

Глава 14. Delphi и базы данных.....	314
14.1 Теория реляционных баз данных.....	315
14.2 Создание первой базы данных Access.....	318
14.3 Пример работы с базами данных.....	322
14.3.1 Свойства компонента TADOTable.....	326
14.3.2 Методы компонента TADOTable.....	328
14.4 Управление отображением данных.....	328



Глава 14. Delphi и базы данных



Базы данных считаются основным козырем Delphi. Это действительно так. Хотя этот язык не создавался специально под эту сферу, но реализация работы с данными здесь просто поражает. Даже специализированные языки для работы с базами данных (такие, как MS Visual FoxPro) явно уступают по простоте и мощи программирования этого типа приложений.

Delphi скрывает все сложности и в то же время даёт тебе величайшую мощь. Ещё не было такой задачи, которую я не смог бы реализовать на Delphi за короткий промежуток времени. А главное, что всё это реализовано очень удобно и легко для понимания.

Когда я первый раз услышал про базы данных, я сильно испугался. Мне казалось это очень сложным. Но когда я увидел, что в Delphi можно создавать простые приложения, но даже со сложными базами без единой строчки кода, я просто влюбился в эту среду разработки. В этой главе мы познакомимся с основами баз данных и напишем несколько полезных примеров.

Для примеров я буду использовать базы Access и современный формат xml. Я советую использовать эти базы в качестве локальных, потому что они поддерживаются большинством систем и отличаются высокой надёжностью.

В последствии я покажу тебе самые простые и распространённые базы dbf и paradox. Лично я их стараюсь не использовать в своих проектах из-за их ненадёжности, и потому что в них регулярно нарушается индексная целостность, что приводит к неработоспособности программ. Но из-за их распространённости, знать принципы работы с ними просто необходимо. Даже локальная версия 1С Предприятия использует этот ужасный формат DBF. Так что если ты захочешь написать свою программу для работы с чужими данными, то ты просто обязан знать, как работать с этим форматом данных.



14.1 Теория реляционных баз данных

Ещё десять лет назад, программирование баз данных было очень сложным занятием. За какие-либо достижения в этой области многие программисты получили в своё время докторские степени. Сейчас уже такое трудно себе представить, потому что благодаря Delphi, процесс написания программ упростился, а количество разновидностей баз данных уже исчисляется десятками.

Ключ	Фамилия	Имя	Отчество
1	Иванов	Иван	Петрович
2	Петров	Сергей	Владимирович
3	Сидоров	Алексей	Викторович
4	Смирнов	Андрей	Сергеевич

Таблица 14.1 Пример простейшей базы данных

Базы данных делятся на локальные (установленные на компьютере клиента, там же где и работает программа) и удалённые (установленные на сервере, удалённом компьютере). Серверные базы данных располагаются на удалённом компьютере и работают под управлением серверного программного обеспечения. К их главным преимуществам можно отнести возможность работы с одной базой данных одновременно несколькими пользователями, и при этом осуществляется минимальная нагрузка на сеть.

Есть ещё сетевые базы данных, но их мы рассматривать не будем, потому что они создают слишком большую нагрузку на сеть и неудобны в работе, как для программиста, так и для конечного пользователя. Поэтому работать с такими базами мы не будем, и я никому не рекомендую этого делать. Почему? Это я попытаюсь тебе сейчас объяснить.

Когда твоя программа присоединяется к сетевой базе данных, то она выкачивает с сервера практически полную его копию. Если ты внёс изменения, то твоя копия полностью закачивается обратно. Это очень неудобно, потому что создаётся большая нагрузка на сеть из-за излишней перекачки данных.

При клиент-серверной технологии программа клиент посылает простой текстовый запрос на сервер на получение каких-либо данных. Сервер обрабатывает его и возвращает только необходимую порцию данных. Когда нужно изменить какие-то данные, опять посылается запрос к серверу на их изменение и сервер изменяет данные в своей базе. Таким образом, по сети происходит перекачка в основном только текстовых запросов, которые в основном занимают меньше килобайта. Все данные обрабатывает сервер, а значит, машина клиента загружается намного меньше и не так сильно требовательна к ресурсам. Сервер отправляет клиенту только самые необходимые данные, а значит, отсутствует излишняя перекачка копии все базы.



