячейка задается прямоугольником Rect на канве сетки. Параметр state определяет состояние ячейки после перерисовки. Параметр Column содержит экземпляр класса колонки, которой принадлежит ячейка.
- *DefaultDrawDataCell (const Rect: TRect; Field: TField; State: TGridDrawState).* Метод перерисовывает текст в ячейке колонки, определяемой параметром Field, содержащим связанный с колонкой объект поля. Ячейка задается прямоугольником Rect на канве сетки.

Рассмотрим пример использования метода OnDrawColumnCell для перерисовки изображения в ячейках TDBGrid. Допустим в таблице, приведенной выше на рис. 4.2, записи с пустым номером телефона необходимо выделять желтым цветом. Обработчик события OnDrawColumnCell в данном случае может иметь вид:

```
procedure TForm1.DBGrid1DrawColumnCell(Sender: TObject; const Rect: TRect;
  DataCol: Integer; Column: TColumn; State: TGridDrawState);
begin
if (IBDataSet2.FieldByName('Phone').Value = NULL) then
begin
  with  DBGrid1.Canvas do
  begin
    Brush.Color:=clYellow;
    FillRect(Rect);
    TextOut(Rect.Left+2,Rect.Top+2,Column.Field.Text);
  end;
end;
end;
```

В приведенном коде при прорисовке ячейки проверяется, не пустой ли номер телефона в текущей записи набора данных. Если это так, то для канвы DBGrid1 устанавливается желтый цвет заливки, а затем вызываются методы FillRect и TextOut свойства Canvas для заливки ячейки и прорисовки текста в определенных координатах.

Следует отметить, что приведенный выше код подходит только для тех случаев, если нет специального форматирования выводимого текста в полях таблицы. Использование метода TextOut нарушает внутреннее форматирование. Чтобы этого избежать, при перерисовке таблицы необходимо использовать метод DefaultDrawColumnCell компонента TDBGrid, который отрисовывает указанную ячейку со всеми настройками DBGrid, но с заданным цветом на канве.

Ниже приведен модифицированный код обработчика.

```
procedure TForm1.DBGrid1DrawColumnCell(Sender: TObject; const Rect: TRect;
  DataCol: Integer; Column: TColumn; State: TGridDrawState);
begin
if (IBDataSet2.FieldByName('Phone').Value = NULL) then
begin
  DBGrid1.Canvas.Brush.Color:=clYellow;
  DBGrid1.DefaultDrawColumnCell(Rect,DataCol,Column,State);
end;
end;
```

Работа со столбцами

В работе компонента TDBGrid важную роль играет класс TColumn, который инкапсулирует свойства колонки (столбца). Его основным назначением является правильное отображение данных из поля набора данных, связанного с этой колонкой [6].

Совокупность колонок компонента TDBGrid доступна через свойство *Columns* типа TDBGridColumns и представляет собой массив (коллекцию, индексированный список) объектов Column типа TColumn, описывающих отдельные столбцы таблицы [4, 6].

Основные свойства и методы класса TDBGridColumns доступны на этапе выполнения программы. Наиболее часто используется несколько свойств:

Count типа Integer. Возвращает общее число колонок.

Items [Index: Integer] типа Tcolumn. Доступ к колонкам осуществляется при помощи данного свойства, представляющего собой индексный список объектов колонок таблицы. Нумерация колонок начинается с нуля.

State типа TDBGridColumnsState. Определяет способ создания колонок сетки: при csDefault колонки создаются динамически с параметрами, соответствующими связанным полям; при csCustomized параметры колонок определены разработчиком и могут отличаться от параметров полей. Значение свойства устанавливается автоматически.

По умолчанию для каждого поля набора данных, связанного с компонентом DBGrid, автоматически создается отдельный столбец, и все столбцы в сетке доступны [4]. Такие столбцы являются динамическими.

Однако для компонента можно создать массив статических столбцов, включающий в себя произвольное количество полей связанного набора данных [6].

Вышеупомянутое свойство State автоматически устанавливается в значение csDefault, если столбцы динамические. При любом ручном изменении свойств устанавливается значение csCustomized.

Для создания статических столбцов используется специальный Редактор столбцов (рис. 4.3). Редактор вызывается из контекстного меню компонента DBGrid путем выбора пункта Columns Editor, либо открывается при двойном щелчке на компоненте TDBGrid, либо кнопкой свойства Columns в Инспекторе объектов.

В заголовке Редактора столбцов выводится составное имя массива столбцов, например, DBGridl.Columns, ниже располагается панель инструментов.

Рис. 4.3. *Редактор столбцов*

Большую часть Редактора столбцов занимает список статических столбцов, при этом столбцы перечисляются в порядке их создания (этот порядок может отличаться от исходного порядка полей в наборе данных) [4].

Первоначально список статических столбцов пуст, показывая тем самым, что все столбцы сетки являются динамическими. Редактор столбцов на этапе разработки приложения позволяет выполнить ряд операций по управлению набором статических столбцов [4]:

- создать статический столбец;
- удалить статический столбец;
- изменить порядок следования статических столбцов.

Кроме того, для любого выбранного в Редакторе статического столбца (объекта типа TColumn) через Инспектор объектов можно задать или изменить его свойства и определить обработчики его событий. Это допустимо потому, что соответствующие статическим столбцам объекты типа TColumn доступны уже на этапе разработки приложения [1].

Новый статический столбец добавляется при помощи кнопки *Add New* либо нажатием клавиши <*Insert*>, после этого к списку добавляется строка, соответствующая новому столбцу. В левой части строки содержится номер этого столбца в массиве столбцов, в правой – имя поля набора данных, с которым связан столбец.

Сразу после добавления к списку столбец не связан ни с одним полем и вместо имени поля указывается TColumn. При выполнении приложения подобный столбец окажется пустым. Чтобы связать столбец с каким-либо полем, необходимо установить значение его свойства FieldName.

Вновь создаваемые статические столбцы получают значения свойств по умолчанию, зависящие также от полей набора данных, с которыми эти столбцы связаны.

При помощи кнопки *Add All Fields* в сетку можно добавить все поля набора данных.

Для удаления из списка статических столбцов необходимо их выделить, после чего нажать кнопку  панели инструментов Редактора столбцов либо нажать клавишу <*Delete*>.

При изменении порядка столбцов сетке автоматически изменяется порядок связанных с ними полей набора данных, что необходимо учитывать при доступе к полям по номерам объектов типа TField,а не по именам объектов.

Если хотя бы один столбец сетки является статическим, то динамические столбцы уже не создаются ни для одного из оставшихся полей набора данных. Причем в наборе данных доступными являются статические столбцы, а остальные столбцы считаются отсутствующими.

Достоинством статических столбцов является то, что для них можно установить значения свойств, отличные от свойств соответствующего поля набора данных, в то время как функционирование динамических столбцов полностью зависит от свойств объекта поля (при изменении свойств объекта типа TField соответственно изменяются свойства объекта типа TColumn) [4]. Например, динамический столбец получает от поля такие характеристики, как имя и ширину, а для статического столбца можно установить свое имя, и оно не будет меняться даже в случае, если с этим столбцом связывается другое поле набора данных.

К достоинствам можно отнести и то, что объекты типа Tcolumn статических столбцов создаются на этапе разработки приложения, и через Инспектор объектов можно задать или изменить свойства таких столбцов.

Наиболее важные свойства класса TColumn (объекта столбца), доступные для чтения (просмотра) и изменения [4]:

Field типа TField – определяет объект поля набора данных, связанный со столбцом.

FieldName типа string – указывает имя поля набора данных, с которым связан столбец. При установке этого свойства с помощью Инспектора объектов значение можно выбирать в списке.

PopupMenu типа TPopupMenu – связывает с колонкой всплывающее меню.

PickList типа TStrings – представляет собой список для выбора заносимых в поле значений. Текущая ячейка совместно со списком PickList образуют своего рода компонент ComboBox или DBComboBox. Если для столбца сформирован список выбора, то при попытке редактирования ячейки этого столбца справа появляется стрелка, при нажатии которой список раскрывается и позволяет выбрать одно из значений. При этом можно ввести в ячейку любое допустимое значение.

Title типа TColumnTitle – представляет собой объект заголовка столбца. В свою очередь этот объект имеет такие свойства, как Caption, Alignment, Color и Font, определяющие соответственно название, выравнивание, цвет и шрифт заголовка.

DropDownRows – определяет число строк разворачивающегося списка ячейки.

Alignment типа TAlignment – управляет выравниванием значений в ячейках столбца и может принимать значения:

- taLeftJustify (выравнивание по левой границе);
- taCenter (выравнивание по центру);
- taRight Justify (выравнивание по правой границе).

Color типа TColor – цвет фона колонки. В свою очередь этот объект имеет такие свойства, как Caption, Alignment, Color и Font, определяющие соответственно название, выравнивание, цвет и шрифт заголовка .

Visible типа Boolean – задает видимость колонки.

Showing типа Boolean – возвращает значение True, если колонка видима, и False в противном случае.

После ручного изменения свойств столбца через Инспектор объектов восстановить их первоначальные значения можно нажатием кнопки на панели инструментов Редактора столбцов либо при помощи группы специальных методов (DefauitColor, DefaultFont и др.) [6].

Создадим, например, статические столбцы для компонента TDBGrid (см. рис. 4.2).

С помощью редактора столбцов добавим столбцы для отображения номера лицевого счета, ФИО и телефона абонента. Для всех создаваемых столбцов заполним свойство FieldName соответствующими полями набора данных, а в свойстве Title пропишем заголовок русскими буквами. Для столбцов с номером лицевого счета и телефоном зададим расположение данных в центре ячейки путем установки свойства Alignment в значение taCenter. После выполнения указанных действий компонент будет иметь вид, представленный на рис. 4.4.

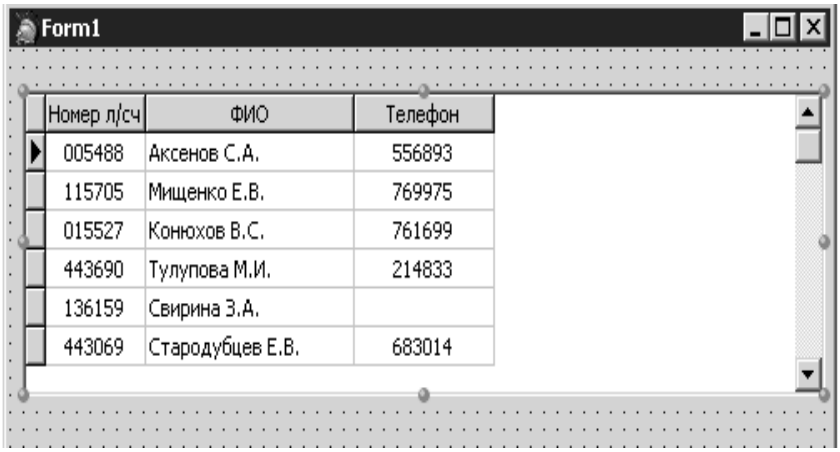


Рис. 4.4. *Компонент TDBGrid со статическими столбцами*

4.2.2. Компонент TDBCtrlGrid

Кроме компонента TDBGrid, для управления записями таблицы предназначен компонент TDBCtrlGrid – модифицированная сетка [4].

Компонент TDBCtrlGrid фактически сочетает в себе два способа отображения данных: это таблица, каждая ячейка которой отображается в виде формы [15].

Компонент DBCtrlGrid представляет собой набор панелей, каждая из которых служит платформой для размещения данных отдельной записи набора данных [4, 6].

На панели могут размещаться любые визуальные компоненты отображения данных, предназначенные для работы с отдельными полями. Нельзя использовать компоненты TDBGrid, TDBCtrlGrid, TDBRichEdit, TDBListBox, TDBRadioGroup, TDBLookupListBox. Не рекомендуется размещать на панели компоненты TDBMemo и TDBimage, так как это может привести к значительному снижению производительности [6].

При проектировании приложения визуальные компоненты для работы с полями необходимо разместить в верхней левой панели компонента TDBCtrlGrid. Когда визуальные компоненты помещаются на панель модифицированной сетки, у них автоматически устанавливается значение свойства DataSource, взятое из аналогичного свойства модифицированной сетки. Самостоятельное задание свойства DataSource для дочерних компонентов отображения данных не допускается. С каждым компонентом необходимо связать только нужное поле набора данных. Во время выполнения программы расположение и состав компонентов верхней панели будут реплицированы на все оставшиеся панели.

На рис. 4.5 приведена форма с размещенным на ней компонентом TDBCtrlGrid на этапе разработки.

Рис. 4.5. *Компонент TDBCtrlGrid на этапе разработки*

На верхней панели размещены: компонент DBText для отображения номера лицевого счета и два компонента DBEdit для вывода ФИО и номера телефона абонента.

На рис. 4.6 представлена та же форма во время выполнения программы. Как можно видеть, на каждой из панелей отображаются идентичным образом записи набора данных.

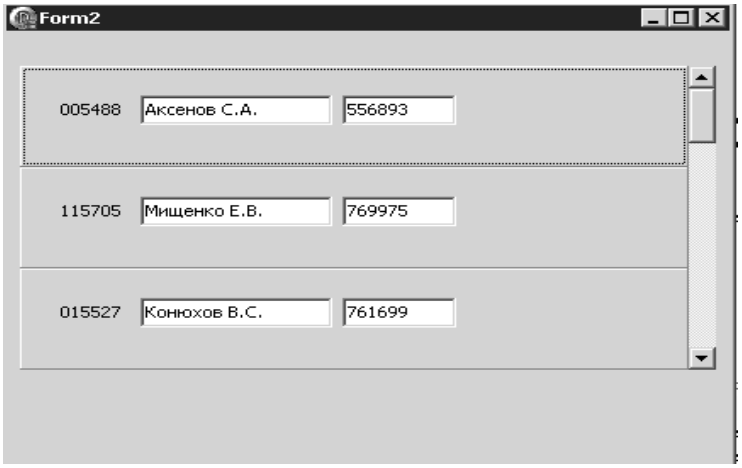


Рис. 4.6. *Компонент TDBCtrlGrid на этапе выполнения программы*

Приемы работы с TDBCtrlGrid (добавление, редактирование записей и т. д.) аналогичны приемам работы с TDBGrid.

Основные свойства, характерные для компонента TDBCtrlGrid:

Allowlnsert и AllowDelete типа Boolean. Разрешают/запрещают вставку и удаление записей соответственно в набор данных, записи которого отображаются на панелях компонента (по умолчанию True).

ColCount и RowCount типа Integer. Свойства определяют число одновременно видимых столбцов и строк панелей (по умолчанию три и один соответственно).

Orientation типа TDBCtrlGridOrientation. От значения свойства (goVertical или goHorizontal) зависит вертикальное (по умолчанию) или горизонтальное (с горизонтальной полосой прокрутки) размещение колонок панелей.

PanelBorder типа TDBCtrlGridBorder. Свойство определяет рамку панели и принимает значение gbNone (рамки нет) либо gbRaised (трехмерная приподнятая рамка) по умолчанию.

PanelHeight и PanelWidth типа Integer. Высота и ширина панели в пикселах. Все панели сетки имеют одинаковые размеры, и они взаимосвязаны с размерами самого компонента. При изменении значений свойств PanelHeight и Panelwidth размеры компонента изменяются таким образом, чтобы в нем

помещалось указанное в свойствах colcount и RowCount число панелей и наоборот.

SelectedColor типа TColor. Определяет цвет выделенной ячейки;

ShowFocus типа Boolean. Разрешает/запрещает обводить пунктирным прямоугольником ячейку с фокусом ввода.

На этапе выполнения программы дополнительно доступны такие свойства, как Canvas типа TCanvas (канва компонента с ее свойствами и методами), PanelCount типа Integer (число панелей, одновременно видимых в модифицированной сетке, доступно для чтения).

При использовании компонента TDBCtrlGrid программисту предоставляются дополнительные возможности управления набором данных, с помощью метода DoKey. Процедура DoKey (Key: TDBCtrlGridKey) выполняет различные операции, которые задает параметр Key, принимающий значения [4]:

- gkNull (пустое действие);
- gkEditMode (перевод компонента в режим редактирования);
- gkPriorTab (передача фокуса ввода следующему элементу формы);
- gkNextTab (передача фокуса ввода предыдущему элементу формы);
- gkLef t (переход на один столбец влево);
- gkRight (переход на один столбец вправо);
- gkUp (переход на одну панель вверх);
- gkDown (переход на одну панель вниз);
- gkScrollUp (переход на одну строку вверх);
- gkScrollDown (переход на одну строку вниз);
- gkPageUp (переход вперед на число записей, равное произведению числа строк и столбцов панелей сетки Colcount * RowCount, т.е. на экран вверх);
- gkPageDown (переход на экран вниз);
- gkHome (переход на первую запись);
- gkEnd (переход на последнюю запись);
- gkinsert (перевод компонента в режим редактирования);
- gkAppend (перевод компонента в режим добавления записи);
- gkDelete (перевод компонента в режим удаления записи);
- gkCancel (восстановление режима просмотра).

При необходимости разработчик может выполнить программную прорисовку панелей, используя для этого событие *OnPaintPanel* типа TPaintPanelEvent, возникающее непосредственно перед отображением панелей [4]. Тип события описан так:

type TPaintPanelEvent = procedure (DBCtrlGrid: TDBCtrlGrid; Index: Integer) of object;,

где Index – номер панели, отображение элементов которой выполняется в настоящий момент. Кроме прорисовки, в обработчике события OnPaintPanel можно выполнить другие действия, например, обработку связанных с панелью данных.

4.3. НАВИГАЦИЯ ПО НАБОРУ ДАННЫХ

При отображении на форме наборов данных всегда возникает необходимость предоставления пользователю возможности удобного перемещения (навигации) по записям набора. Навигация может осуществляться несколькими путями [6]. В компонентах TDBGrid и TDBCtrlGrid, которые отображают сразу несколько записей набора данных, можно использовать клавиши вертикального перемещения курсора или вертикальную полосу прокрутки. В случае если на форме находятся только компоненты, отображающие одно поле только текущей записи набора данных (TDBEdit, TDBComboBox и т. д.), очевидна необходимость дополнительного элемента управления, отвечающего за перемещение по записям.

Для управления набором данных предназначен специальный компонент TDBNavigator – навигатор, который обеспечивает соответствующий интерфейс пользователя. TDBNavigator (рис. 4.7) представляет собой совокупность управляющих кнопок, обеспечивающих выполнение различных операций с набором данных путем автоматического вызова соответствующих методов [4]. Всего компонент содержит 10 кнопок, разработчик может оставить в наборе любое количество кнопок в любом сочетании.

Рис. 4.7. *Компонент TDBNavigator*

Компонент TDBNavigator при помощи свойства *DataSource* связывается с компонентом TDataSource и через него с набором данных [6]. Такая схема позволяет обеспечить изменение текущих значений полей сразу во всех связанных с TDataSource компонентах отображения данных. Таким образом, TDBNavigator только дает команду на выполнение перемещения по набору данных или другой управляющей операции, а всю реальную работу выполняют компонент набора данных и компонент TDataSource. Компоненты отображения данных только принимают новые данные от набора данных.

Следует отметить, что состав кнопок компонента зависит от набора данных, к которому подключен компонент. Например, если TDBNavigator подключен к набору данных TIBQuery, то кнопки вставки, удаления, редактирования записей будут недоступны при любом значении свойств компонента. Доступны будут только кнопки для навигации по записям.

Компонент TDBNavigator обладает следующими свойствами:

VisibleButtons типа TButtonSet. Свойство фактически представляет собой список видимых кнопок. Тип TButtonSet представляет собой множество

(TButtonSet = set of TNavigateBtn), каждый элемент которого как бы представляет одну кнопку и управляет ее видимостью:

- nbFirst – перемещение на первую запись набора данных;
- nbPrior – перемещение на предыдущую запись набора данных;
- nbNext – перемещение на следующую запись набора данных;
- nbLast – перемещение на последнюю запись набора данных;
- nblnsert – вставка новой записи в текущей позиции набора данных;
- nbDelete – удаление текущей записи, курсор перемещается на следующую запись;
- nbEdit – набор данных переводится в режим редактирования;
- nbPost – в базу данных переносятся все изменения в текущей записи;
- nbcancel – все изменения в текущей записи отменяются;
- nbRefresh – восстанавливаются первоначальные значения текущей записи, сделанные после последнего переноса изменений в БД.

По умолчанию в навигаторе видимы все кнопки.

ConfirmDelete типа Boolean. Включает или отключает подтверждение удаления записи. При установке значения свойства всегда следует помнить, что операция удаления записи является самой критичной к возможной потере данных вследствие ошибки. Поэтому в основном не рекомендуется отключать механизм контроля удаления.

Flat типа Boolean. Свойство управляет внешним видом кнопок. По умолчанию оно имеет значение False, и кнопки отображаются в объемном виде. При установке свойства Flat в значение True кнопки приобретают плоский вид, соответствующий современному стилю [4].

Hints типа Tstring. Свойство устанавливает подсказки для отдельных кнопок. При этом следует учесть, что для отображения подсказок нужно присвоить значение True свойству showHint, по умолчанию имеющему значение False.

Среди методов компонента TDBNavigator можно выделить метод *BtnClick(index: TNavigateBtn),* который служит для имитации нажатия кнопки, заданной параметром index. Тип TNavigateBtn этого параметра идентичен типу TButtonSet, возможные значения соответствующего параметра уже были перечислены. Например, нажатие кнопки nbNext, вызывающей переход к следующей записи набора данных, можно эмулировать программным кодом:

```
DBNavigator1.BtnClick(nbNext);
```

В случае необходимости выполнения дополнительных действий при щелчке на любой кнопке можно воспользоваться обработчиками событий BeforeAction и OnClick, в которых параметр Button определяет нажатую кнопку.

4.4. Компоненты для представления отдельных полей

Большинство компонентов отображения данных предназначено для представления данных из отдельных полей [6]. Для связи с полями таблиц БД все такие компоненты имеют свойства *DataSource* для указания источника данных и DataField, которое указывает на требуемое поле набора данных.

Среди визуальных компонентов для вывода отдельных полей можно выделить несколько групп.

1. Универсальные компоненты для отображения и редактирования значений полей. Сюда относятся компоненты TDBText и TDBEdit.
2. Компоненты для отображения и редактирования полей типа Memo. Сюда относятся компоненты TDBMemo и TDBRichEdit.
3. Компоненты для отображения и редактирования логического поля. К этой группе относится единственный компонент DBCheckBox.
4. Компоненты для выбора значения поля из фиксированного числа значений. В этой группе можно выделить подгруппы:
   - компоненты для выбора значения из фиксированного набора (DBRadioGroup);
   - компоненты для выбора значения из фиксированного списка (DBListBox, DBComboBox);
   - компоненты для выбора из списка, сформированного по значениям поля набора данных (DBLookupComboBox, DBLookupListBox). Компоненты синхронного просмотра данных
5. Компонент для вывода изображений DBImage.
6. Компонент для вывода диаграмм и графиков DBChart.

Необходимое количество компонентов для работы с полями размещается на форме путем перетаскивания с закладки Data Controls. Для большинства стандартных полей при разработке приложений используются компоненты TDBText, TDBEdit, TDBComboBox, TDBListBox [6].

4.4.1. Универсальные компоненты TDBText и TDBEdit

Компоненты TDBText и TDBEdit являются наиболее универсальными для отображения полей различных типов среди всех визуальных компонентов для работы с отдельными полями.

TDBText – один из самых простых визуальных компонентов. Этот компонент представляет собой статический текст, который отображает текущее значение некоторого поля связанного набора данных. При этом данные можно просматривать в режиме "только для чтения".

Компонент является аналогом компонента TLabel палитры Standart.

При использовании компонента следует обратить внимание на возможную длину отображаемых данных [6]. Для предотвращения обрезания текста можно использовать свойства *AutoSiz*e и *WordWrap*.

Компонент TDBEdit – один из наиболее часто используемых компонентов при разработке приложений БД.

Компонент TDBEdit представляет собой стандартный однострочный текстовый редактор, в котором отображаются, а также доступны для редактирования, данные из поля связанного набора данных. Компонент можно создать на форме простым перетаскиванием из списка объектов-полей связанного набора данных, что избавит от необходимости заполнения свойств для связи с полем набора данных.

Компонент является аналогом компонента TEdit палитры Standart.

Значение поля, отображаемое в компоненте, может быть изменено или добавлено в набор данных. Как только компонент потеряет фокус, содержащееся в нем значение поля будет отображено в наборе данных.

Компонент TDBEdit может осуществлять проверку редактируемых данных по заданной для поля маске [6]. Маска задается свойством *EditMask*, однако непосредственно для редактора задать маску нельзя. Маску можно задать в связанном с редактором поле TField, которое имеет свойство EditMask, используемое затем при проверке данных в редакторе.

Для проверки редактируемого текста на соответствие маске используется метод *ValidateEdit* компонента TDBEdit. Метод осуществляет проверку после каждого введенного или измененного символа, в случае ошибки генерирует исключение, и курсор устанавливается на первый ошибочный символ [6].

В компоненте TDBEdit можно использовать буфер обмена. Буфер доступен средствами операционной системы для пользователей или программно при помощи методов *CopyToClipboard, CutToClipboard, PasteFromCiipboard.*

Компоненты TDBText и TDBEdit были показан на форме рис. 4.5 при рассмотрении компонента TDBCtrlGrid.

4.4.2. Компоненты для полей Memo: TDBMemo и TDBRichEdit

Компонент TDBMemo представляет собой поле редактирования, к которому подключается поле с типом данных Memo или BLOB [6]. Компонент предоставляет возможность одновременного просмотра и редактирования нескольких строк переменной длины.

В компоненте TDBMemo, как и в компоненте TDBEdit, можно использовать буфер обмена.

Компонент имеет свойство *AutoDisplay*, которое полезно для ускорения навигации по набору данных при отображении полей типа BLOB. При значении True любое новое значение поля автоматически отображается в компоненте. При значении False новое значение появляется только после двойного

щелчка на компоненте или после нажатия клавиши <*Enter*> при активном компоненте.

В случае, когда AutoDisplay имеет значение False, для загрузки данных программным путем можно использовать метод LoadMemo.

Поведением компонента при работе со слишком длинными строками можно управлять при помощи свойства *WordWrap* [6]. При значении True слишком длинная строка сдвигается влево при перемещении текстового курсора за правую границу компонента. При значении False остаток длинной строки переносится на новую строку, при этом реально новая строка в данных не создается.

Компонент TDBRichEdit предоставляет возможности полноценного текстового редактора для просмотра и изменения текстовых данных, хранящихся в связанном поле набора данных.

Внешне компонент ничем не отличается от поля редактирования TDBMemo. Однако в отличие от компонента TDBMemo, который позволяет работать только с однородным (неформатированным) текстом, компонент TDBRichEdit предоставляет возможность работы с форматированным текстом, умеет интерпретировать специальные символы разметки текста в формате RTF (Rich Text Format – расширенный текстовый формат) [15].

4.4.3. Компонент для работы с логическими полями: TDBCheckBox

Логическое поле (поле логического типа) может содержать одно из двух значений: True (истина) или False (ложь).

Для отображения и изменения значения логического поля можно использовать редактор DBEdit. Однако удобнее выполнять эти действия с помощью флажка (независимого переключателя) DBCheckBox, который позволяет "включить" или "выключить" значение логического поля [4].

Флажок DBCheckBox является аналогом компонента CheckBox палитры компонентов Standart.

Флажок DBCheckBox можно применять также для отображения и редактирования строковых полей, но только в том случае, если строковое поле может иметь только два значения.

За включение/выключение флажка DBCheckBox отвечают свойства *ValueChecked* и *ValueUnChecked*. По умолчанию они имеют значения True и False, но допускают установку любых строковых значений.

Свойство ValueChecked типа string содержит строковые значения, которые устанавливают связанный с этим полем флажок во включенное состояние. Отдельные значения разделяются точкой с запятой. В качестве значений допускаются любые алфавитно-цифровые символы, в том числе русские буквы. Регистр алфавитных символов не различается.

Свойство ValueChecked может быть задано, например, таким образом:

```
DBCheckBox1.ValueChecked := 'True;T;Yes;Y;Да;Д';
```

Включение флажка происходит, если значение поля набора данных совпадает со значением свойства valuechecked (с любым из списка). Если флажок включает пользователь, то значение поля данных приравнивается к единственному или первому в списке значению свойства ValueChecked.

Свойство ValueUnChecked типа string содержит строковые значения, которые устанавливают связанный с этим полем флажок в выключенное состояние. Значения задаются таким же образом, как и для свойства ValueChecked.

Принцип выключения флажка в зависимости от свойства ValueUnchecked аналогичен принципу включения в зависимости от свойства ValueChecked.

Если поле не содержит ни одного из значений, указанных в свойствах ValueChecked и ValueUnChecked, то флажок устанавливается в неопределенное состояние.

На рис. 4.8 приведен пример формы с компонентом DBCheckBox. Компонент связан с полем Executed из таблицы Request и отображает факт отметки о выполнении ремонтной заявки. Свойство ValueChecked имеет значение 1, свойство ValueUnChecked – 0.

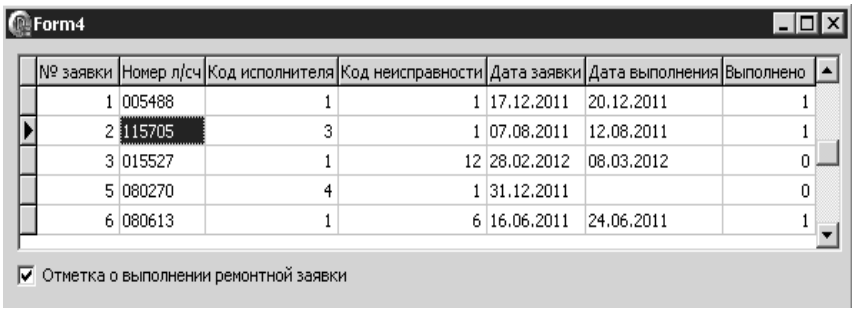


Рис. 4.8. *Компонент DBCheckBox*

Следует отметить, что, несмотря на наличие описанных свойств, возможности флажка DBCheckBox по редактированию строковых полей намного меньше, чем у элемента DBEdit [4].

4.4.4. Компоненты для выбора значения из списка

В ряде случаев при разработке приложений БД возникает необходимость ввода или отображения в поле одного из значений, входящих в состав фиксированного набора [4]. С этой целью в приложениях Delphi используются компонент DBRadioGroup (для выбора значения поля из фиксированного на-

бора) и компоненты TDBListBox и TDBComboBox (для выбора значения поля из списка значений).

Компонент TDBRadioGroup

Компонент TDBRadioGroup представляет собой группу зависимых переключателей, состояние которых зависит от значений поля связанного набора данных. В поле можно передавать фиксированные значения, связанные с отдельными переключателями в группе.

Если текущее значение связанного поля соответствует значению какого-либо переключателя, то он включается. Если пользователь включает другой переключатель, то связанное с переключателем значение заносится в поле.

Переключатель обладает рядом свойств:

Items типа TStrings. С помощью данного свойства выполняется управление числом и названиями переключателей и можно получить доступ к отдельным переключателям в группе [4]. Свойство содержит список строк, отображаемых как заголовки переключателей. Отсчет строк в массиве начинается с нуля: items [0], items [1] и т. д. Управление списком заголовков выполняется при разработке приложения с помощью Редактора строк или программно посредством методов класса Tstrings (Add и Delete).

Values типа TStrings. Свойство содержит список значений поля, на которые должны реагировать переключатели группы. Каждому элементу свойства values соответствует один переключатель (порядок следования сохраняется).

Управление списком осуществляется аналогично управлению списком items. Если возможные значения свойства Values не заданы, то они выбираются из значений свойства items, т. е. в этом случае значение, соответствующее переключателю, совпадает с названием этого переключателя.

Itemindex типа integer (доступно на этапе выполнения). Свойство предоставляет доступ к отдельному переключателю, содержит позицию (номер) переключателя, выбранного в группе в текущий момент. Это свойство используется для выбора отдельного переключателя или для определения, какой из переключателей является выбранным. Если свойство itemindex имеет значение -1, то не выбран ни один из переключателей.

Следует отметить, что при программном выборе переключателя необходимо переводить набор данных в режим редактирования, а после установки нового значения поля закреплять сделанные изменения, например, вызовом метода Post [4].

Группа переключателей работает таким образом. Переключатель включается при переходе к очередной записи, если значение связанного с ним поля содержит одно из значений, присутствующих в списке Values. Если же поле содержит значение, отсутствующее в списке возможных, то ни один переключатель не выбирается. Изменение значения поля происходит при выборе другого переключателя группы.

Для поля, с которым связана группа переключателей, пользователь может выбрать значение только из списка. Попытки ввести в поле произвольное значение, например с помощью компонента DBGrid, блокируются.

На рис. 4.9 приведена форма с группой переключателей Ресурс.

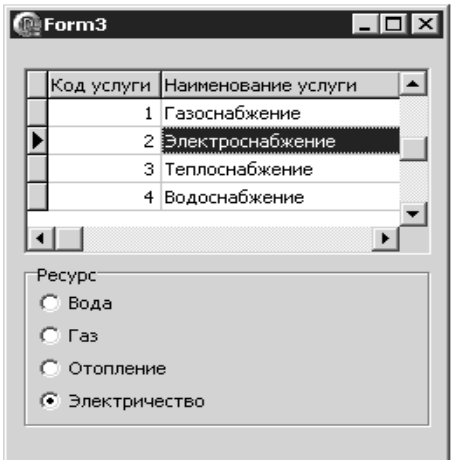


Рис. 4.9. *Компонент TDBRadioGroup. Группа переключателей Ресурс*

Отметим, что пример является лишь демонстрацией работы переключателя и в данном случае не несет никакого практического значения. Переключатель связан с полем ServiceNM набора данных. При этом списки Items и Values заполнены не совпадающими заголовками. При перемещении, например, на вторую запись в таблице ("Электроснабжение") автоматически выбирается последний переключатель, имеющий название "Электричество" (при этом значение – "Электроснабжение"). Если выбрать другой переключатель, например второй, то в поле ServiceNM второй записи автоматически занесется новое значение – "Газ". Новое значение поля будет зафиксировано в наборе данных и отображено визуальными компонентами (в данном случае – DBGrid) после потери фокуса группой переключателей, например, в связи с переходом к другой записи.

Далее рассмотрим списки, которые представляют более широкие возможности выбора для поля одного из фиксированных значений.

Компоненты TDBListBox и TDBComboBox

Компоненты TDBListBox и TDBComboBox представляют собой компоненты-списки (простой и комбинированный списки). По своей функциональности списки напоминают группу переключателей DBRadioGroup, однако предоставляют большие возможности [4].

Компоненты DBListBox и DBComboBox функционально практически ничем не отличаются – и тот, и другой отображают текущее значение связанного с ним поля набора данных и позволяют изменить его на любое фиксированное строковое значение из списка. При выборе нужной строки простого списка DBListBox содержащееся в ней значение автоматически заносится в поле, с которым связан этот список. В дополнение к этому комбинированный список DBComboBox позволяет вводить в поле произвольное значение.

Компонент DBComboBox имеет свойство *Style*, которое определяет для него пять различных стилей.

Специальных методов компоненты не содержат [6].

На рис. 4.10 приведен пример рис. 4.9, но с использованием вместо компонента DBRadioGroup компонента-списка TDBComboBox. Компонент отображает значение текущего связанного поля и позволяет выбрать для поля новое значение, как из списка, так и произвольное.

Рис. 4.10. *Компонент TDBComboBox*

Компоненты синхронного просмотра данных DBLookupListBox и DBLookupComboBox

При разработке приложений для работы с базами данных часто возникает необходимость в связывании двух наборов данных по ключевому полю [6]. Показ значений поля из одной таблицы в соответствии со значениями поля из другой таблицы обеспечивают компоненты синхронного просмотра данных TDBLookupListBox и TDBLookupComboBox.

Механизм синхронного просмотра. При разработке приложений для работы с БД часто возникает необходимость в связывании двух наборов дан-

ных по ключевому полю [6]. Механизм связывания полей из различных наборов данных по ключевому полю называется *синхронным просмотром*.

Таблица, в которой расположено поле, значения которого замещаются на синхронные, – это *исходная таблица*. Таблица, содержащая ключевое поле и поле данных для синхронного просмотра – это *таблица синхронного просмотра.*

В Delphi механизм синхронного просмотра реализован на уровне отдельных полей и компонентов.

Специальное поле синхронного просмотра можно динамически создать в наборе данных, и оно будет автоматически замещать в дочерней таблице одно значение другим в зависимости от значения ключевого поля. Создание подстановочного поля (Lookup-поля) для дочерней таблицы было кратко рассмотрено на стр. 44 и будет подробно рассмотрено при разработке демонстрационного приложения в п. 5.3. Следует отметить, что механизм создания подстановочного поля в дочерней таблице на практике используется наиболее часто.

Однако помимо простого синхронного просмотра данных может возникнуть задача редактирования данных в аналогичной ситуации. Для решения подобных задач предназначены специальные компоненты синхронного просмотра данных TDBLookupListBox и TDBLookupComboBox. Использование таких компонентов делает пользовательский интерфейс значительно более удобным и наглядным.

Для демонстрации механизма синхронного просмотра рассмотрим следующий пример на учебной БД. Например, в таблице Request (содержит данные о ремонтных заявках) имеется поле AccountCD (номер лицевого счета абонента, от которого поступила заявка) и поле FailureCD (код неисправности). Под этим же номером лицевого счета в таблице Abonent хранится информация об абоненте (ФИО, адрес, телефон), а под кодом неисправности – ее наименование в таблице Disrepair. При разработке пользовательского интерфейса приложения БД необходимо, чтобы при просмотре списка ремонтных заявок в форме приложения отображались не лицевые счета абонентов и номера неисправностей, а более понятная информация.

В наборе данных ремонтных заявок вместо поля номера лицевого счета должно появиться поле ФИО абонента из таблицы Abonent, а вместо поля с кодом неисправности – поле с ее наименованием из таблицы Disrepair.

В рассматриваемом примере ключевыми являются поля AccountCD из таблицы Abonent и FailureCD из таблицы Disrepair, а выбор конкретного ФИО или наименования производится по совпадению значений ключевого поля и заменяемого поля из исходного набора данных – Request. Причем необходимо, чтобы в таблицах Abonent и Disrepair поля AccountCD и FailureCD соответственно были уникальными (это условие удовлетворяется, так как данные поля для таблиц являются первичными ключами). Исходная

таблица в данном случае – таблица Request, а таблицы синхронного просмотра – таблица Abonent и таблица Disrepair.

Далее рассмотрим подробнее компоненты и их свойства для реализации механизма синхронного просмотра.

Особенности компонентов синхронного просмотра. Компоненты DBLookupListBox и DBLookupComboBox во многом похожи на простой список DBListBox и комбинированный список DBComboBox соответственно и отличаются от них только способом формирования списка возможных значений.

Основное назначение компонентов – автоматически устанавливать соответствие между полями двух наборов данных по одинаковому значению заданного поля исходной таблицы и ключевого поля таблицы синхронного просмотра. В списке синхронного просмотра отображаются возможные значения для редактирования поля основной таблицы.

Как и в любых других компонентах отображения данных отдельных полей, в компонентах синхронного просмотра, прежде всего, присутствуют средства связывания с требуемым полем некоторого набора данных [6]. Это свойства Datasource и DataField. Для синхронного просмотра следует выбирать такое поле, значения которого не дают пользователю полной информации об объекте и совпадают с ключевым полем в таблице синхронного просмотра. Название этого поля в принципе может не совпадать с названием ключевого поля (хотя при проектировании БД такая ситация нежелательна), но типы данных должны быть одинаковыми.

Далее необходимо задать таблицу синхронного просмотра, ключевое поле и поле синхронного просмотра.

Свойства, отвечающие за включение механизма синхронного просмотра:

ListSource типа TDataSource. Свойство указывает на компонент TDataSource, связанный с таблицей синхронного просмотра.

KeyField типа string. Свойство задает ключевое поле таблицы синхронного просмотра.

KeyValue типа Variant. Во время работы компонента в свойстве KeyValue содержится текущее значение, которое связывает между собой два набора данных.

ListField типа string. Свойство определяет поле синхронного просмотра. Здесь можно задавать сразу несколько полей, которые будут отображаться в компоненте синхронного просмотра. Названия полей разделяются точкой с запятой. Если свойство не определено, то в компоненте будут отображаться значения ключевого поля.

ListFieldindex типа Integer. Свойство используется, когда свойство ListField содержит список полей, и служит для выбора основного поля из списка. Дело в том, что компоненты синхронного просмотра поддерживают механизм наращиваемого поиска, который позволяет быстро находить нужное значение в больших списках. Свойство ListFieldindex определяет,

какое поле используется при наращиваемом поиске. В компоненте TDBiookupComboBox свойство ListFieldindex также определяет, какое поле будет передано в строку редактирования.

NullValueKey типа TShortCut. Определяет комбинацию клавиш, нажатие которых задает нулевое значение поля.

Рассмотрим, как на практике реализовать механизм синхронного просмотра в описанном примере о ремонтных заявках.

На форму поместим компонент TDBGrid и свяжем его с набором данных таблицы Request. Как уже было отмечено, необходимо реализовать механизм синхронного просмотра для полей AccountCD и FailureCD. Для этого на форму поместим компоненты TDBLookupComboBox и TDBLookupListBox.

При перемещении по записям набора данных в первом компоненте будет отображаться ФИО абонента, подавшего ремонтную заявку, а во втором – наименование неисправности. Описанная форма представлена на рис. 4.11.

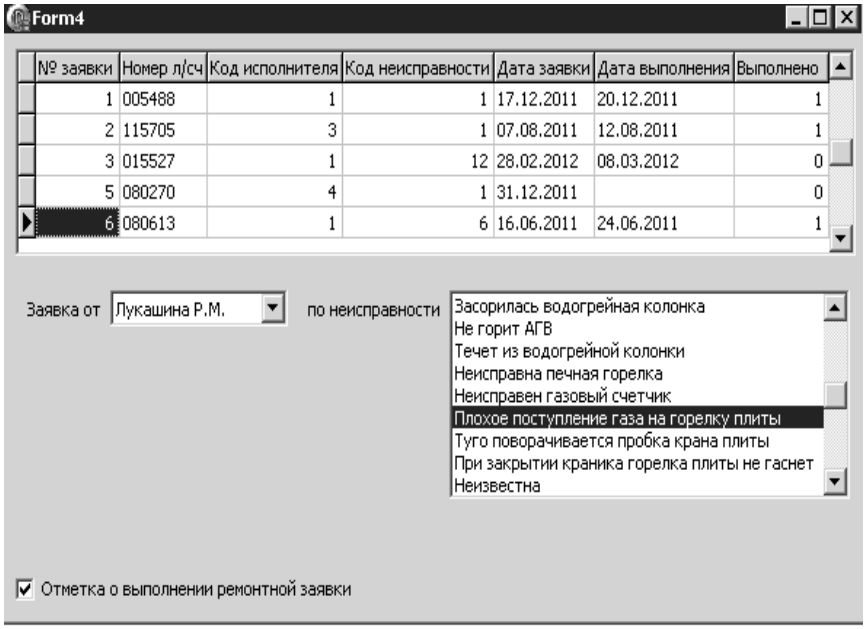


Рис. 4.11. *Компоненты синхронного просмотра данных*

Ключевые свойства первого компонента TDBLookupComboBox настроены таким образом:

- свойство Listsource указывает на компонент TDataSource, который связан с набором данных синхронного просмотра – таблицей Abonent;

- свойство ListField указывает на поле FIO, все значения которого доступны в списке компонента;
- свойство KeyField указывает на поле AccountCD, которое имеется в двух таблицах и по которому осуществляется связь.