Благодаря всему этому, сетевые базы данных уже устарели и практически не используются. Их практически полностью вытесняет технология клиент-сервер. А вот локальные базы данных будут жить всегда. Может измениться формат их хранения или добавиться какие-то новые функции, но сами базы данных будут существовать.

В этой главе мы затронем только локальные базы данных, а серверные рассмотрим немного позже. Для дальнейшего рассмотрения нам надо определить новое понятие – *таблица*. Пока что я говорил только общие принципы, поэтому использовал общее понятие *баз данных*. Таблица базы данных – это как двух мерный массив, в котором в столбец выстроены данные (яркий пример таблицы – Excel). База данных – грубо говоря это всего лишь файл, в котором может храниться от одной до нескольких таблиц. Большинство локальных баз данных могут хранить только одну таблицу (dBase, Paradox, XML). Но есть представители локальных баз, где в одном файле заключено несколько таблиц (например Access, который мы будем рассматривать в этой главе).

Локальные базы данных

Из локальных баз данных мы будем рассматривать реляционные, как самые распространённые. Что такое реляционная база данных? Это таблица, в которой в качестве столбцов выступают имена хранимых в ней данных, а каждая строка хранит сами данные. Таблица базы данных похожа на электронную таблицу Excel (если быть точнее, то Excel хранит свои данные в виде собственного формата, построенного на основе технологии баз данных). Локальные таблицы баз данных могут храниться на локальном жёстком диске или централизованно сохраняться на сетевой диск файлового сервера. Эти файлы можно копировать с помощью стандартных средств как любой другой файл, потому что сами таблицы базы данных не привязаны к определённому месту расположения. Главное, чтобы программа могла найти твою таблицу.

В каждой таблице должно быть одно уникальное поле, которое однозначно будет идентифицировать строку. Это поле называется ключевым. Эти поля очень часто используются для связывания нескольких таблиц между собой (с этим мы ещё познакомимся). Но даже если у тебя таблица не связана, ключевое поле всё равно обязательно. Представь, что ты пишешь телефонную базу данных. Сколько у тебя будет "Ивановых"? Как ты будешь отличать их? Вот тут тебе поможет ключ. В качестве ключа желательно использовать численный тип и если позволяет база данных, то будет лучше, если он будет типа "autoincrement" (автоматически увеличивающееся/уменьшающееся число или счётчик).

Имена столбцов в таблице базе данных, также должны быть уникальными, но в этом случае не обязательно числовыми. Их можно называть как угодно, лишь бы было уникально и тебе понятно, а остальное никого не касается.

Каждый столбец (поле базы данных) обязательно должен иметь определённый тип. Количество типов и их разновидности зависит от типа базы данных, например формат dBASE (файлы с расширением DBF) поддерживает только 6 типов, а Paradox уже до 15.

База данных может храниться в одном файле (Access) или в нескольких (Paradox, dBase). Точнее сказать, данные таблицы всегда хранятся в одном файле, а вот дополнительная информация может располагаться в отдельных файлах. В качестве дополнительной информации могут быть индексы, ограничения или список значений по умолчанию для конкретных полей. Если хотя бы один из файлов запортился или будет удалён, то данные могут стать недоступными для редактирования.

Что такое *индексы*? Очень часто данные из таблиц подвергаются каким-то изменениям, поэтому прежде чем произвести редактирование над какой-либо строкой, необходимо её найти. Даже статические таблицы, использующиеся в качестве справочников, тоже подвергаются операциям поиска перед выводом запрашиваемых данных. Поиск достаточно трудоёмкая операция, особенно если таблица содержит очень много строк. Индексы направлены на ускорение этой процедуры, а так же могут использоваться в качестве отправной точки при сортировке. На данном этапе тебе достаточно знать, что не проиндексированное поле невозможно упорядочить.

Если тебе надо, чтобы какая-то таблица была упорядочена по полю «*Фамилия*», то это поле надо сначала проиндексировать. Затем нужно только указать, что таблица должна работать сейчас с таким-то индексом, и она сортируется автоматически.

Delphi и базы данных

Для работы с базами в Delphi есть несколько наборов компонент. Каждый набор очень хорошо подходит для решения определённого круга задач. Почему такое разнообразие компонентов? Все они используют разные технологии доступа к данным и отличаются по возможностям. В отличие от Microsoft, которая встроила в свои продукты разработки только технологию доступа к данным ADO собственной разработки, фирма Borland дала нам разнообразие средств работающих через разные технологии и не ограничивает нас только своими разработками. Такое положение вещей даёт нам громадные преимущества перед другими программистами.