Ключевые свойства второго компонента TDBLookupListBox настроены аналогично для связи с таблицей Disrepair.

4.4.5. Компонент TDBImage

Компонент TDBImage предназначен для просмотра изображений, хранящихся в базах данных в графическом формате [6]. Если компонент DBImage связать с полем, не содержащим изображение, например, с числовым, то в области компонента выводится имя этого поля [4].

Компонент DBimage поддерживает работу с буфером обмена Windows, тем самым позволяя не только просматривать изображение, но также изменить (заменить) его, вставив нужный рисунок из буфера обмена. Для выполнения этих действий используются обычные для Windows-программ комбинации клавиш: копирование в буфер – <*Ctrl*>+<*Insert*>, удаление в буфер – <*Shift*>+<*Delete*>, вставка из буфера – <*Shift*>+<*Insert*>.

Указанные действия также могут быть выполнены программно. Метод CopyToClipboard копирует изображение в буфер обмена, метод CutToClipboard вырезает (перемещает) изображение в буфер обмена, а метод PasteFromClipboard вставляет изображение из буфера обмена.

При использовании любого способа вставки из буфера новое изображение автоматически заменяет предыдущее содержимое компонента DBimage.

Компонент обладает следующими свойствами:

Picture типа TPicture отвечает за визуализацию изображения. С помощью методов свойства Picture можно полностью заменить существующее изображение или сохранить новое в новой записи набора данных.

AutoDisplay позволяет управлять процессом загрузки новых изображений из набора данных в компонент. При значении True любое новое значение поля автоматически отображается в компоненте. При значении False новое значение появляется только после двойного щелчка на компоненте или после нажатия клавиши <Enter> при активном компоненте. Можно также вывести содержимое поля программно с помощью метода *LoadPicture* [4].

QuickDraw задает используемую изображением палитру и применяется для ускорения просмотра изображений. При значении True применяется стандартная системная палитра. В результате уменьшается время загрузки изображения, но может ухудшиться и качество изображения, в некоторых случаях до полного искажения. При значении False используется собственная палитра изображения, и процесс загрузки замедляется.

4.5. Графическое представление данных

Для представления данных из некоторого набора данных в виде графиков различных видов предназначен компонент TDBChart [6]. Компонент позволяет выводить графики (диаграммы) различных типов, в том числе объемные. Этот компонент является достаточно сложным и имеет большое количество разнообразных свойств, многие из которых являются объектами и также имеют свои свойства [4]. Графики строятся на основе всех имеющихся в наборе данных значений полей, и при этом TDBChart может одновременно показывать графики для нескольких полей данных.

Функционально компонент не отличается от компонента TChart, однако является более универсальным за счет того, что в качестве источника данных можно использовать набор данных.

Настройка параметров компонента осуществляется специальным редактором диаграмм, который можно открыть двойным щелчком на перенесенном на форму компоненте.

Рассмотрим процесс подключения к компоненту набора данных и построение графиков. Построение графика будем рассматривать на примере учебной БД. Допустим, необходимо построить круговую диаграмму, показывающую суммы, начисленные по каждой из услуг (на основе данных таблицы NachislSumma). Предварительно создаем набор данных IBDataset3, который выбирает данные с помощью запроса:

```
SELECT
  SERVICECD,
  SUM(NACHISLSUM)
FROM
  NACHISLSUMMA
GROUP BY
  SERVICECD;
```

Рассмотрим процесс построения диаграммы по данным запроса.

На страницах редактора диаграмм отображается информация о различных свойствах-объектах.

Основой любого графика в компоненте TDBChart является так называемая *серия*. Серии представлены свойством *Series [index:Longint]* типа TchartSeries – это массив диаграмм, выводимых в области компонента.

Для того чтобы построить график значений некоторого поля набора данных, необходимо выполнить определенные действия, большинство из которых выполняется в специализированном редакторе диаграмм.

1. Создать новую серию и определить ее тип.
2. Задать для серии набор данных.
3. Связать с осями координат нужные поля набора данных и, в зависимости от типа серии, задать дополнительные параметры.
4. Открыть набор данных.

Две главные страницы редактора диаграмм – Chart и Series.

Страница Chart содержит многостраничный блокнот и предназначена для настройки параметров самого графика. Страница Series также содержит многостраничный блокнот и используется для настройки серий значений данных.

1. Для создания новой серии необходимо в редакторе перейти на главную страницу Chart, а на ней открыть страницу Series. На этой странице нужно щелкнуть на кнопке Add, а затем в появившемся диалоге выбрать тип серии (рис. 4.12).

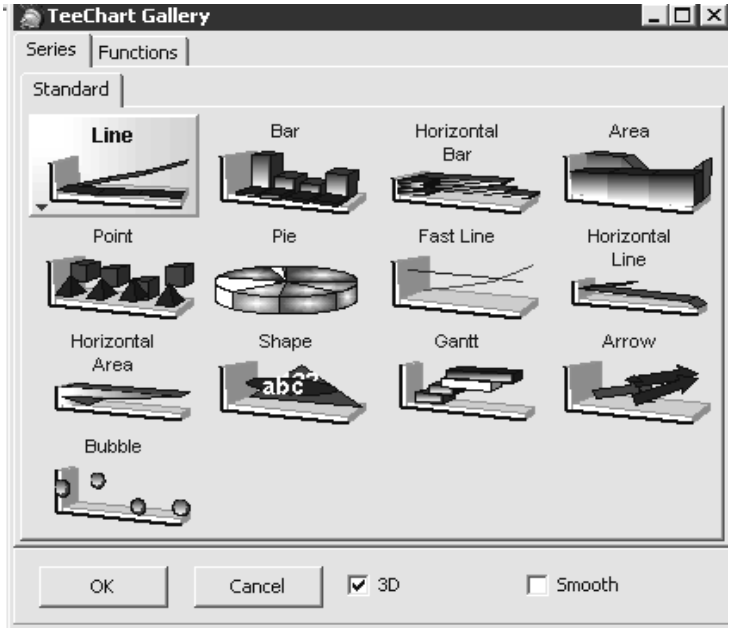


Рис. 4.12. *Выбор типа диаграммы*

После этого в списке на странице Series появляется строка новой серии (рис. 4.13). Здесь можно переопределить тип, цвет и видимость серии, щелкнув на соответствующей зоне строки.

Для выбранной диаграммы можно:

- изменить имя по умолчанию – нопка Title;
- изменить тип диаграммы – кнопка Change;
- скопировать диаграмму – кнопка Clone;
- удалить диаграмму – кнопка Delete.

Все остальные страницы блокнота на главной странице Chart предназначены для настройки параметров графика.

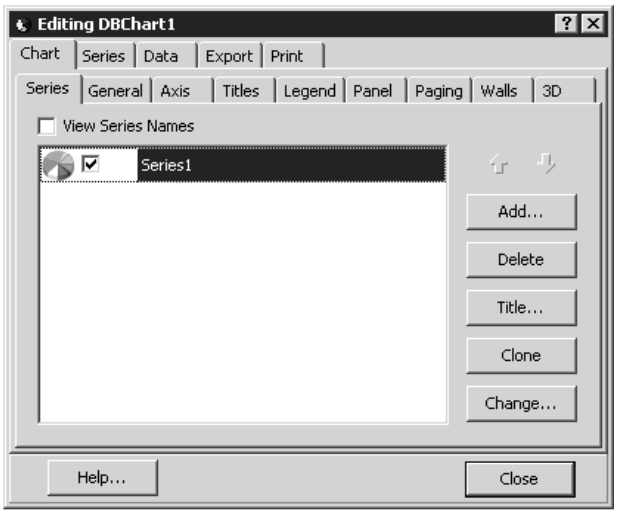


Рис. 4.13. *Редактор диаграмм*

2. После создания серии необходимо задать набор данных (рис. 4.14). Для этого необходимо перейти на главную страницу Series и на ней из списка названий серий выбрать необходимую. После этого на странице Data Source из списка выбрать строку DataSet.

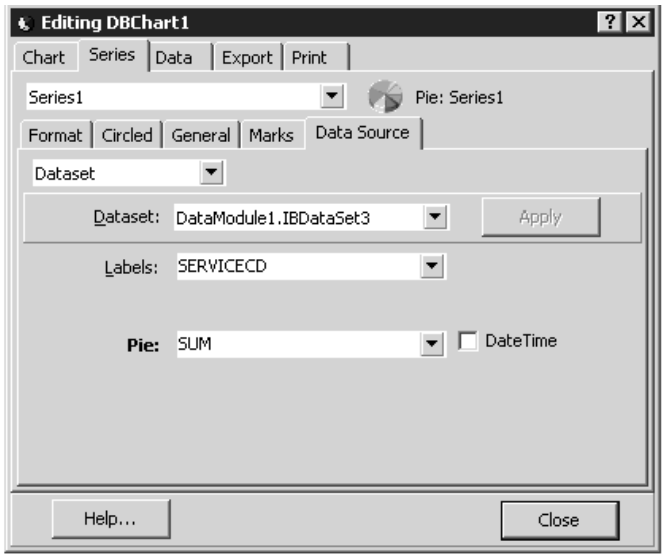


Рис. 4.14. *Выбор набора данных*

Далее в появившемся списке DataSet выбрать нужный набор данных (для рассматриваемого примера выбираем IBDataset3 с запросом по таблице Nachislsumma).

3. Далее необходимо связать нужные поля набора данных с осями координат или, в случае рассматриваемой круговой диаграммы, сопоставить поля для разбиения диаграммы на части.

В списке Labels выбираем поле ServiceCD, которое привязывает его значения (коды услуг) в виде меток к частям круговой диаграммы. В списке Pie выбираем поле SUM, которое дает суммы, начисленные по каждой из услуг. Именно от значений данного поля будет зависеть размер каждой части круговой диаграммы.

4. После выполнения вышеописанных действий необходимо открыть набор данных, и компонент TDBChart построит график (рис. 4.15).

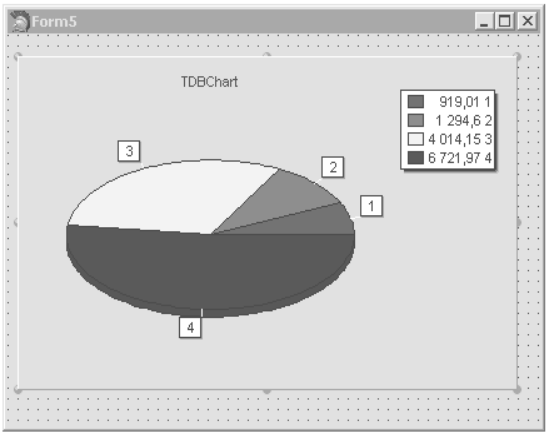


Рис. 4.15. *Компонент TDBChart*

Аналогичным образом на этот же компонент можно поместить и другие графики. Также можно настроить вывод данных в наиболее удобном и наглядном виде. Можно усложнить запрос набора данных для вывода не просто кодов, а наименования услуг.

При выполнении приложения диаграмма выглядит так же, как и при проектировании. При этом ее функционирование является динамическим, т. е. при изменении данных, содержащихся в наборе, диаграмма изменяется автоматически [4].

Для компонента TDBChart характерны три основных свойства [6]:

AutoRefresh типа Boolean. Свойство разрешает или запрещает обновление данных в серии при открытии связанного набора данных.

RefreshInterval типа LongInt. Свойство задает временной интервал в секундах между обновлениями данных в сериях из связанных наборов данных.

ShowGlassCursor типа Boolean. Свойство разрешает показ курсора "песочные часы" при обновлении данных.

Среди методов компонента можно отметить следующие [6]:

- процедура *CheckDataSource* обновляет данные в сериях;
- функция *IsValidDataSource (ASeries:TChartSeries; AComponent: Tcomponent)* проверяет, связан ли набор данных AComponent с серией ASeries, и возвращает значение типа Boolean;
- процедура *RefreshData* обновляет данные во всех сериях;
- процедура *RefreshDataSet (ADataSet : TDataSet; ASeries: TChartSeries)* считывает все записи в наборе данных AdataSet и переносит их в серию ASeries.

Для компонента можно задать обработчик события OnProcessRecord, который вызывается при переносе данных из отдельной записи набора данных в серию.

Таким образом, нами были рассмотрены визуальные компоненты отображения и редактирования данных, которые играют важную роль при создании интерфейсов приложений БД.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для каких целей предназначены визуальные компоненты для работы с данными?
2. Нарисуйте и кратно поясните схему, отражающую классификацию визуальных компонентов для работы с данными.
3. Какие основные свойства характерны для визуальных компонентов для работы с данными БД?
4. Что представляет собой компонент TDBGrid? Перечислите наиболее важные свойства компонента.
5. Опишите особенности статических и динамических столбцов компонента TDBGrid. В чем состоят преимущества использования статических столбцов?
6. Как осуществляется работа со столбцами в редакторе столбцов компонента TDBGrid?
7. Что представляет собой компонент TDBCtrlGrid? Какой метод предоставляет дополнительные возможности управления набором данных при использовании компонента TDBCtrlGrid?
8. Опишите основное назначение и особенности использования компонента TDBNavigator.
9. Как осуществляется работа с полями набора данных с помощью компонентов TDBText и TDBEdit?
10. Какие компоненты используются для представления значений полей типа Memo?

11. Опишите особенности использования компонента TDBCheckBox.

12. Какие компоненты используются для выбора значений полей набора данных из списка? Опишите подробно особенности компонента, представляющего собой группу зависимых переключателей.

13. Опишите механизм синхронного просмотра данных и особенности компонентов синхронного просмотра.

14. Опишите особенности использования компонента TDBImage.

15. Опишите процесс представления данных набора данных в виде графика с помощью компонента TDBChart

5. Пример разработки приложения для работы с базой данных

Рассмотрим пример разработки приложения БД, демонстрирующего теоретический материал, описанный в настоящем пособии.

Приложение разрабатывается для работы с учебной базой данных, являющейся упрощенной моделью БД системы "Абонент+". Описание учебной БД можно найти в приложении А, а скрипт для ее создания – в приложении Б. Для разработки приложения используется среда Delphi (Code Gear Delphi 2007) и стандартные компоненты IBX. Запросы выполнены на SQL СУБД Firebird 2.5.

Целью разрабатываемого приложения является демонстрация на практике общих принципов построения приложений БД, работы с наборами данных и со специализированными компонентами.

Демонстрационное приложение должно реализовать следующий функционал:

- размещение невизуальных компонентов в модуле данных;
- диалог авторизации при запуске;
- просмотр, редактирование, сортировка, поиск данных об абонентах на главной форме приложения;
- просмотр данных об истории оплат/начислений на отдельной форме;
- ввод оплаты на отдельной форме.

Запустим среду Code Gear Delphi и создадим новый проект под названием Abonents.

5.1. Разработка модуля данных

Разработку приложения начнем с добавления в проект специальной невизуальной формы – модуля данных. Использование модуля данных позволит в разрабатываемом приложении разделить логику работы с данными и пользовательский интерфейс. С помощью команды File/New/Other/DelphiFiles /DataModule создадим модуль данных AbonentDataModule, на котором будем располагать компоненты доступа к данным и компоненты-источники данных. Сначала расположим на форме модуля компоненты с вкладки InterBase – компонент TIBDatabase для соединения с базой данных, компонент транзакции TIBTransaction и набор данных TIBDataSet, соответствующий таблице Abonent с данными об абонентах, – и источник данных TDataSource для связи с набором данных с вкладки DataAccess. По мере разработки приложения будет добавлять необходимые наборы данных. Именовать компоненты будем исходя из названий соответствующих таблиц либо самого проекта, предваренных префиксом: id – для компонентов TIBDataset, iq – для компонентов TIBQuery, ds – для TDataSource, ita – для TIBTransaction, idb – для

TIBDatabase. Переименуем размещенные компоненты (путем изменения свойства Name) в idbAbonents, itaAbonents, idAbonent и dsAbonent.

На рис. 5.1 представлена форма модуля данных на начальном этапе разработки приложения.

В первую очередь настроим компонент соединения idbAbonents: TIBDatabase. В параметрах соединения idbAbonents укажем только путь к учебной БД, не заполняя пользователя и пароль. В качестве DefaultTransaction (компонент для работы с транзакциями по умолчанию) укажем itaAbonents.

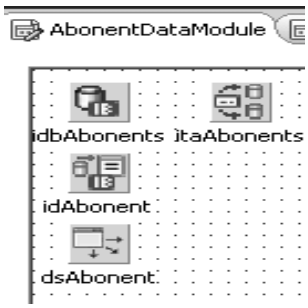


Рис. 5.1. *Форма модуля данных в начале разработки*

В компоненте itaAbonents необходимо установить свойство DefaultDatabase, указав в нем имя компонента idbAbonents.

Для компонента idAbonent типа TIBDataset заполним свойства Database и Transaction для подключения к БД, а в свойстве SelectSQL запишем запрос:

```
select * from Abonent
```

Используя редактор наборов данных, определим действия по редактированию наборов данных. Для этого выберем ключевые и редактируемые поля в редакторе. Например, для набора данных idAbonent редактор с заполненными полями будет выглядеть, как показано на рис. 5.2.

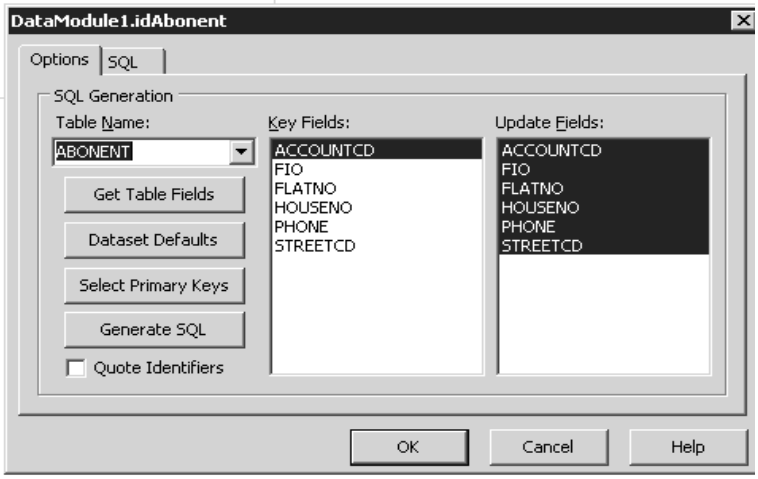


Рис. 5.2. *Редактор набора данных idAbonent*

По нажатию GenerateSQL будут созданы SQL-запросы для обновления набора данных, которые можно просмотреть и отредактировать на вкладке SQL либо в свойствах DeleteSQL, InsertSQL, ModifySQL.

Автоматически созданные запросы представлены ниже.

DeleteSQL:

```
delete from ABONENT
where
  ACCOUNTCD = :OLD_ACCOUNTCD
```

InsertSQL:

```
insert into ABONENT
  (ACCOUNTCD, FIO, FLATNO, HOUSENO, PHONE, STREETCD)
values
  (:ACCOUNTCD, :FIO, :FLATNO, :HOUSENO, :PHONE, :STREETCD)
```

ModifySQL:

```
update ABONENT
set
  ACCOUNTCD = :ACCOUNTCD,
  FIO = :FIO,
  FLATNO = :FLATNO,
  HOUSENO = :HOUSENO,
  PHONE = :PHONE,
  STREETCD = :STREETCD
where
  ACCOUNTCD = :OLD_ACCOUNTCD
```

Для источника данных dsAbonent типа TDataSource укажем набор данных idAbonent в свойстве Dataset.

При изложении теоретического материала было отмечено, что модуль данных имеет события OnCreate и OnDestroy, в которых имеет смысл предусмотреть соответственно операторы открытия соединений с наборами данных БД и закрытия этих соединений.

Ниже приведены коды двух обработчиков событий (на данном этапе разработки) для установления и разрыва соединений с компонентами модуля AbonentDataModule.

```
procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  idbAbonents.Connected:=True;
  itaAbonents.Active:=True;
  idAbonent.Open;
end;

procedure TDataModule1.DataModuleDestroy(Sender: TObject);
begin
```

```
  itaAbonents.Active:=False;
  idbAbonents.Connected:=False;
  idAbonent.Close;
end;
```

После окончания предварительной разработки модуля данных выберем пункт меню Project/Options/Forms и перетащим мышью созданный модуль AbonentDataModule из списка Available Forms на первое место списка Auto-Create Forms. Это необходимо для инициализации модуля данных (и невизуальных компонентов, расположенных в нем) до создания остальных форм приложения.

5.2. Создание диалога авторизации

Приступим к разработке интерфейсной части проекта. Разрабатываемое приложение должно при запуске программы выдавать диалог авторизации, и в случае успешной авторизации пользователя должна открываться главная форма проекта, на которой выводится список абонентов.

Ранее, при описании компонента TIBDatabase было упомянуто свойство LoginPrompt, которое определяет, выдавать ли диалог подключения к базе. Если LoginPrompt = True, то при попытке подключения к базе выдается стандартный диалог. Однако компонент TIBDatabase имеет событие OnLogin, позволяющее заменить стандартный диалог запроса пользователя и пароля [12].

Продемонстрируемом данную возможность в разрабатываемом приложении.

С помощью команды File/New/Form создадим форму-диалог авторизации TfrmConnect (рис. 5.3).

Рис. 5.3. *Диалог авторизации*

Форма TfrmConnect содержит поля TEdit с вкладки Standart для ввода имени пользователя и пароля. Данная форма могла бы содержать также строку ввода (и диалог выбора) пути к файлу БД, так как в большинстве случаев константное имя файла БД, указанное в свойстве компонента TIBDatabase на этапе разработки, затрудняет перенос приложения на другие компьютеры. Однако в разрабатываемом приложении оставим все-таки путь к базе в виде константы.

В обработчиках событий нажатия кнопок пропишем коды.

```
procedure TfrmConnect.Button1Click(Sender: TObject);
begin
  Close;
  ModalResult:=mrOK;
end;

procedure TfrmConnect.Button2Click(Sender: TObject);
begin
  Close;
  ModalResult:=mrCancel;
  Application.Terminate;
end;
```

Из обработчиков видно, что по нажатию кнопки OK форма авторизации закрывается и свойству ModalResult присваивается соответствующее значение для последующего анализа, а при нажатии Отмена завершается работа приложения.

Далее для компонента idbAbonents, размещенном в модуле данных, установим LoginPrompt = True и определим обработчик события OnLogin:

```
procedure TDataModule1.idbAbonentsLogin(Database: TIBDatabase;
  LoginParams: TStrings);
var
  dlg: TfrmConnect; // диалог авторизации
  res : TModalResult;
begin
  dlg:=TfrmConnect.Create(Application);
  res := dlg.ShowModal;
  if res=mrOK then
  begin
    with LoginParams do
    begin
      Values['USER_NAME'] := dlg.User_Name.text;
      Values['PASSWORD'] := dlg.User_Pass.Text;
    end;
  end; //mrOK
  dlg.Free;
end;
```

Таким образом, при попытке соединения с базой пользователю предлагается диалоговое окно авторизации. Если пользователь после работы с этим окном щелкнул в нем на кнопке OK, наступает событие OnLogin, обработчиком которого является процедура idbAbonentsLogin. В ней в качестве параметров соединения устанавливаются имя пользователя и пароль, введенные в диалоге авторизации. Если соединение успешно, то открывается главная форма проекта. Если авторизация не пройдена, необходимо выдать сообщение и завершить работу приложения.

Для этого код обработчика OnCreate модуля данных необходимо изменить:

```
procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  try
    idbAbonents.Connected:=True;
  except
    Application.MessageBox('Невозможно подключиться к базе!',
'Приложение закрывается', MB_ICONSTOP);
    Application.Terminate;
  end;
  itaAbonents.Active:=True;
  idAbonent.Open;
end;
```

5.3. Разработка главной формы приложения

При создании проекта была одновременно создана форма приложения Form1, переименуем ее в AbonentMainForm, а в свойстве Caption укажем Абоненты.

Отметим, что посмотреть состав разрабатываемого проекта можно, используя пункт меню View/Project Manager (рис. 5.4).

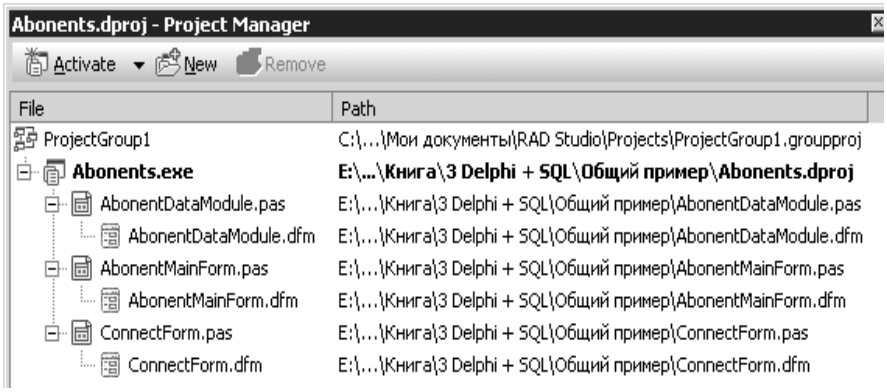


Рис. 5.4. *Менеджер проекта*

Для того, что созданный модуль данных был доступен для формы, в секцию implementation необходимо добавить код:

```
uses AbonentDataModule;
```

Данный код необходимо прописывать для каждой вновь создаваемой в проекте формы.

На форму поместим главное меню AbonentMainMenu (компонент TMainMenu с вкладки Standart) и в свойство Items добавим пункт Выход для закрытия приложения.

Также на форму поместим компонент TDBGrid с вкладки Data Controls для отображения набора данных idAbonent и устанавим его свойство DataSource в значение DataModule1.dsAbonent.

Компоненту типа TDBGrid присвоим имя dgAbonent.

В таблице Abonent поле StreetCD хранит код улицы, на которой проживает абонент. Для удобства просмотра выведем в списке абонентов вместо кода название улицы. Данная возможность уже была кратко описана (см. стр. 44), здесь рассмотрим подробнее. Для создания поля синхронного просмотра (lookup-поля) выполним 3 шага.

1. Добавим в модуль данных еще один набор данных itStreet типа TIBTable с данными справочника улиц (таблица Street).

2. В наборе данных idAbonent создадим поле синхронного просмотра (поле подстановки) [6]. Откроем редактор полей (Field Editor) компонента idAbonent, добавим новое поле StreetNM и определим его свойства (рис. 5.5):

LookupDataSet = itStreet (свойство задает набор данных синхронного просмотра).

LookupResultField = StreetNM (представляет поле синхронного просмотра из набора данных LookupDataSet, значения которого будут появляться в созданном поле).

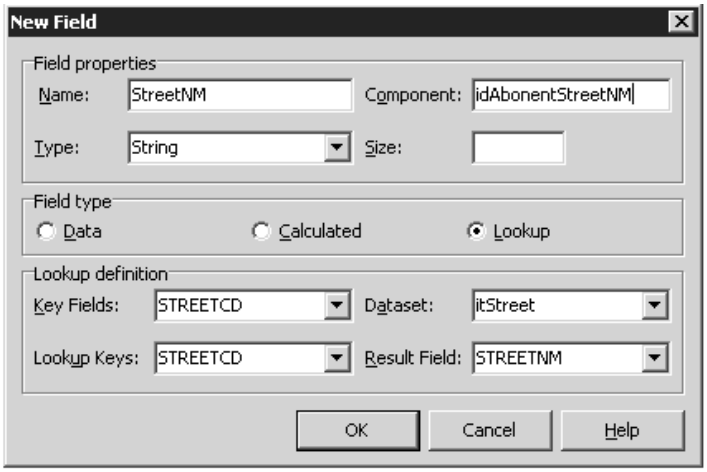


Рис. 5.5. *Добавление Lookup-поля*

LookupKeyFields = StreetCD (содержит поле (или поля) из набора данных синхронного просмотра, по значению которого выбирается значение из поля LookupResultField).

KeyFields = StreetCD (определяет поле (или поля) из исходного набора данных, для которого создается поле синхронного просмотра).

3. Для компонента TDBGrid создадим статический список столбцов с помощью Редактора столбцов. Сначала добавим в список все столбцы набора данных idAbonent, затем удалим объект-столбец StreetCD из списка, после чего информация по улице будет представлена только данными столбца StreetNM (рис. 5.6). Для удобства настроим отображение, например, заголовка столбца AccountCD путем определения свойств Title.Alignment = taCenter и Title.Caption = Лицевой счет. Для остальных столбцов зададим свойства аналогично.

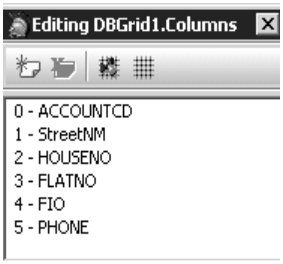


Рис. 5.6. *Редактор столбцов компонента TDBGrid*

После выполнения указанных шагов в списке абонентов вместо кода улицы будет отображаться ее название (см. рис. 5.8).

Редактирование данных об абонентах возможно непосредственно в TDBGrid, так как выполнены некоторые условия:

- набор данных idAbonent является редактируемым;
- для набора данных idAbonent заполнены свойства ModifySQL, InsertSQL, DeleteSQL;
- для компонента TDBGrid свойство ReadOnly = False, Editing = True.

Для удобства навигации и редактирования данных разместим на форме компонент TDBNavigator и заполним его свойство DataSource = DataModule1.dsAbonent.

Таким образом, мы реализовали возможность просмотра и редактирования данных об абонентах на главной форме приложения.

Среди задач, поставленных перед приложением в начале раздела 5, стоит задача реализации сортировки и поиска данных об абонентах на главной форме приложения, просмотр данных об истории оплат/начислений на отдельной форме.

С главной формы приложения AbonentMainForm предусмотрим возможность выполнения пользователем:

- сортировки списка абонентов по номеру лицевого счета или по ФИО;
- поиска абонентов по заданному номеру лицевого счета;

- просмотра для выбранного абонента формы с информацией по оплатам и начислениям.

Для выполнения перечисленных действий разместим на форме визуальные компоненты:

- компонент типа TRadioGroup (с меткой "Сортировать") для задания пользователем вариантов сортировки (без сортировки, по номеру лицевого счета, по ФИО);
- компонент eAccountCD типа Tedit для ввода пользователем номера лицевого счета абонента и кнопку bLocate (с меткой "Поиск") для поиска данных;
- кнопку bSaldo (с меткой "История оплат и начислений") для открытия формы с историей оплат и начислений для текущего абонента списка TDBGrid, а также создадим дополнительный пункт в главном меню формы для открытия истории (продублируем возможность в меню).

Главная форма проекта на этапе разработки показана на рис. 5.7.

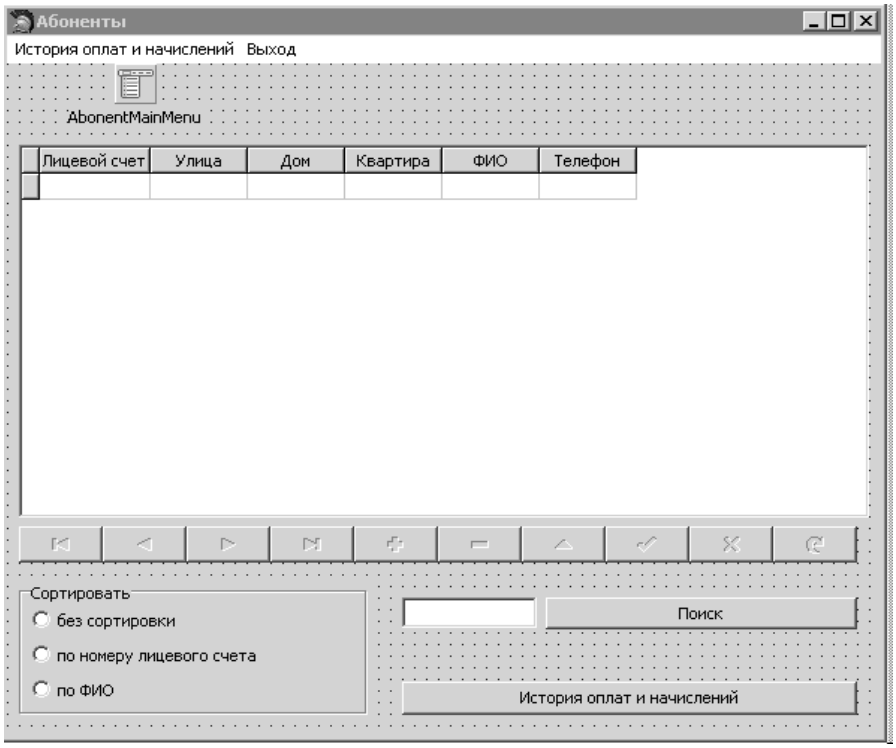


Рис. 5.7. *Главная форма проекта на этапе разработки*

После запуска приложения главная форма проекта примет вид, показанный на рис. 5.8.

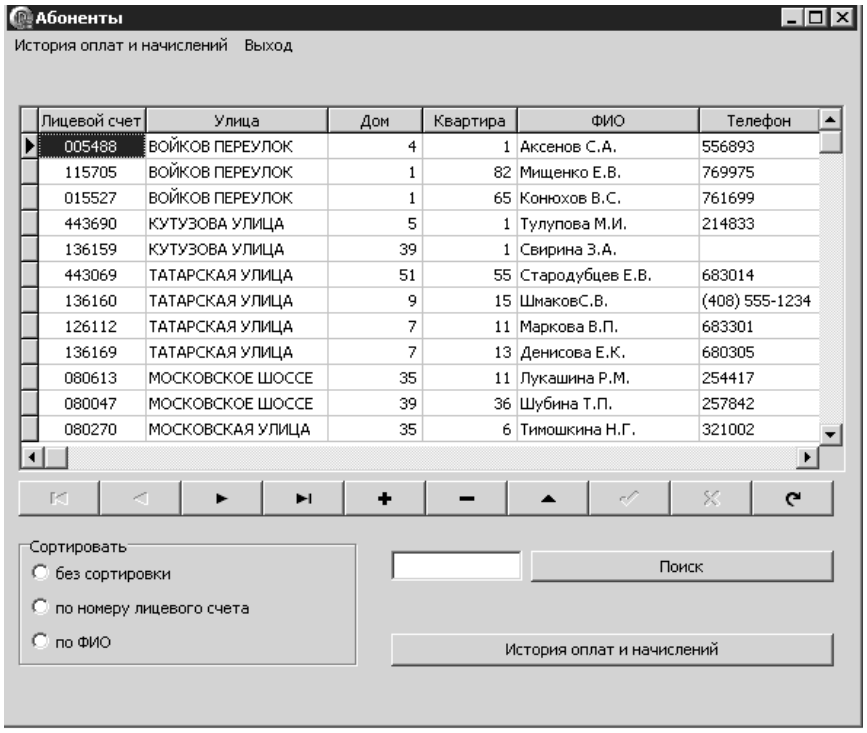


Рис. 5.8. *Главная форма проекта на этапе выполнения программы*

Следует отметить, что на этапе отладки приложения можно установить в Инспекторе объектов свойство Active = True для требуемых наборов данных, и тогда данные в визуальных компонентах в статическом виде будут отображаться уже на этапе разработки.

Реализуем возможность сортировки данных по выбранному полю. Для этого напишем программный код в обработчике OnClick группы переключателей:

```
procedure TfrmAbonentMain.RadioGroup1Click(Sender: TObject);
var
  sort: string;
begin
  AbonentDataModule.DataModule1.idAbonent.Close;
  AbonentDataModule.DataModule1.idAbonent.SelectSQL.Clear;
```

```
  AbonentDataModule.DataModule1.idAbonent.SelectSQL.Add
('SELECT * FROM Abonent');
  case RadioGroup1.ItemIndex of
  //устанавливаем режим сортировки
  0: sort := '';
  1: sort := 'ORDER BY AccountCD';
  2: sort := 'ORDER BY FIO';
  end;
  AbonentDataModule.DataModule1.idAbonent.SelectSQL.Add(sort);
  AbonentDataModule.DataModule1.idAbonent.Open;
end;
```

В данной процедуре программно устанавливается SQL-запрос для компонента idAbonent в зависимости от выбранного пользователем варианта сортировки. Набор данных перед этим закрывается методом Close, а после установки запроса открывается вновь с помощью метода Open.

Форма с данными, отсортированными по ФИО, представлена на рис. 5.9.

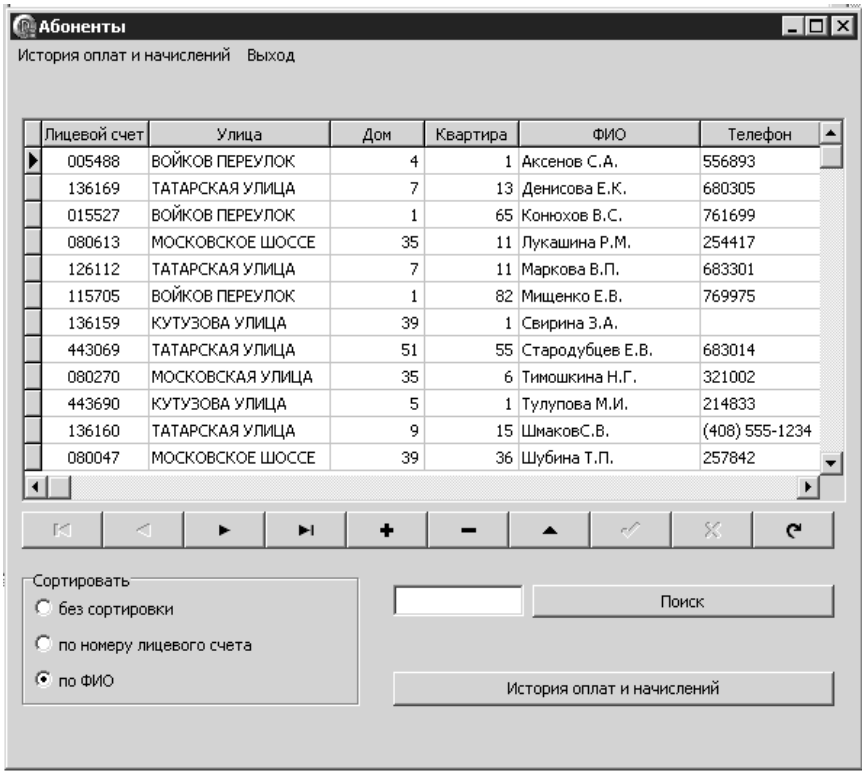


Рис. 5.9. *Сортировка данных по ФИО*

Реализуем поиск абонента по заданному номеру лицевого счета. Поиск будет выполняться после ввода значения в поле eAccountCD и по последующему нажатию кнопки Поиск на главной форме приложения. Для выполнения поиска напишем программный код в обработчике события OnClick кнопки:

```
procedure TfrmAbonentMain.bLocateClick(Sender: TObject);
begin
  if eAccountCD.Text <> '' then
    if not AbonentDataModule.DataModule1.idAbonent.Locate('AccountCD',
eAccountCD.Text, []) then
      ShowMessage('Запись не найдена!')
  else
    ShowMessage('Введите номер лицевого счета!');
end;
```

В данной процедуре выполняется поиск с помощью метода Locate, причем поиск ведется по полному совпадению номера лицевого счета. На найденную запись устанавливается указатель в DBGrid, либо выдается сообщение о неудачном поиске.

5.4. Разработка формы истории оплат и начислений

Приступим к разработке формы с историей оплат и начислений. История оплат и начислений абонента должна содержать:

- информация об абоненте: ФИО, телефон, адрес (из таблицы Abonent);
- месяц и год истории (на основе данных таблиц NachislSumma и PaySumma вычисляется хранимой процедурой);
- остаток на начало месяца (на основе данных таблиц NachislSumma и PaySumma вычисляется хранимой процедурой);
- сумма начислений (на основе данных таблицы NachislSumma вычисляется хранимой процедурой);
- сумма оплат (на основе данных таблицы PaySumma вычисляется хранимой процедурой);
- остаток на конец месяца (на основе данных таблиц NachislSumma и PaySumma вычисляется хранимой процедурой).

Создадим форму frmSaldo, которая будет открываться по нажатию кнопки bSaldo главной формы и отображать историю для текущего абонента из списка на главной форме. Обработчик события OnClick кнопки bSaldo приведен ниже.

```
procedure TfrmAbonentMain.bSaldoClick(Sender: TObject);
var
  SaldoForm: TfrmSaldo; // форма истории оплат и начислений
begin
```

```
  SaldoForm := TfrmSaldo.Create(Self);
  SaldoForm.ShowModal;
end;
```

Данные об истории оплат и начислений абонента будем получать на основе двух ХП. Процедура выбора Abonent_Balance по заданному номеру лицевого счета, месяцу и году выводит информацию об абоненте и данные по денежным суммам абонента: сальдо (разница между начислением и оплатой) на начало месяца, сумма начислений, сумма оплат, сальдо на конец месяца. Процедура Abonent_Balance_Period вызывает процедуру Abonent_Balance для получения данных по каждому месяцу и выводит информацию по сальдо за заданный период.

Разработка данных процедур и отчета на основе данных, возвращаемых ими, была подробно рассмотрена в учебном пособии [2]. Коды процедур приведены в приложении В. В настоящем пособии покажем, как вывести данные на отдельной форме, организовать ввод значений в одну из таблиц, от которых зависят процедуры, и увидеть изменение данных.

На форму модуля данных поместим новый компонент транзакции itaSaldo, новый набор данных idSaldo типа TIBDataset и источник данных dsSaldo для выборки информации по сальдо. В свойстве Transaction компонента idSaldo установим транзакцию itaSaldo, тем самым отделив управление отображением сальдо от остальных действий в приложении. В свойстве SelectSQL компонента idSaldo запишем запрос, выбирающий данные из процедуры Abonent_Balance_Period:

```
SELECT
  FIO, Address, Phone,
  Cur_month, Cur_year,
  BegSaldo, Nachisleno,Oplacheno, EndSaldo
FROM
  Abonent_Balance_Period (:AccountCD, :BegMes, :BegGod, :EndMes, :EndGod)
ORDER BY
  Cur_year desc, Cur_month desc;
```

Поместим на форму frmSaldo компоненты eFIO типа TDBEdit (ФИО абонента), ePhone типа TDBEdit (телефон), eAddress типа TDBEdit (адрес) с соответствующими метками и компонент TDBGrid1 для вывода информации об истории начислений и оплат абонента.

Для всех компонентов укажем в качестве источника данных DataModule1.dsSaldo, а для компонентов типа TDBEdit также укажем поля набора данных (FIO, Phone, Address). В компоненте TDBGrid откроем редактор столбцов компонента, удалим данные поля из списка (для исключения дублирования информации) и переопределим названия столбцов.

Также поместим на форму кнопки bPay и bClose для ввода оплаты и закрытия формы соответственно.

Форма истории оплат и начислений на этапе разработки приведена на рис. 5.10.

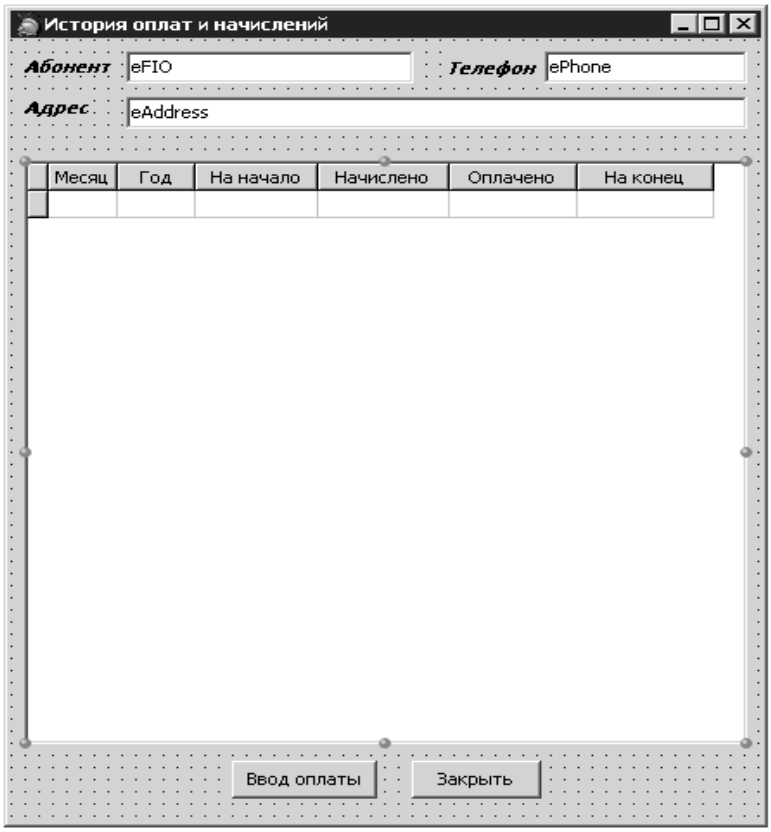


Рис. 5.10. *Форма истории оплат и начислений на этапе разработки*

Для успешной работы запроса SELECT необходимо перед его выполнением присвоить значения входным параметрам процедуры.

В качестве параметра AccountCD будем брать номер лицевого счета абонента, текущего в списке на главной форме на момент создания формы истории оплат. Параметры EndMes и EndGod будем получать из текущей даты. Параметры BegMes и BegGod необходимо получать в соответствии с историей конкретного абонента.

Для нахождения месяца и года начала истории для конкретного абонента поместим на форму модуля данных компонент iqMinMonth типа TIBQuery и запишем в его свойстве SQL запрос:

```
SELECT
  IIF(Nacn_.Min_month > Pay_.Min_month, Pay_.Min_month, Nacn_.Min_month)
as Min_month,
  Min_god.Min_year as Min_year
FROM
  (SELECT
    IIF(Nacn.Min_year > Pay.Min_year, Pay.Min_year, Nacn.Min_year)
  FROM
    (SELECT N.AccountCD, Min(N.Nachislyear)
    FROM Nachislsumma N
    GROUP BY N.AccountCD
    HAVING N.AccountCD = :AccountCD
    )Nacn (AccountCD, Min_year)
    INNER JOIN
    (SELECT P.AccountCD, Min(P.Payyear)
    FROM Paysumma P
    GROUP BY P.AccountCD
    HAVING P.AccountCD = :AccountCD
    )Pay (AccountCD, Min_year)
    ON (Nacn.AccountCD = Pay.AccountCD)
  ) Min_god (Min_year)
  INNER JOIN
  (SELECT N.Nachislyear, Min(N.Nachislmonth)
  FROM Nachislsumma N
  GROUP BY N.AccountCD, N.Nachislyear
  HAVING N.AccountCD = :AccountCD
  )Nacn_ (Min_year, Min_month)
  ON (Min_god.Min_year = Nacn_.Min_year)
  INNER JOIN
  (SELECT P.Payyear, Min(P.Paymonth)
  FROM Paysumma P
  GROUP BY P.AccountCD, P.Payyear
  HAVING P.AccountCD = :AccountCD
  )Pay_ (Min_year, Min_month)
  ON (Min_god.Min_year = Pay_.Min_year)
```

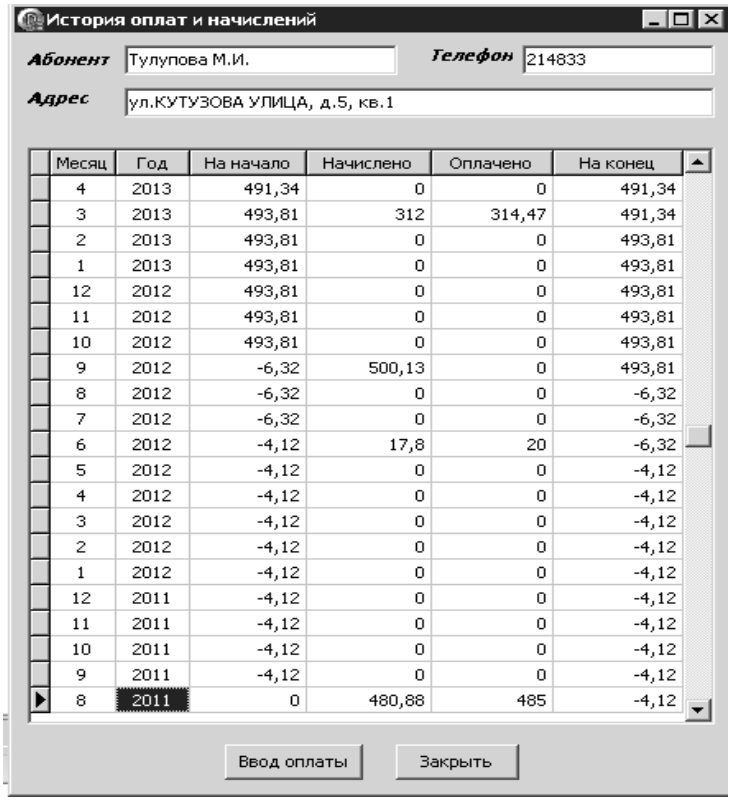
Приведенный запрос сначала находит самый ранний (наименьший) год среди тех, за которые у конкретного абонента (параметр :AccountCD) были начисление и/или оплата, затем находит самые ранние месяца в этом году в таблице начислений и в таблице оплат и выбирает среди них наименьший.

Значения всех необходимых для работы процедуры Abonent_Balance _Period (и формирования набора idSaldo) параметров будем получать в обработчике события OnCreate формы frmSaldo и сохранять их в переменных формы. Для формирования выборки данных по сальдо создадим отдельную процедуру OpenSaldo. Коды процедур представлены ниже.

```
procedure TfrmSaldo.OpenSaldo(AccountCD: string; BegMonth,BegYear,
EndMonth,EndYear: Word);
begin
  //работа с набором данных
  AbonentDataModule.DataModule1.idSaldo.Close;
  //заполнение параметров
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('AccountCD').
AsString := AccountCD;
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('BegMes').
AsInteger := BegMonth;
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('BegGod').
AsInteger := BegYear;
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('EndMes').
AsInteger := EndMonth;
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('EndGod').
AsInteger := EndYear;
  //открытие набора данных
  AbonentDataModule.DataModule1.idSaldo.Open;
end;
procedure TfrmSaldo.FormCreate(Sender: TObject);
begin
  //номер лицевого счета абонента из текущей строки списка главной формы
  AccountCD := AbonentMainForm.frmAbonentMain.DBGrid1.DataSource.DataSet.
FieldByName('AccountCD').AsString;
  //получение месяца и года начала истории
  AbonentDataModule.DataModule1.iqMinMonth.Close;
  AbonentDataModule.DataModule1.iqMinMonth.ParamByName
('AccountCD').AsString := AccountCD;
  AbonentDataModule.DataModule1.iqMinMonth.Open;
  BegYear := AbonentDataModule.DataModule1.iqMinMonth.FieldByName
('Min_year').AsInteger;
  BegMonth := AbonentDataModule.DataModule1.iqMinMonth.FieldByName
('Min_month').AsInteger;
  //разбиение текущей даты
  DecodeDate(Now,CurYear,CurMonth,CurDay);
  //формирование набора данных с сальдо
  OpenSaldo(AccountCD,BegMonth,BegYear,CurMonth,CurYear);
end;
```

В вышеприведенных процедурах продемонстрировано, как может быть выполнено программное задание значений параметров для наборов данных, а также возврат результата запроса в переменную.

Форма истории оплат и начислений на этапе выполнения, например для абонента с номером лицевого счета ‘443690’, показана на рис. 5.11.



Месяц	Год	На начало	Начислено	Оплачено	На конец
4	2013	491,34	0	0	491,34
3	2013	493,81	312	314,47	491,34
2	2013	493,81	0	0	493,81
1	2013	493,81	0	0	493,81
12	2012	493,81	0	0	493,81
11	2012	493,81	0	0	493,81
10	2012	493,81	0	0	493,81
9	2012	-6,32	500,13	0	493,81
8	2012	-6,32	0	0	-6,32
7	2012	-6,32	0	0	-6,32
6	2012	-4,12	17,8	20	-6,32
5	2012	-4,12	0	0	-4,12
4	2012	-4,12	0	0	-4,12
3	2012	-4,12	0	0	-4,12
2	2012	-4,12	0	0	-4,12
1	2012	-4,12	0	0	-4,12
12	2011	-4,12	0	0	-4,12
11	2011	-4,12	0	0	-4,12
10	2011	-4,12	0	0	-4,12
9	2011	-4,12	0	0	-4,12
8	2011	0	480,88	485	-4,12

Рис. 5.11. *Форма истории оплат и начислений на этапе выполнения*

На рис. 5.11 показано начало истории оплат и начислений абонента ‘Тулупова М.И.’ (конец списка, так как данные отсортированы по убыванию), и можно видеть, история началась с августа 2011 года.

5.5. РАЗРАБОТКА ФОРМЫ ВВОДА ОПЛАТЫ

Реализуем возможность ввода оплаты с формы. Данные, отображаемые в DBGrid1, не являются редактируемым набором данных, поэтому непосредственно вводить сумму оплаты в строках таблицы нельзя.

Добавим в проект форму frmPay для ввода оплаты, которая будет вызываться в обработчике OnClick кнопки bPay формы истории оплат и начислений. Код обработчика приведен ниже.

```
procedure TfrmSaldo.bPayClick(Sender: TObject);
var
  PayForm: TfrmPay; // форма ввода оплаты
```

```
begin
  PayForm := TfrmPay.Create(Self);
  if PayForm.ShowModal = mrOK then
  begin
    //обновляем данные на форме сальдо
    AbonentDataModule.DataModule1.itaSaldo.Commit;
    OpenSaldo(AccountCD,BegMonth,BegYear,CurMonth,CurYear);
  end;
end;
```

В обработчике предусмотрено обновление информации на форме истории оплат сразу после добавления факта оплаты в базу данных.

Необходимо помнить, что все действия в приложении выполняются в рамках транзакций. Поэтому недостаточно после изменения информации в БД (в данном случае – после добавления оплаты в таблицу PaySumma) только обновить данные набора данных. Для просмотра обновленной информации в приложении следует завершить активную транзакцию, а потом переформировать набор данных (закрытие/открытие которого вызовет автоматически старт новой транзакции).

На форме модуля данных создадим набор данных idPaySum, который будет содержать все данные таблицы PaySumma, и источник данных dsPaySum. Также создадим отдельную транзакцию itaPaySum, которую установим в свойстве Transaction компонента idPaySum. Новая транзакция необходима, чтобы разделить управление добавлением факта оплаты в базу от остальных действий в приложении.

На рис. 5.12 представлена форма модуля данных на окончательном этапе разработки демонстрационного приложения.

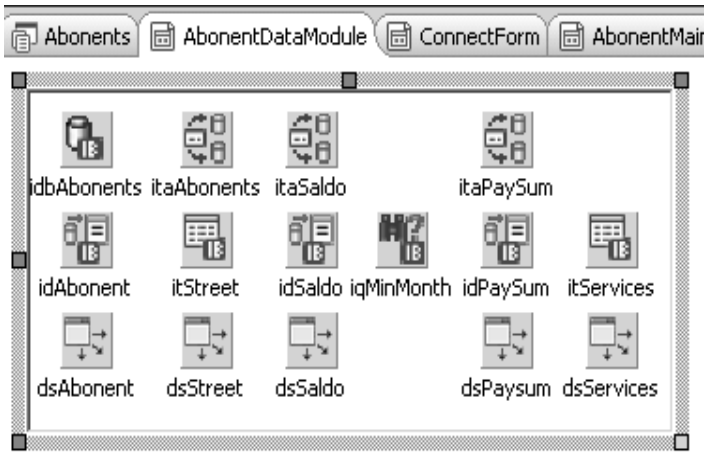


Рис. 5.12. *Форма модуля данных*

Для набора данных idPaySum, используя редактор наборов данных, создадим SQL-запросы на редактирование данных, в том числе запрос на добавление записей в таблицу:

```
insert into PAYSUMMA
  (ACCOUNTCD, PAYDATE, PAYFACTCD, PAYMONTH, PAYSUM, PAYYEAR,
SERVICECD)
values
  (:ACCOUNTCD, :PAYDATE, :PAYFACTCD, :PAYMONTH, :PAYSUM,
:PAYYEAR, :SERVICECD)
```

Большое значение при работе с редактируемыми наборами данных имеет использование генераторов значений [16]. Для корректного добавления оплат в базу данных создадим в базе генератор Gen_PayFactCD для поля PayFactCD таблицы PaySumma и установим его значение равным 78 (в таблице PaySumma 77 записей). Код представлен ниже.

```
CREATE SEQUENCE Gen_PayFactCD;
ALTER SEQUENCE Gen_PayFactCD RESTART WITH 78;
```

Далее для набора данных idPaySum заполним свойство GeneratorField, указав имя генератора Gen_PayFactCD, имя ключевого поля PayFactCD и событие OnNewRecord (рис. 5.13).

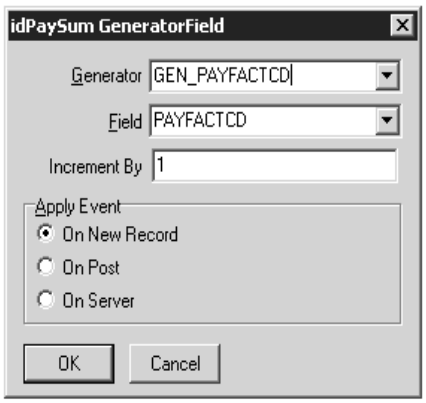


Рис. 5.13. *Заполнение свойства GeneratorField*

Разработаем интерфейс формы ввода оплаты frmPay. На форму frmPay поместим компоненты lcFIO и lcService типа TDBLookupCombopBox для ввода ФИО абонента и названия услуги, компоненты deMonth и deYear типа TDBEdit для ввода месяца и года, за который производится оплата, компонент deSum типа TDBEdit для ввода суммы оплаты и компонент TDateTimePicker1 с закладки Win32 для ввода даты оплаты.

Для всех компонентов, кроме TDateTimePicker1, укажем в качестве источника данных DataModule1.dsPaysum, для компонентов типа TDBEdit также укажем необходимые поля набора данных. Свойства компонента lcFIO настроим таким образом:

DataField = ACCOUNTCD;
DataSource = DataModule1.dsPaysum;
KeyField = ACCOUNTCD;
ListField = FIO;
ListSource = DataModule1.dsAbonent.

Свойства компонента lcService настроим по аналогии.

Также поместим на форму кнопки bAdd и bClose для добавления факта оплаты в базу данных и закрытия формы соответственно. Форма ввода оплаты на этапе разработки представлена на рис. 5.14.

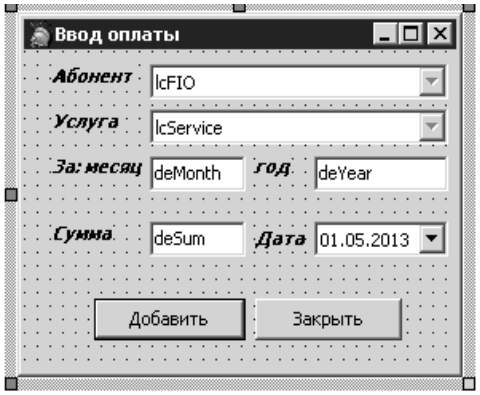


Рис. 5.14. *Форма ввода оплаты на этапе разработки.*

Ниже приведен обработчик события OnCreate формы, в котором программно добавляется новая "пустая" запись в таблицу оплат и в нее автоматически заносится номер лицевого счета.

```
procedure TfrmPay.FormCreate(Sender: TObject);
var
  AccountCD: string;
begin
  //переоткрываем набор данных, что влечет старт транзакции
  AbonentDataModule.DataModule1.idPaySum.Close;
  AbonentDataModule.DataModule1.idPaySum.Open;
  //добавляем пустую запись
  AbonentDataModule.DataModule1.idPaySum.Append;
  //автоматически заполняем номер лицевого счета
```

```
  AccountCD := AbonentMainForm.frmAbonentMain.DBGrid1.DataSource.DataSet.
FieldByName('AccountCD').AsString;
AbonentDataModule.DataModule1.idPaySum.FieldByName('AccountCD').
AsString := AccountCD;
end;
```

В приведенной процедуре перед добавлением пустой записи с помощью метода Append предварительно выполняется закрытие/открытие набора данных, что влечет старт транзакции itaPaySum. Затем данная транзакция либо успешно завершается при щелчке по кнопке bAdd (при успешном занесении всех заполненных пользователем полей в базу данных), либо откатывается при ошибке или щелчке по кнопке bClose. Свойству ModalResult присваивается соответствующее значение (mrOK или mrNone). Коды обработчиков OnClick для кнопок bAdd и bClose приведены ниже.

```
procedure TfrmPay.bAddClick(Sender: TObject);
begin
  try
    //заполняем запись по введенным пользователем значениям
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('ServiceCD').AsInteger := lcService.KeyValue;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PayMonth').AsInteger := deMonth.Field.Value;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PayYear').AsInteger := deYear.Field.Value;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PaySum').AsCurrency := deSum.Field.Value;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PayDate').AsDateTime := DateTimePicker1.Date;
    //подтверждаем изменения набора данных
    AbonentDataModule.DataModule1.idPaySum.Post;
    //добавить по заполненным пользователем полям
    AbonentDataModule.DataModule1.itaPaySum.Commit;
    ModalResult:=mrOK;
  except
    AbonentDataModule.DataModule1.itaPaySum.Rollback;
    ShowMessage('Неверно заполнены данные!');
  end;
end;
procedure TfrmPay.bCloseClick(Sender: TObject);
begin
  AbonentDataModule.DataModule1.itaPaySum.Rollback;
  ModalResult:=mrNone;
end;
```

Заполним, например, форму для ввода оплаты абоненту ‘Тулупова М.И.’ (рис. 5.15).

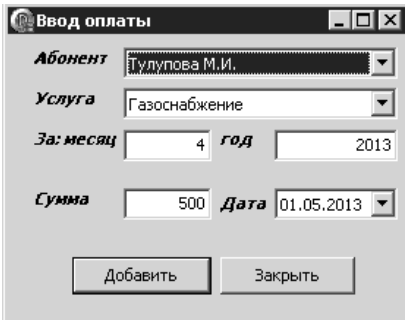
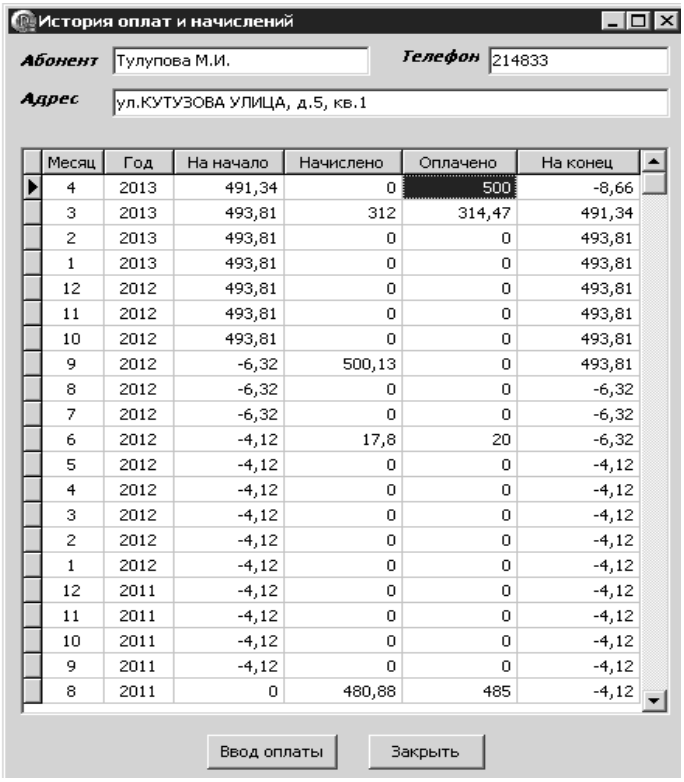


Рис. 5.15. *Ввод оплаты для абонента*

После щелчка по кнопке "Добавить" факт оплаты будет добавлен в базу данных. На форме истории оплат и начислений абонента ‘Тулупова М.И.’ сразу отобразится новая добавленная оплата (рис. 5.16).



Месяц	Год	На начало	Начислено	Оплачено	На конец
4	2013	491,34	0	500	-8,66
3	2013	493,81	312	314,47	491,34
2	2013	493,81	0	0	493,81
1	2013	493,81	0	0	493,81
12	2012	493,81	0	0	493,81
11	2012	493,81	0	0	493,81
10	2012	493,81	0	0	493,81
9	2012	-6,32	500,13	0	493,81
8	2012	-6,32	0	0	-6,32
7	2012	-6,32	0	0	-6,32
6	2012	-4,12	17,8	20	-6,32
5	2012	-4,12	0	0	-4,12
4	2012	-4,12	0	0	-4,12
3	2012	-4,12	0	0	-4,12
2	2012	-4,12	0	0	-4,12
1	2012	-4,12	0	0	-4,12
12	2011	-4,12	0	0	-4,12
11	2011	-4,12	0	0	-4,12
10	2011	-4,12	0	0	-4,12
9	2011	-4,12	0	0	-4,12
8	2011	0	480,88	485	-4,12

Рис. 5.16. *Отображение добавленной оплаты в истории*

И так, мы подробно рассмотрели пример разработки демонстрационного приложения учебной БД в среде Delphi.

Но помимо отображения данных базы в визуальных компонентах при разработке приложений практически всегда возникает необходимость формирования отчетности. Настоящее пособие не содержит примеров формирования отчетов. Подробную информацию о разработке отчетов в генераторе FastReport из среды Delphi можно найти в учебном пособии [2], а в учебном пособии [3] рассмотрена разработка отчетов в текстовых процессорах MS Word и OpenOffice Writer.

Библиографический список

1. Маркин А. В. Построение запросов и программирование на SQL. Учебное пособие. 2-е изд., перераб. и допол. М.: Диалог-МИФИ, 2011.

2. Маркин А.В. Создание отчетов в FastReport. Учебное пособие. Рязань: РГРТУ, 2010.

3. Маркин А. В. Разработка отчетов в информационных системах: учеб.пособие. М.: Диалог-МИФИ, 2012.

4. Хомоненко А. Д., Гофман В. Э. Работа с БД в Delphi. 3-е изд., перераб. и доп. СПб.: БХВ-Петербург, 2005.

5. [Электронный ресурс]. URL: http://www.abonentplus.ru

6. Дарахвелидзе П., Марков Е. Программирование в Delphi 7. СПб.: БХВ-Петербург, 2003.

7. Фленов М. Е. Библия Delphi.3-е изд., перераб. и доп. СПб.: БХВ-Петербург, 2011.

8. Архангельский А. Я. Программирование в Delphi для Windows. Версии 2006, 2007, Turbo Delphi. М.: ООО «Бином-Пресс», 2007.

9. FIBPlus – компоненты для Delphi, C++ Builder и Kylix для работы с Firebird и InterBase [Электронный ресурс]. URL: http://www.devrace.com /ru/fibplus/.

10. Компоненты и драйверы [Электронный ресурс]. URL: http://www.ibase.ru/components.htm.

11. Ковязин А. Н., Востриков С. М. Мир InterBase: Архитектура, администрирование и разработка приложений БД в InterBase, Firebird, Yaffil. 3-е изд., доп. СПб.: Кудиц-образ, 2005.

12. Работа с компонентами IBX или использование InterBase eXpress в приложениях на Delphi и C++Builder, с СУБД Firebird и InterBase [Электронный ресурс]. URL: http://www.ibase.ru/devinfo/ibx.htm.

13. Кузьменко Д. Транзакции в InterBase и Firebird [Электронный ресурс]. URL: http://www.ibase.ru/devinfo/ibtrans.htm

14. MySQL+Lazarus: Работа и базой данной на Web сервере из Lazarus [Электронный ресурс]. URL: http://www.freepascal.ru/article/lazarus /20090416150500.

15. Фаронов В. В. Программирование БД в Delphi 7: Учебный курс. СПб.: Питер, 2006.

16. Осипов Д. Л. Базы данных и Delphi: Теория и практика. СПб.: БХВ-Петербург, 2011.

Приложения

А. Описание учебной БД

Учебная БД представляет собой очень упрощенный пример информационной модели расчетно-аналитической компоненты расчетно-платежного комплекса «Абонент+», который используется для информационного обеспечения деятельности организаций по оказанию населению жилищно-коммунальных услуг и расчетам за них [3].

Учебная БД состоит из 8 таблиц: 5 таблиц-справочников и 3 информационных таблиц.

В учебной БД используются такие таблицы-справочники:

- **Street.** Справочник улиц, на которых проживают абоненты. Поля таблицы:
  StreetCD – уникальный код улицы (первичный ключ);
  StreetNM – название улицы, расшифровывающее код улицы.
- **Abonent.** Справочник абонентов. Поля таблицы:
  AccountCD – номер лицевого счета абонента, уникальным образом идентифицирующий каждого из абонентов (первичный ключ);
  StreetCD – код улицы, на которой проживает абонент (внешний ключ, ссылающийся на первичный ключ таблицы Street);
  HouseNo – номер дома, в котором проживает абонент;
  FlatNo – номер квартиры;
  Fio – фамилия, имя и отчество абонента в формате «Фамилия И.О.»;
  Phone – номер телефона.
- **Services.** Справочник услуг, оказываемых абонентам жилищно-коммунальными организациями. Поля таблицы:
  ServiceCD – код услуги (первичный ключ);
  ServiceNM – наименование услуги.
- **Disrepair.** Справочник типовых неисправностей газового оборудования абонентов. Поля таблицы:
  FailureCD – код неисправности газового оборудования (первичный ключ);
  FailureNM – наименование неисправности газового оборудования.
- **Executor.** Справочник исполнителей заявок, поданных абонентами на ремонт газового оборудования. Исполнителями являются работники ремонтной службы газораспределительной организации, оказывающей соответствующие услуги. Поля таблицы:
  ExecutorCD – уникальный код, идентифицирующий исполнителей ремонтных заявок (первичный ключ);
  Fio – фамилия, имя и отчество исполнителя в формате «Фамилия И.О.».

В качестве информационных таблиц учебной БД выделены следующие таблицы:

- **NachislSumma.** Таблица для хранения информации о размерах ежемесячного начисления абонентам за оказанные им услуги (которые расшифровываются в справочнике услуг Services). Оплата за ремонт газового оборудования производится по факту оказания услуг, и начисление за него не производится. Поля таблицы:

  NachislFactCD – уникальный идентификатор факта начисления (первичный ключ);

  AccountCD – номер лицевого счета абонента, которому было сделано начисление (внешний ключ, ссылающийся на первичный ключ таблицы Abonent);

  ServiceCD – код услуги, за которую выполнено начисление (внешний ключ, ссылающийся на первичный ключ таблицы Services);

  NachislSum – значение начисленной суммы;

  NachislMonth – номер месяца, за который произведено начисление;

  NachislYear – год, за месяц которого выполнено начисление.

- **PaySumma.** Таблица для хранения значений оплат, внесенных абонентами за оказанные им услуги. Для каждого факта оплаты по какой-либо услуге указывается дата оплаты, оплачиваемые месяц и год. Таким образом, при сопоставлении информации по конкретному абоненту, хранящейся в таблице NachislSumma, можно узнать размер долга (дебет) или переплаты (кредит) у данного абонента на указанный месяц года. Поля таблицы:

  PayFactCD – уникальный идентификатор факта оплаты по услуге Request (первичный ключ);

  AccountCD – номер лицевого счета абонента, оплатившего оказанную ему услугу (внешний ключ, ссылающийся на первичный ключ таблицы Abonent);

  ServiceCD – код оплаченной услуги (внешний ключ, ссылающийся на первичный ключ таблицы Services);

  PaySum – значение оплаченной суммы;

  PayDate – дата оплаты;

  PayMonth – номер оплачиваемого месяца;

  PayYear – оплачиваемый год.

Request. Таблица для хранения информации о заявках абонентов на ремонт газового оборудования. Каждая ремонтная заявка характеризуется номером лицевого счета заявившего абонента (расшифровка в справочнике Abonent), определенной неисправностью газового оборудования (расшифровка в справочнике Disrepair), исполнителем ремонтной работы (справочник Executor), датой регистрации заявки, датой выполнения ремонта и признаком погашения (1/0). Поля таблицы:

RequestCD – уникальный код ремонтной заявки (первичный ключ);

AccountCD – номер лицевого счета абонента, подавшего данную ремонтную заявку (внешний ключ, ссылающийся на первичный ключ таблицы Abonent);

FailureCD – код неисправности газового оборудования, заявленной абонентом в данной ремонтной заявке (внешний ключ, ссылающийся на первичный ключ таблицы Disrepair);

ExecutorCD – код исполнителя, ответственного за выполнение данной ремонтной заявки (внешний ключ, ссылающийся на первичный ключ таблицы Executor);

IncomingDate – дата поступления заявки;

ExecutionDate – дата выполнения заявки;

Executed – поле логического типа, признак того, погашена заявка или нет.

В общем случае зарегистрированная ремонтная заявка может быть:

- не назначена ни одному из исполнителей, а, следовательно, не выполнена и не погашена;
- назначена одному из исполнителей, но не выполнена им, а, следовательно, и не погашена;
- назначена одному из исполнителей, выполнена им, но не погашена;
- назначена одному из исполнителей, выполнена им и погашена.

Для поддержания правил ссылочной целостности, реализующих запрет удаления записи в родительской таблице при наличии связанных записей в дочерних таблицах, в учебной БД определены следующие триггеры.

Td_Abonent. Триггер запускается после удаления строки в таблице Abonent. Если в таблицах NachislSumma или PaySumma имеются записи с внешним ключом AccountCD, ссылающимся на удаляемую строку таблицы Abo-nent, то триггер вызывает исключение Del_Restrict и операция удаления прерывается.

Td_Services. Триггер запускается после удаления строки в таблице Services. Если в таблицах PaySumma или NachislSumma имеются записи с внешним ключом ServiceCD, ссылающимся на удаляемую строку в таблице Services, то триггер вызывает пользовательское исключение Del_Restrict и прерывает выполнение операции.

Текст определения описанных триггеров приведен в скрипте по созданию учебной БД в приложении Б.

На рис. А.1 представлена модель учебной БД на логическом уровне, а на рис. А.2 – на физическом уровне. Подробно особенности проектирования БД описаны в учебном пособии [1].

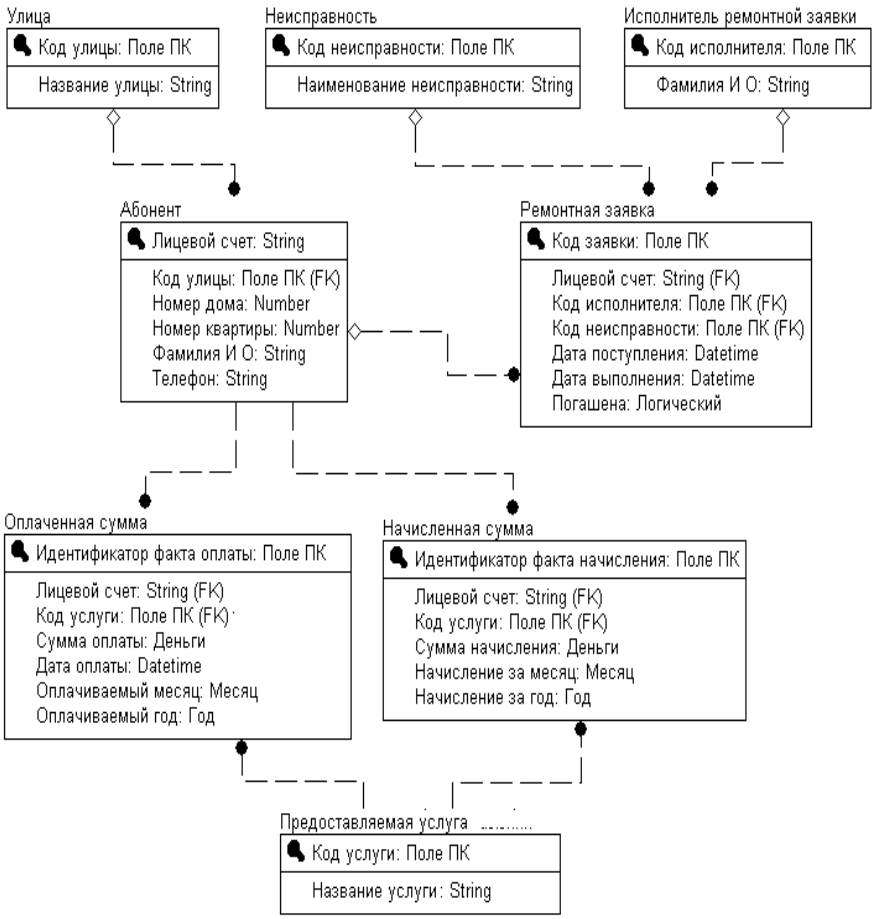


Рис. А.1. *Модель учебной БД на логическом уровне*

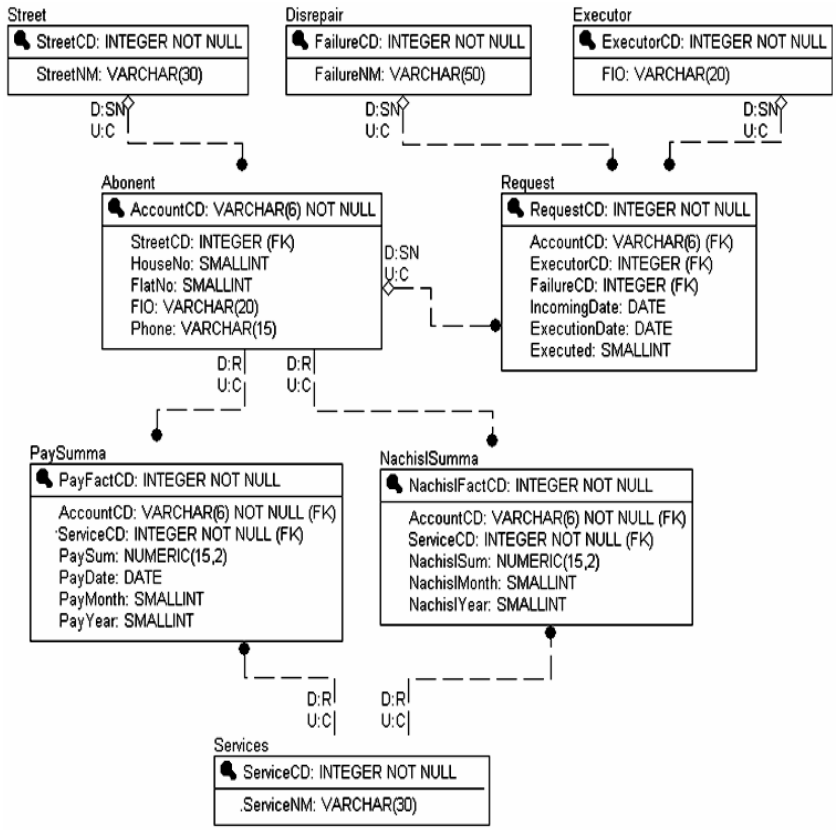


Рис. А.2. *Модель учебной БД на физическом уровне*

В табл. А.1 приводится содержимое всех таблиц учебной БД. Все запросы, приведенные в пособии, выполнены именно над этими данными, и их полное указание поможет проанализировать логику получения результатов.

Таблица А.1. Данные таблицы Street

StreetCD	***StreetNM***
1	ЦИОЛКОВСКОГО УЛИЦА
2	НОВАЯ УЛИЦА
3	ВОЙКОВ ПЕРЕУЛОК
4	ТАТАРСКАЯ УЛИЦА
5	ГАГАРИНА УЛИЦА
6	МОСКОВСКАЯ УЛИЦА
7	КУТУЗОВА УЛИЦА
8	МОСКОВСКОЕ ШОССЕ

Таблица А.2. Данные таблицы Abonent

AccountCD	***StreetCD***	***HouseNo***	***FlatNo***	***Fio***	***Phone***
005488	3	4	1	Аксенов С.А.	556893
015527	3	1	65	Конюхов В.С.	761699
080047	8	39	36	Шубина Т. П.	257842
080270	6	35	6	Тимошкина Н.Г.	321002
080613	8	35	11	Лукашина Р.М.	254417
115705	3	1	82	Мищенко Е.В.	769975
126112	4	7	11	Маркова В.П.	683301
136159	7	39	1	Свирина З.А.	NULL
136160	4	9	15	Шмаков С.В.	NULL
136169	4	7	13	Денисова Е.К.	680305
443069	4	51	55	Стародубцев Е.В.	683014
443690	7	5	1	Тулупова М.И.	214833

Таблица А.3. Данные таблицы Services

ServiceCD	***ServiceNM***
1	Газоснабжение
2	Электроснабжение
3	Теплоснабжение
4	Водоснабжение

Таблица А.4. Данные таблицы Disrepair

FailureCD	***FailureNM***
1	Засорилась водогрейная колонка
2	Не горит АГВ
3	Течет из водогрейной колонки
4	Неисправна печная горелка
5	Неисправен газовый счетчик
6	Плохое поступление газа на горелку плиты
7	Туго поворачивается пробка крана плиты
8	При закрытии краника горелка плиты не гаснет
12	Неизвестна

Таблица А.5. Данные таблицы Executor

ExecutorCD	***Fio***
1	Стародубцев Е.М.
2	Булгаков Т.И.
3	Шубин В.Г.
4	Шлюков М.К.
5	Школьников С.М.

Таблица А.6. Данные таблицы NachislSumma

NachislfactCD	*AccountCD*	*ServiceCD*	*NachislSum*	*NachislMonth*	*NachislYear*
1	136160	2	56	1	2013
2	005488	2	46	12	2010
3	005488	2	56	4	2013
4	115705	2	40	1	2010
5	115705	2	250	9	2011
6	136160	1	18,3	1	2012
7	080047	2	80	10	2012
8	080047	2	80	10	2011
9	080270	2	46	12	2011
10	080613	2	56	6	2011
11	115705	2	250	9	2010
12	115705	2	58,7	8	2011
13	136160	2	20	5	2011
15	136169	2	20	5	2011
16	136169	2	58,7	11	2011
17	443069	2	80	9	2011
18	443069	2	38,5	8	2011
19	005488	2	58,7	12	2011
20	015527	1	28,32	7	2012
21	080047	1	19,56	3	2012
22	080613	1	10,6	9	2012
23	443069	1	38,28	12	2012
24	015527	1	38,32	4	2013
25	115705	1	37,15	10	2013
26	080613	1	12,6	8	2010
27	136169	1	25,32	1	2013
28	080270	1	57,1	2	2012
29	136159	1	8,3	8	2013
30	005488	1	62,13	4	2010
31	115705	1	37,8	5	2011
32	443690	1	17,8	6	2012
33	080047	1	22,56	5	2013
34	126112	1	15,3	8	2010
35	080047	1	32,56	9	2011
36	080613	1	12,6	4	2012
37	115705	1	37,15	11	2013
38	080270	1	58,1	12	2010
39	136169	1	28,32	1	2011
40	015527	1	18,32	2	2012
41	443690	1	21,67	3	2013
42	080613	1	22,86	4	2010
43	080270	1	60,1	5	2011

Окончание табл. А.6

NachislfactCD	***AccountCD***	***ServiceCD***	***NachislSum***	***NachislMonth***	***NachislYear***
44	136169	1	28,32	2	2012
45	080047	1	22,2	7	2013
46	126112	1	25,3	8	2011
47	443069	1	38,32	9	2011
48	136159	1	8,3	10	2012
49	115705	1	37,15	6	2013
50	136160	1	18,3	12	2010
51	005488	3	279,8	5	2012
52	005488	3	266,7	2	2013
53	015527	3	343,36	11	2013
54	080047	3	271,6	2	2013
55	080270	3	278,25	11	2013
56	080613	3	254,4	7	2011
57	080613	3	258,8	2	2013
58	080613	3	239,33	5	2013
59	126112	3	179,9	4	2012
60	136159	3	180,13	9	2013
61	136160	3	238,8	3	2010
62	136160	3	237,38	3	2011
63	136169	3	349,19	6	2012
64	136169	3	346,18	7	2012
65	443690	3	290,33	3	2013
66	015527	4	580,1	7	2012
67	015527	4	611,3	10	2013
68	080270	4	444,34	3	2011
69	080270	4	453,43	6	2012
70	080270	4	454,6	4	2013
71	115705	4	553,85	1	2012
72	126112	4	435,5	6	2012
73	136159	4	349,38	4	2011
74	136159	4	418,88	6	2012
75	136169	4	528,44	10	2013
76	443069	4	466,69	5	2012
77	443069	4	444,45	10	2013
78	443690	4	480,88	8	2011
79	443690	4	500,13	9	2012

Таблица А.7. Данные таблицы PaySumma

PayFactCD	*AccountCD*	*ServiceCD*	*PaySum*	*PayDate*	*PayMonth*	*PayYear*
1	005488	2	58,7	08.01.2012	12	2011
2	005488	2	40	06.01.2011	12	2010
3	005488	2	56	06.05.2013	4	2013
4	115705	2	40	10.02.2010	1	2010
5	115705	2	250	03.10.2011	9	2011
6	136160	2	20	13.06.2011	5	2011
7	136160	2	56	12.02.2013	1	2013
8	136169	2	20	22.06.2011	5	2011
9	080047	2	80	26.11.2012	10	2012
10	080047	2	80	21.11.2011	10	2011
11	080270	2	30	03.01.2012	12	2011
12	080613	2	58,5	19.07.2011	6	2011
13	115705	2	250	06.10.2010	9	2010
14	115705	2	58,7	04.09.2011	8	2011
15	136169	2	58,7	01.12.2011	11	2011
16	443069	2	80	03.10.2011	9	2011
17	443069	2	38,5	13.09.2011	8	2011
18	136160	1	18	05.02.2012	1	2012
19	015527	1	30	03.08.2012	7	2012
20	080047	1	19,56	02.04.2012	3	2012
21	080613	1	11	03.10.2012	9	2012
22	443069	1	38,28	04.02.2013	12	2012
23	015527	1	40	07.05.2013	4	2013
24	115705	1	37,15	04.11.2013	10	2013
25	080613	1	12	20.09.2010	8	2010
26	136169	1	25,32	03.02.2013	1	2013
27	080270	1	60	05.03.2012	2	2012
28	136159	1	8,3	10.09.2013	8	2013
29	005488	1	65	03.05.2010	4	2010
30	115705	1	37,8	12.07.2011	5	2011
31	443690	1	20	10.07.2012	6	2012
32	080047	1	22,56	25.06.2013	5	2013
33	126112	1	15,3	08.09.2010	8	2010
34	080047	1	32,56	18.10.2011	9	2011

Продолжение табл. А.7

PayFactCD	***AccountCD***	***ServiceCD***	***PaySum***	***PayDate***	***PayMonth***	***PayYear***
35	080613	1	12,6	22.05.2012	4	2012
36	115705	1	37,15	23.12.2013	11	2013
37	080270	1	58,1	07.01.2011	12	2010
38	136169	1	28,32	08.02.2011	1	2011
39	015527	1	20	18.03.2012	2	2012
40	443690	1	19,47	10.04.2013	3	2013
41	080613	1	22,86	04.05.2010	4	2010
42	080270	1	60	07.06.2011	5	2011
43	136169	1	28,32	05.03.2012	2	2012
44	080047	1	22,2	10.08.2013	7	2013
45	126112	1	25,3	10.09.2011	8	2011
46	443069	1	38,32	09.10.2011	9	2011
47	136159	1	8,3	14.11.2012	10	2012
48	115705	1	37,15	10.08.2013	6	2013
49	136160	1	16	07.01.2011	12	2010
50	005488	3	280	10.06.2012	5	2012
51	005488	3	260	11.03.2013	2	2013
52	015527	3	345	15.12.2013	11	2013
53	080047	3	271,6	12.03.2013	2	2013
54	080270	3	278	06.12.2013	11	2013
55	080613	3	254,4	10.08.2011	7	2011
56	080613	3	258,8	08.03.2013	2	2013
57	080613	3	239,35	11.06.2013	5	2013
58	126112	3	179,9	01.05.2012	4	2012
59	136159	3	180,13	21.10.2013	9	2013
60	136160	3	240	04.04.2010	3	2010
61	136160	3	200	06.04.2011	3	2011
62	136169	3	349,19	14.07.2012	6	2012
63	136169	3	346,18	13.08.2012	7	2012
64	443690	3	295	09.04.2013	3	2013
65	015527	4	580,1	08.08.2012	7	2012
66	015527	4	611,3	03.11.2013	10	2013
67	080270	4	444,5	18.04.2011	3	2011
68	080270	4	450	14.07.2012	6	2012
69	080270	4	460	12.05.2013	4	2013

Окончание табл. А.7

PayFactCD	***AccountCD***	***ServiceCD***	***PaySum***	***PayDate***	***PayMonth***	***PayYear***
70	115705	4	553,85	02.02.2012	1	2012
71	126112	4	435,5	12.07.2012	6	2012
72	136159	4	349,38	18.05.2011	4	2011
73	136159	4	420	09.07.2012	6	2012
74	136169	4	528,44	26.11.2013	10	2013
75	443069	4	466,69	03.06.2012	5	2012
76	443069	4	444,45	16.11.2013	10	2013
77	443690	4	485	05.09.2011	8	2011

Таблица А.8. Данные таблицы Request

RequestCD	***AccountCD***	***ExecutorCD***	***FailureCD***	***IncomingDate***	***ExecutionDate***	***Executed***
1	005488	1	1	17.12.2011	20.12.2011	1
2	115705	3	1	07.08.2011	12.08.2011	1
3	015527	1	12	28.02.2012	08.03.2012	0
5	080270	4	1	31.12.2011	NULL	0
6	080613	1	6	16.06.2011	24.06.2011	1
7	080047	3	2	20.10.2012	24.10.2012	1
9	136169	2	1	06.11.2011	08.11.2011	1
10	136159	3	12	01.04.2011	03.04.2011	0
11	136160	1	6	12.01.2013	12.01.2013	1
12	443069	5	2	08.08.2011	10.08.2011	1
13	005488	5	8	04.09.2010	05.12.2010	1
14	005488	4	6	04.04.2013	13.04.2013	1
15	115705	4	5	20.09.2010	23.09.2010	1
16	115705	NULL	3	28.12.2011	NULL	0
17	115705	1	5	15.08.2011	06.09.2011	1
18	115705	2	3	28.12.2012	04.01.2013	1
19	080270	4	8	17.12.2011	27.12.2011	1
20	080047	3	2	11.10.2011	11.10.2011	1
21	443069	1	2	13.09.2011	14.09.2011	1
22	136160	1	7	18.05.2011	25.05.2011	1
23	136169	5	7	07.05.2011	08.05.2011	1

Б. Скрипт для создания учебной БД

```
SET SQL DIALECT 3;
CREATE DATABASE 'C:\SQLLAB.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096 DEFAULT CHARACTER SET WIN1251;
/******************************************************************/
/*                          Domains                               */
/******************************************************************/
CREATE DOMAIN BOOL AS SMALLINT CHECK (VALUE IN (0, 1));
CREATE DOMAIN MONEY AS NUMERIC(15,2);
CREATE DOMAIN TMONTH AS SMALLINT
CHECK (VALUE BETWEEN 1 AND 12);
CREATE DOMAIN PKFIELD AS INTEGER;
CREATE DOMAIN TYEAR AS SMALLINT
CHECK (VALUE BETWEEN 1990 AND 2100);
/******************************************************************/
/*                          Exceptions                            */
/******************************************************************/
CREATE EXCEPTION INS_RESTRICT
'Ограничение добавления записи в дочернюю таблицу';
CREATE EXCEPTION DEL_RESTRICT
'Ограничение удаления записи из родительской таблицы';
CREATE EXCEPTION UPD_RESTRICT
'Ограничение модификации записи в родительской таблице';
/******************************************************************/
/*                          Tables                                */
/******************************************************************/
CREATE TABLE STREET (
   STREETCD  PKFIELD NOT NULL PRIMARY KEY,
   STREETNM  VARCHAR(30) );
CREATE TABLE SERVICES (
   SERVICECD  PKFIELD NOT NULL PRIMARY KEY,
   SERVICENM  VARCHAR(30) );
CREATE TABLE DISREPAIR (
   FAILURECD  PKFIELD NOT NULL PRIMARY KEY,
   FAILURENM  VARCHAR(50) );
CREATE TABLE EXECUTOR (
   EXECUTORCD  PKFIELD NOT NULL PRIMARY KEY,
   Fio  VARCHAR(20) );
CREATE TABLE ABONENT (
   ACCOUNTCD  VARCHAR(6) NOT NULL PRIMARY KEY,
   STREETCD  PKFIELD REFERENCES STREET
     ON DELETE SET NULL ON UPDATE CASCADE,
   HOUSENO  SMALLINT,
   FLATNO  SMALLINT,
```

```
   Fio  VARCHAR(20),
   PHONE  VARCHAR(15));
CREATE TABLE NACHISLSUMMA (
   NACHISLFACTCD  PKFIELD NOT NULL PRIMARY KEY,
   ACCOUNTCD VARCHAR(6) NOT NULL REFERENCES ABONENT
    ON UPDATE CASCADE,
   SERVICECD PKFIELD NOT NULL REFERENCES SERVICES
    ON UPDATE CASCADE,
   NACHISLSUM     MONEY,
   NACHISLMONTH   TMONTH,
   NACHISLYEAR    TYEAR);
CREATE TABLE PAYSUMMA (
   PAYFACTCD  PKFIELD NOT NULL PRIMARY KEY,
   ACCOUNTCD  VARCHAR(6) NOT NULL REFERENCES ABONENT
    ON UPDATE CASCADE,
   SERVICECD PKFIELD NOT NULL REFERENCES SERVICES
    ON UPDATE CASCADE,
   PAYSUM  MONEY,
   PAYDATE  DATE,
   PAYMONTH  TMONTH,
   PAYYEAR  TYEAR);
CREATE TABLE REQUEST (
   REQUESTCD PKFIELD NOT NULL PRIMARY KEY,
   ACCOUNTCD  VARCHAR(6) REFERENCES ABONENT
    ON DELETE SET NULL ON UPDATE CASCADE,
   EXECUTORCD  PKFIELD REFERENCES EXECUTOR
    ON DELETE SET NULL ON UPDATE CASCADE,
   FAILURECD  PKFIELD REFERENCES DISREPAIR
    ON DELETE SET NULL ON UPDATE CASCADE,
   INCOMINGDATE  DATE,
   EXECUTIONDATE  DATE,
   EXECUTED  BOOL);
/*****************************************************************/
/*                        Insert  STREET                         */
/*****************************************************************/
INSERT INTO STREET (STREETCD, STREETNM) VALUES (3,
'ВОЙКОВ ПЕРЕУЛОК');
INSERT INTO STREET (STREETCD, STREETNM) VALUES (7,
'КУТУЗОВА УЛИЦА');
INSERT INTO STREET (STREETCD, STREETNM) VALUES (6,
'МОСКОВСКАЯ УЛИЦА');
INSERT INTO STREET (STREETCD, STREETNM) VALUES (8,
'МОСКОВСКОЕ ШОССЕ');
INSERT INTO STREET (STREETCD, STREETNM) VALUES (4,
'ТАТАРСКАЯ УЛИЦА');
INSERT INTO STREET (STREETCD, STREETNM) VALUES (5,
'ГАГАРИНА УЛИЦА');
```

```
INSERT INTO STREET (STREETCD, STREETNM) VALUES (1,
'ЦИОЛКОВСКОГО УЛИЦА');
INSERT INTO STREET (STREETCD, STREETNM) VALUES (2, 'НОВАЯ УЛИЦА');
COMMIT WORK;
/*************************************************************/
/*                    Insert  SERVICES                       */
/*************************************************************/
INSERT INTO SERVICES (SERVICECD, SERVICENM) VALUES (1,
'Газоснабжение');
INSERT INTO SERVICES (SERVICECD, SERVICENM) VALUES (2,
'Электроснабжение');
INSERT INTO SERVICES (SERVICECD, SERVICENM) VALUES (3,
'Теплоснабжение');
INSERT INTO SERVICES (SERVICECD, SERVICENM) VALUES (4,
'Водоснабжение');
COMMIT WORK;
/*************************************************************/
/*                    Insert  DISREPAIR                      */
/*************************************************************/
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (1,
'Засорилась водогрейная колонка');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (2,
'Не горит АГВ');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (3,
'Течет из водогрейной колонки');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (4,
'Неисправна печная горелка');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (5,
'Неисправен газовый счетчик');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (6,
'Плохое поступление газа на горелку плиты');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (7,
'Туго поворачивается пробка крана плиты');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (8,
'При закрытии краника горелка плиты не гаснет');
INSERT INTO DISREPAIR (FAILURECD, FAILURENM) VALUES (12,
'Неизвестна');
COMMIT WORK;
/*************************************************************/
/*                    Insert  EXECUTOR                       */
/*************************************************************/
INSERT INTO EXECUTOR (EXECUTORCD, Fio) VALUES (1,
'Стародубцев Е.М.');
INSERT INTO EXECUTOR (EXECUTORCD, Fio) VALUES (2, 'Булгаков Т.И.');
INSERT INTO EXECUTOR (EXECUTORCD, Fio) VALUES (3, 'Шубин В.Г.');
INSERT INTO EXECUTOR (EXECUTORCD, Fio) VALUES (4, 'Шлюков М.К.');
INSERT INTO EXECUTOR (EXECUTORCD, Fio) VALUES (5, 'Школьников С.М.');
```

```
COMMIT WORK;
/*****************************************************************/
/*                        Insert  ABONENT                        */
/*****************************************************************/
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('005488', 3, 4, 1, 'Аксенов С.А.', '556893');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('115705', 3, 1, 82, 'Мищенко Е.В.', '769975');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('015527', 3, 1, 65, 'Конюхов В.С.', '761699');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('443690', 7, 5, 1, 'Тулупова М.И.', '214833');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('136159', 7, 39, 1, 'Свирина З.А.', NULL);
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('443069', 4, 51, 55, 'Стародубцев Е.В.', '683014');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('136160', 4, 9, 15, 'Шмаков С.В.', NULL);
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('126112', 4, 7, 11, 'Маркова В.П.', '683301');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('136169', 4, 7, 13, 'Денисова Е.К.', '680305');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('080613', 8, 35, 11, 'Лукашина Р.М.', '254417');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('080047', 8, 39, 36, 'Шубина Т. П.', '257842');
INSERT INTO ABONENT (ACCOUNTCD, STREETCD, HOUSENO, FLATNO, Fio,
PHONE) VALUES ('080270', 6, 35, 6, 'Тимошкина Н.Г.', '321002');
COMMIT WORK;
/*****************************************************************/
/*                        Insert  NACHISLSUMMA                   */
/*****************************************************************/
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (19,
'005488', 2, 58.7, 12, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (2,
'005488', 2, 46, 12, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (3,
'005488', 2, 56, 4, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (4,
'115705', 2, 40, 1, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (5,
'115705', 2, 250, 9, 2011);
```

```
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (13,
'136160', 2, 20, 5, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (1,
'136160', 2, 56, 1, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (15,
'136169', 2, 20, 5, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (7,
'080047', 2, 80, 10, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (8,
'080047', 2, 80, 10, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (9,
'080270', 2, 46, 12, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (10,
'080613', 2, 56, 6, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (11,
'115705', 2, 250, 9, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (12,
'115705', 2, 58.7, 8, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (16,
'136169', 2, 58.7, 11, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (17,
'443069', 2, 80, 9, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (18,
'443069', 2, 38.5, 8, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (6,
'136160', 1, 18.3, 1, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (20,
'015527', 1, 28.32, 7, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (21,
'080047', 1, 19.56, 3, 2012);
```

```
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (22,
'080613', 1, 10.6, 9, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (23,
'443069', 1, 38.28, 12, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (24,
'015527', 1, 38.32, 4, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (25,
'115705', 1, 37.15, 10, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (26,
'080613', 1, 12.6, 8, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (27,
'136169', 1, 25.32, 1, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (28,
'080270', 1, 57.1, 2, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (29,
'136159', 1, 8.3, 8, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (30,
'005488', 1, 62.13, 4, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (31,
'115705', 1, 37.8, 5, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (32,
'443690', 1, 17.8, 6, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (33,
'080047', 1, 22.56, 5, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (34,
'126112', 1, 15.3, 8, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (35,
'080047', 1, 32.56, 9, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (36,
'080613', 1, 12.6, 4, 2012);
```

```
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (37,
'115705', 1, 37.15, 11, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (38,
'080270', 1, 58.1, 12, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (39,
'136169', 1, 28.32, 1, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (40,
'015527', 1, 18.32, 2, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (41,
'443690', 1, 21.67, 3, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (42,
'080613', 1, 22.86, 4, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (43,
'080270', 1, 60.1, 5, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (44,
'136169', 1, 28.32, 2, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (45,
'080047', 1, 22.2, 7, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (46,
'126112', 1, 25.3, 8, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (47,
'443069', 1, 38.32, 9, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (48,
'136159', 1, 8.3, 10, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (49,
'115705', 1, 37.15, 6, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (50,
'136160', 1, 18.3, 12, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (51,
'005488', 3, 279.8, 5, 2012);
```

```
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (52,
'005488', 3, 266.7, 2, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (53,
'015527', 3, 343.36, 11, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (54,
'080047', 3, 271.6, 2, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (55,
'080270', 3, 278.25, 11, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (56,
'080613', 3, 254.4, 7, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (57,
'080613', 3, 258.8, 2, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (58,
'080613', 3, 239.33, 5, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (59,
'126112', 3, 179.9, 4, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (60,
'136159', 3, 180.13, 9, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (61,
'136160', 3, 238.8, 3, 2010);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (62,
'136160', 3, 237.38, 3, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (63,
'136169', 3, 349.19, 6, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (64,
'136169', 3, 346.18, 7, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (65,
'443690', 3, 290.33, 3, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (66,
'015527', 4, 580.1, 7, 2012);
```

```
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (67,
'015527', 4, 611.3, 10, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (68,
'080270', 4, 444.34, 3, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (69,
'080270', 4, 453.43, 6, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (70,
'080270', 4, 454.6, 4, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (71,
'115705', 4, 553.85, 1, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (72,
'126112', 4, 435.5, 6, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (73,
'136159', 4, 349.38, 4, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (74,
'136159', 4, 418.88, 6, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (75,
'136169', 4, 528.44, 10, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (76,
'443069', 4, 466.69, 5, 2012);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (77,
'443069', 4, 444.45, 10, 2013);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (78,
'443690', 4, 480.88, 8, 2011);
INSERT INTO NACHISLSUMMA (NACHISLFACTCD, ACCOUNTCD,
SERVICECD, NACHISLSUM, NACHISLMONTH, NACHISLYEAR) VALUES (79,
'443690', 4, 500.13, 9, 2012);

COMMIT WORK;
/*****************************************************************/
/*                        Insert  PAYSUMMA                       */
/*****************************************************************/
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (1, '005488', 2, 58.7,
'2012-01-08', 12, 2011);
```

```
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (2, '005488', 2, 40,
'2011-01-06', 12, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (3, '005488', 2, 56,
'2013-05-06', 4, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (4, '115705', 2, 40,
'2010-02-10', 1, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (5, '115705', 2, 250,
'2011-10-03', 9, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (6, '136160', 2, 20,
'2011-06-13', 5, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (7, '136160', 2, 56,
'2013-02-12', 1, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (8, '136169', 2, 20,
'2011-06-22', 5, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (9, '080047', 2, 80,
'2012-11-26', 10, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (10, '080047', 2, 80,
'2011-11-21', 10, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (11, '080270', 2, 30,
'2012-01-03', 12, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (12, '080613', 2, 58.5,
'2011-07-19', 6, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (13, '115705', 2, 250,
'2010-10-06', 9, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (14, '115705', 2, 58.7,
'2011-09-04', 8, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (15, '136169', 2, 58.7,
'2011-12-01', 11, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (16, '443069', 2, 80,
'2011-10-03', 9, 2011);
```

```
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (17, '443069', 2, 38.5,
'2011-09-13', 8, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (18, '136160', 1, 18,
'2012-02-05', 1, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (19, '015527', 1, 30,
'2012-08-03', 7, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (20, '080047', 1, 19.56,
'2012-04-02', 3, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (21, '080613', 1, 11,
'2012-10-03', 9, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (22, '443069', 1, 38.28,
'2013-02-04', 12, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (23, '015527', 1, 40,
'2013-05-07', 4, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (24, '115705', 1, 37.15,
'2013-11-04', 10, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (25, '080613', 1, 12,
'2010-09-20', 8, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (26, '136169', 1, 25.32,
'2013-02-03', 1, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (27, '080270', 1, 60,
'2012-03-05', 2, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (28, '136159', 1, 8.3,
'2013-09-10', 8, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (29, '005488', 1, 65,
'2010-05-03', 4, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (30, '115705', 1, 37.8,
'2011-07-12', 5, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (31, '443690', 1, 20,
'2012-07-10', 6, 2012);
```

```
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (32, '080047', 1, 22.56,
'2013-06-25', 5, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (33, '126112', 1, 15.3,
'2010-09-08', 8, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (34, '080047', 1, 32.56,
'2011-10-18', 9, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (35, '080613', 1, 12.6,
'2012-05-22', 4, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (36, '115705', 1, 37.15,
'2013-12-23', 11, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (37, '080270', 1, 58.1,
'2011-01-07', 12, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (38, '136169', 1, 28.32,
'2011-02-08', 1, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (39, '015527', 1, 20,
'2012-03-18', 2, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (40, '443690', 1, 19.47,
'2013-04-10', 3, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (41, '080613', 1, 22.86,
'2010-05-04', 4, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (42, '080270', 1, 60,
'2011-06-07', 5, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (43, '136169', 1, 28.32,
'2012-03-05', 2, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (44, '080047', 1, 22.2,
'2013-08-10', 7, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (45, '126112', 1, 25.3,
'2011-09-10', 8, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (46, '443069', 1, 38.32,
'2011-10-09', 9, 2011);
```

```
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (47, '136159', 1, 8.3,
'2012-11-14', 10, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (48, '115705', 1, 37.15,
'2013-08-10', 6, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (49, '136160', 1, 16,
'2011-01-07', 12, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (50, '005488', 3, 280,
'2012-06-10', 5, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (51, '005488', 3, 260,
'2013-03-11', 2, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (52, '015527', 3, 345,
'2013-12-15', 11, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (53, '080047', 3, 271.6,
'2013-03-12', 2, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (54, '080270', 3, 278,
'2013-12-06', 11, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (55, '080613', 3, 254.4,
'2011-08-10', 7, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (56, '080613', 3, 258.8,
'2013-03-8', 2, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (57, '080613', 3, 239.35,
'2013-06-11', 5, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (58, '126112', 3, 179.9,
'2012-05-01', 4, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (59, '136159', 3, 180.13,
'2013-10-21', 9, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (60, '136160', 3, 240,
'2010-04-04', 3, 2010);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (61, '136160', 3, 200,
'2011-04-06', 3, 2011);
```

```
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (62, '136169', 3, 349.19,
'2012-07-14', 6, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (63, '136169', 3, 346.18,
'2012-08-13', 7, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (64, '443690', 3, 295,
'2013-04-09', 3, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (65, '015527', 4, 580.1,
'2012-08-08', 7, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (66, '015527', 4, 611.3,
'2013-11-03', 10, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (67, '080270', 4, 444.5,
'2011-04-18', 3, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (68, '080270', 4, 450,
'2012-07-14', 6, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (69, '080270', 4, 460,
'2013-05-12', 4, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (70, '115705', 4, 553.85,
'2012-02-02', 1, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (71, '126112', 4, 435.5,
'2012-07-12', 6, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (72, '136159', 4, 349.38,
'2011-05-18', 4, 2011);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (73, '136159', 4, 420,
'2012-07-09', 6, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (74, '136169', 4, 528.44,
'2013-11-26', 10, 2013);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (75, '443069', 4, 466.69,
'2012-06-03', 5, 2012);
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (76, '443069', 4, 444.45,
'2013-11-16', 10, 2013);
```

```
INSERT INTO PAYSUMMA (PAYFACTCD, ACCOUNTCD, SERVICECD,
PAYSUM, PAYDATE, PAYMONTH, PAYYEAR) VALUES (77, '443690', 4, 485,
'2011-09-05', 8, 2011);

COMMIT WORK;
/*****************************************************************/
/*                        Insert  REQUEST                        */
/*****************************************************************/
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (1,
'005488', 1, 1, '2011-12-17', '2011-12-20', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (2,
'115705', 3, 1, '2011-08-07', '2011-08-12', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (3,
'015527', 1, 12, '2012-02-28', '2012-03-08', 0);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (5,
'080270', 4, 1, '2011-12-31', NULL, 0);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (6,
'080613', 1, 6, '2011-06-16', '2011-06-24', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (7,
'080047', 3, 2, '2012-10-20', '2012-10-24', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (9,
'136169', 2, 1, '2011-11-06', '2011-11-08', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (10,
'136159', 3, 12, '2011-04-01', '2011-04-03', 0);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (11,
'136160', 1, 6, '2013-01-12', '2013-01-12', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (12,
'443069', 5, 2, '2011-08-08', '2011-08-10', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (13,
'005488', 5, 8, '2010-09-04', '2010-12-05', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (14,
'005488', 4, 6, '2013-04-04', '2013-04-13', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (15,
'115705', 4, 5, '2010-09-20', '2010-09-23', 1);
```

```
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (16,
'115705', NULL, 3, '2011-12-28', NULL, 0);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (17,
'115705', 1, 5, '2011-08-15', '2011-09-06', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (18,
'115705', 2, 3, '2012-12-28', '2013-01-04', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (19,
'080270', 4, 8, '2011-12-17', '2011-12-27', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (20,
'080047', 3, 2, '2011-10-11', '2011-10-11', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (21,
'443069', 1, 2, '2011-09-13', '2011-09-14', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (22,
'136160', 1, 7, '2011-05-18', '2011-05-25', 1);
INSERT INTO REQUEST (REQUESTCD, ACCOUNTCD, EXECUTORCD,
FAILURECD, INCOMINGDATE, EXECUTIONDATE, EXECUTED) VALUES (23,
'136169', 5, 7, '2011-05-07', '2011-05-08', 1);
COMMIT WORK;
/*****************************************************************/
/*                          Triggers                             */
/*****************************************************************/
SET TERM ^ ;
CREATE TRIGGER TD_ABONENT FOR ABONENT
ACTIVE AFTER DELETE POSITION 0
AS
DECLARE VARIABLE NUMROWS INTEGER;
BEGIN
   SELECT COUNT(*)
     FROM NACHISLSUMMA
     WHERE
       NACHISLSUMMA.ACCOUNTCD = OLD.ACCOUNTCD INTO NUMROWS;
   IF (NUMROWS > 0) THEN
   BEGIN
     EXCEPTION Del_Restrict;
   END
   SELECT COUNT(*)
     FROM PAYSUMMA
     WHERE
       PAYSUMMA.ACCOUNTCD = OLD.ACCOUNTCD INTO NUMROWS;
   IF (NUMROWS > 0) THEN
```

```
    BEGIN
      EXCEPTION Del_Restrict;
    END
END
^
CREATE TRIGGER TD_SERVICES FOR SERVICES
ACTIVE AFTER DELETE POSITION 0
AS
DECLARE VARIABLE NUMROWS INTEGER;
BEGIN
    SELECT COUNT(*)
      FROM NACHISLSUMMA
      WHERE
        NACHISLSUMMA.SERVICECD = OLD.SERVICECD INTO NUMROWS;
    IF (NUMROWS > 0) THEN
    BEGIN
      EXCEPTION Del_Restrict;
    END
    SELECT COUNT(*)
      FROM PAYSUMMA
      WHERE
        PAYSUMMA.SERVICECD = OLD.SERVICECD INTO NUMROWS;
    IF (NUMROWS > 0) THEN
    BEGIN
      EXCEPTION Del_Restrict;
    END
END
^
SET TERM ; ^
```

В. Коды хранимых процедур

Текст процедуры **Abonent_Balance**

```
CREATE OR ALTER PROCEDURE Abonent_Balance
  (AccountCD VARCHAR(6), GetMonth INTEGER, GetYear INTEGER)
RETURNS
  (FIO VARCHAR(20), Address VARCHAR(50),
  Phone VARCHAR(15), BegSaldo NUMERIC(15,4),
  Nachisleno NUMERIC(15,4), Oplacheno NUMERIC(15,4),
  EndSaldo NUMERIC(15,4))
AS
DECLARE VARIABLE BegNachisleno NUMERIC(15,4);
DECLARE VARIABLE BegOplacheno NUMERIC(15,4);
BEGIN
  IF (AccountCD IS NULL) THEN
  BEGIN
```

```
  SUSPEND;
  EXIT;
END
/*информация об абоненте*/
SELECT
  A.FIO,'ул.'||(SELECT S.StreetNM FROM Street S WHERE
  S.StreetCD = A.StreetCD)||', д.'||A.HouseNO||', кв.'||A.FlatNO,
  A.Phone
FROM Abonent A
WHERE A.AccountCD = :AccountCD
INTO :FIO, :Address, :Phone;
/*получение начального сальдо*/
SELECT SUM(N.NachislSum)
FROM NachislSumma N
WHERE
  N.AccountCD = :AccountCD AND
  ((N.NachislMonth + N.NachislYear*12) < (:GetMonth + :GetYear*12))
INTO :BegNachisleno;
IF (BegNachisleno IS NULL) THEN BegNachisleno = 0;
SELECT SUM(P.PaySum)
FROM PaySumma P
where
  p.accountCD = :AccountCD and
  ((p.payMonth + P.PayYear*12) < (:GetMonth + :GetYear*12))
into :begOplacheno;
IF (BegOplacheno is null) then begOplacheno = 0;
BegSaldo = BegNachisleno – BegOplacheno;
/*получение суммы начислений*/
select sum(n.nachislSum)
from nachislSumma N
where
  n.accountCD = :AccountCD and
  n.nachislMonth = :GetMonth and
  n.nachislYear = :GetYear
into :nachisleno;
IF (Nachisleno is null) then nachisleno = 0;
/*получение суммы оплат*/
select sum(p.paySum)
from paySumma P
where
  p.accountCD = :AccountCD and
  p.payMonth = :GetMonth and
  p.payYear = :GetYear
into :oplacheno;
IF (Oplacheno is null) then oplacheno = 0;
/*получение конечного сальдо*/
EndSaldo = BegSaldo + Nachisleno – Oplacheno;
```

```
  suspend;
end.
```

Текст процедуры **Abonent_Balance_Period**

```
CREATE OR ALTER PROCEDURE Abonent_Balance_Period
  (AccountCD VARCHAR(6), BegMonth INTEGER, BegYear INTEGER,
  EndMonth INTEGER, EndYear INTEGER)
RETURNS
  (FIO VARCHAR(20), Address VARCHAR(50),
  Phone VARCHAR(15), Cur_month INTEGER, Cur_year INTEGER,
  BegSaldo NUMERIC(15,4), Nachisleno NUMERIC(15,4),
  Oplacheno NUMERIC(15,4), EndSaldo NUMERIC(15,4))
AS
BEGIN
  SELECT
    FIO, Address, Phone
  FROM Abonent_Balance(:AccountCD, :BegMonth, :BegYear)
  INTO :FIO, :Address, :Phone;
  WHILE ((BegMonth + 12*BegYear) <= (EndMonth + 12*EndYear)) DO
  BEGIN
    Cur_year = BegYear;
    Cur_month = BegMonth;
    FOR SELECT
      BegSaldo, Nachisleno, Oplacheno, EndSaldo
    FROM Abonent_Balance(:AccountCD, :BegMonth, :BegYear)
    INTO :BegSaldo, :Nachisleno, :Oplacheno, :EndSaldo
    DO SUSPEND;
    BegMonth = BegMonth + 1;
    IF (BegMonth > 12) THEN
    BEGIN
      BegYear = BegYear + 1;
      BegMonth = 1;
    END
  END
END
```

Г. Код демонстрационного приложения

Код модуля проекта **Abonents**

```
program Abonents;

uses
  Forms,
  AbonentMainForm in 'AbonentMainForm.pas' {frmAbonentMain},
  AbonentDataModule in 'AbonentDataModule.pas' {DataModule1: TDataModule},
  ConnectForm in 'ConnectForm.pas' {frmConnect},
```

```
  Saldo in 'Saldo.pas' {frmSaldo},
  Pay in 'Pay.pas' {frmPay};

{$R *.res}

begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TDataModule1, DataModule1);
  Application.CreateForm(TfrmAbonentMain, frmAbonentMain);
  Application.Run;
end.
```

Код модуля данных **AbonentDataModule**

```
unit AbonentDataModule;

interface

uses
  Windows, SysUtils, Classes, IBDatabase, DB, IBCustomDataSet,
  Controls, Dialogs, Forms, IBTable, IBQuery;

type
  TDataModule1 = class(TDataModule)
    idbAbonents: TIBDatabase;
    itaAbonents: TIBTransaction;
    dsAbonent: TDataSource;
    idAbonent: TIBDataSet;
    idAbonentACCOUNTCD: TIBStringField;
    idAbonentHOUSENO: TSmallintField;
    idAbonentFLATNO: TSmallintField;
    idAbonentFIO: TIBStringField;
    idAbonentPHONE: TIBStringField;
    itStreet: TIBTable;
    dsStreet: TDataSource;
    idAbonentStreetNM: TStringField;
    idAbonentSTREETCD: TIntegerField;
    idSaldo: TIBDataSet;
    dsSaldo: TDataSource;
    idPaySum: TIBDataSet;
    dsPaysum: TDataSource;
    iqMinMonth: TIBQuery;
    itServices: TIBTable;
    dsServices: TDataSource;
    itaPaySum: TIBTransaction;
    itaSaldo: TIBTransaction;
    procedure DataModuleCreate(Sender: TObject);
    procedure DataModuleDestroy(Sender: TObject);
    procedure idbAbonentsLogin(Database: TIBDatabase; LoginParams: TStrings);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  DataModule1: TDataModule1;

implementation

uses ConnectForm;

{$R *.dfm}

procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  try
    idbAbonents.Connected:=True;
  except
    Application.MessageBox('Невозможно подключиться к базе!',
'Приложение закрывается', MB_ICONSTOP);
    Application.Terminate;
  end;
  itaAbonents.Active:=True;
  idAbonent.Open;
  itStreet.Open;
  itServices.Open;
end;

procedure TDataModule1.DataModuleDestroy(Sender: TObject);
begin
  itaAbonents.Active:=False;
  idbAbonents.Connected:=False;
  idAbonent.Close;
  itStreet.Close;
  itServices.Close;
end;

procedure TDataModule1.idbAbonentsLogin(Database: TIBDatabase;
  LoginParams: TStrings);
var
  dlg: TfrmConnect; // созданный диалог
  res : TModalResult;
begin
  dlg:=TfrmConnect.Create(Application);
  res := dlg.ShowModal;
  if res=mrOK then
  begin
    with LoginParams do
```

```
    begin
      Values['USER_NAME'] := dlg.User_Name.text;
      Values['PASSWORD'] := dlg.User_Pass.Text;
    end;
  end; //mrOK
  dlg.Free;
end;

end.
```

Код модуля формы авторизации **ConnectForm**

```
unit ConnectForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TfrmConnect = class(TForm)
    Label1: TLabel;
    User_Name: TEdit;
    Label2: TLabel;
    User_Pass: TEdit;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmConnect: TfrmConnect;

implementation

{$R *.dfm}

procedure TfrmConnect.Button1Click(Sender: TObject);
begin
  Close;
  ModalResult:=mrOK;
end;

procedure TfrmConnect.Button2Click(Sender: TObject);
begin
  Close;
```

```
  ModalResult:=mrCancel;
  Application.Terminate;
end;

end.
```

Код модуля главной формы **AbonentMainForm**

```
unit AbonentMainForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Grids, DBGrids, StdCtrls, ExtCtrls, DBCtrls;

type
  TfrmAbonentMain = class(TForm)
    DBGrid1: TDBGrid;
    RadioGroup1: TRadioGroup;
    eAccountCD: TEdit;
    bLocate: TButton;
    DBNavigator1: TDBNavigator;
    bSaldo: TButton;
    procedure RadioGroup1Click(Sender: TObject);
    procedure bLocateClick(Sender: TObject);
    procedure bSaldoClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmAbonentMain: TfrmAbonentMain;

implementation

uses AbonentDataModule, Saldo;

{$R *.dfm}

procedure TfrmAbonentMain.RadioGroup1Click(Sender: TObject);
var
  sort: string;
begin
  AbonentDataModule.DataModule1.idAbonent.Close;
  AbonentDataModule.DataModule1.idAbonent.SelectSQL.Clear;
  AbonentDataModule.DataModule1.idAbonent.SelectSQL.Add
('SELECT * FROM Abonent');
  case RadioGroup1.ItemIndex of
  //устанавливаем режим сортировки
```

```
  0: sort := '';
  1: sort := 'ORDER BY AccountCD';
  2: sort := 'ORDER BY FIO';
  end;
  AbonentDataModule.DataModule1.idAbonent.SelectSQL.Add(sort);
  AbonentDataModule.DataModule1.idAbonent.Open;
end;

procedure TfrmAbonentMain.bLocateClick(Sender: TObject);
begin
  if eAccountCD.Text <> '' then
    if not AbonentDataModule.DataModule1.idAbonent.Locate('AccountCD',
eAccountCD.Text, []) then
      ShowMessage('Запись не найдена!')
  else
    ShowMessage('Введите номер лицевого счета!');
end;

procedure TfrmAbonentMain.bSaldoClick(Sender: TObject);
var
  SaldoForm: TfrmSaldo; // форма истории оплат и начислений
begin
  SaldoForm := TfrmSaldo.Create(Self);
  SaldoForm.ShowModal;
end;

procedure TfrmAbonentMain.N1Click(Sender: TObject);
var
  SaldoForm: TfrmSaldo; // форма истории оплат и начислений
begin
  SaldoForm := TfrmSaldo.Create(Self);
  SaldoForm.ShowModal;
end;

procedure TfrmAbonentMain.N2Click(Sender: TObject);
begin
  Application.Terminate;
end;

end.
```

Код модуля формы истории оплат и начислений **Saldo**

```
unit Saldo;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Menus, StdCtrls, Grids, DBGrids, Mask, DBCtrls;
```

```
type
  TfrmSaldo = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    DBGrid1: TDBGrid;
    bPay: TButton;
    bClose: TButton;
    eFIO: TDBEdit;
    ePhone: TDBEdit;
    eAddress: TDBEdit;
    procedure bCloseClick(Sender: TObject);
    procedure bPayClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure OpenSaldo(AccountCD: string; Beg-
Month,BegYear,EndMonth,EndYear: Word);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmSaldo: TfrmSaldo;
  AccountCD: string;
  BegYear,BegMonth: Word;
  CurYear,CurMonth,CurDay: Word;

implementation

uses AbonentDataModule, Pay, AbonentMainForm;

{$R *.dfm}

procedure TfrmSaldo.OpenSaldo(AccountCD: string; Beg-
Month,BegYear,EndMonth,EndYear: Word);
begin
  //работа с набором данных
  AbonentDataModule.DataModule1.idSaldo.Close;
  //заполнение параметров
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('AccountCD').
AsString := AccountCD;
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('BegMes').
AsInteger := BegMonth;
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('BegGod').
AsInteger := BegYear;
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('EndMes').
AsInteger := EndMonth;
```

```
  AbonentDataModule.DataModule1.idSaldo.ParamByName ('EndGod').
AsInteger := EndYear;
  //открытие набора данных
  AbonentDataModule.DataModule1.idSaldo.Open;
end;

procedure TfrmSaldo.FormCreate(Sender: TObject);
begin
  //номер лицевого счета абонента из текущей строки списка главной формы
  AccountCD := AbonentMainForm.frmAbonentMain.DBGrid1.DataSource.DataSet.
FieldByName('AccountCD').AsString;
  //получение месяца и года начала истории
  AbonentDataModule.DataModule1.iqMinMonth.Close;
  AbonentDataModule.DataModule1.iqMinMonth.ParamByName
('AccountCD').AsString := AccountCD;
  AbonentDataModule.DataModule1.iqMinMonth.Open;
  BegYear := AbonentDataModule.DataModule1.iqMinMonth.FieldByName
('Min_year').AsInteger;
  BegMonth := AbonentDataModule.DataModule1.iqMinMonth.FieldByName
('Min_month').AsInteger;
  //разбиение текущей даты
  DecodeDate(Now,CurYear,CurMonth,CurDay);
  //формирование набора данных с сальдо
  OpenSaldo(AccountCD,BegMonth,BegYear,CurMonth,CurYear);
end;

procedure TfrmSaldo.bPayClick(Sender: TObject);
var
  PayForm: TfrmPay; // форма ввода оплаты
begin
  PayForm := TfrmPay.Create(Self);
  if PayForm.ShowModal = mrOK then
  begin
    //обновляем данные на форме сальдо
    AbonentDataModule.DataModule1.itaSaldo.Commit;
    OpenSaldo(AccountCD,BegMonth,BegYear,CurMonth,CurYear);
  end;
end;

procedure TfrmSaldo.bCloseClick(Sender: TObject);
begin
  Close;
end;

procedure TfrmSaldo.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  AbonentDataModule.DataModule1.itaSaldo.Commit;
end;
```

```
end.
```

Код модуля формы ввода оплаты **Pay**

```
unit Pay;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, DBCtrls, Mask, ComCtrls;

type
  TfrmPay = class(TForm)
    lcFIO: TDBLookupComboBox;
    lcService: TDBLookupComboBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    bAdd: TButton;
    bClose: TButton;
    Label5: TLabel;
    deMonth: TDBEdit;
    deYear: TDBEdit;
    deSum: TDBEdit;
    Label6: TLabel;
    DateTimePicker1: TDateTimePicker;
    procedure FormCreate(Sender: TObject);
    procedure bCloseClick(Sender: TObject);
    procedure bAddClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmPay: TfrmPay;

implementation

uses AbonentDataModule, AbonentMainForm;

{$R *.dfm}

procedure TfrmPay.FormCreate(Sender: TObject);
var
  AccountCD: string;
begin
  //переоткрываем набор данных, что влечет старт транзакции
```

```
  AbonentDataModule.DataModule1.idPaySum.Close;
  AbonentDataModule.DataModule1.idPaySum.Open;
  //добавляем пустую запись
  AbonentDataModule.DataModule1.idPaySum.Append;
  //автоматически заполняем номер лицевого счета
  AccountCD := AbonentMainForm.frmAbonentMain.DBGrid1.DataSource.DataSet.
FieldByName('AccountCD').AsString;
  AbonentDataModule.DataModule1.idPaySum.FieldByName
('AccountCD').AsString := AccountCD;
end;

procedure TfrmPay.bAddClick(Sender: TObject);
begin
  try
    //заполняем запись по введенным пользователем значениям
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('ServiceCD').AsInteger := lcService.KeyValue;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PayMonth').AsInteger := deMonth.Field.Value;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PayYear').AsInteger := deYear.Field.Value;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PaySum').AsCurrency := deSum.Field.Value;
    AbonentDataModule.DataModule1.idPaySum.FieldByName
('PayDate').AsDateTime := DateTimePicker1.Date;
    //подтверждаем изменения набора данных
    AbonentDataModule.DataModule1.idPaySum.Post;
    AbonentDataModule.DataModule1.itaPaySum.Commit;
    ModalResult:=mrOK;
  except
    AbonentDataModule.DataModule1.itaPaySum.Rollback;
    ShowMessage('Неверно заполнены данные!');
  end;
end;

procedure TfrmPay.bCloseClick(Sender: TObject);
begin
  AbonentDataModule.DataModule1.itaPaySum.Rollback;
  ModalResult:=mrNone;
end;

end.
```