Помимо этого есть группы, которые могут использоваться в любом случае, и здесь я попробую дать краткий обзор доступных нам средств.

На закладке *Data Access* расположены основные компоненты доступа к данным. Эти компоненты общие для всех и могут использоваться совместно с другими группами компонентов.

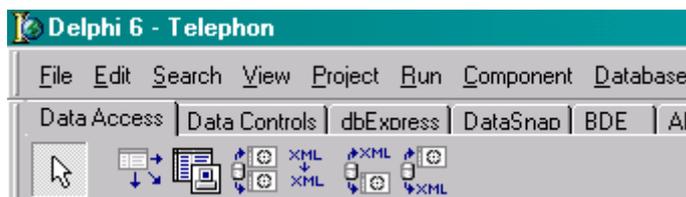


Рис 14.1.1 Закладка *Data Access* палитры компонентов

На закладке *Data Controls* расположены компоненты для отображения и редактирования данных в таблицах. Эти компоненты так же используются в не зависимости от используемой технологии доступа к данным.



Рис 14.1.2 Закладка *Data Controls* палитры компонентов

Закладка *BDE* содержит компоненты, позволяющие получить доступ к базам данных по технологии, разработанной фирмой Borland под названием Borland Database Engine. Эта технология сильно устарела и поставляется только для совместимости со старыми версиями. Не смотря на это, она хорошо работает со старыми типами баз данных, такими как Paradox и dBase.

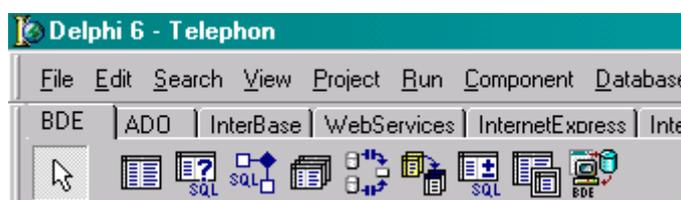


Рис 14.1.3 Зкладка BDE палитры компонентов

DBExpress – это новая технология доступа к данным фирмы Borland. Она отличается большей гибкостью и хорошо подходит для программирования клиент серверных приложений, использующих базы данных. Компоненты с одноимённой закладки я советую использовать с базами данных построенных по серверной технологии, например, Oracle, DB2 или MySQL.

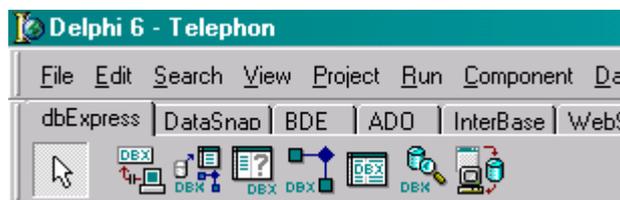


Рис 14.1.4 Зкладка dbExpress палитры компонентов

ADO (*Active Data Objects*) – технология доступа к данным, разработанная корпорацией Microsoft. Очень хорошая библиотека, но я рекомендую её использовать только с базами данных Microsoft, а именно MS Access или MS SQL Server. Её так же можно использовать, если у тебя специфичный сервер баз данных, который может работать только через ODBC.

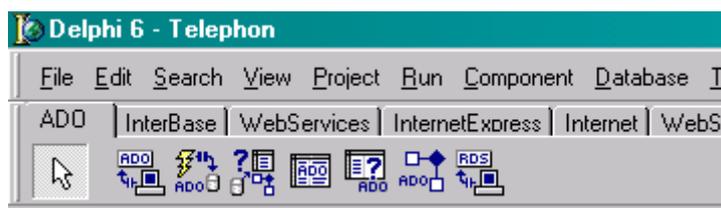


Рис 14.1.5 Зкладка ADO палитры компонентов

Работа с базами данных Access идёт через специальную надстройку DAO, которая может устанавливаться на компьютер вместе с программой Office или идти как отдельная установка. Так что если твоя программа не будет работать на компьютере клиента, то надо позаботиться о установке DAO на этот компьютер.

Я не ставлю своей целью расписать абсолютно все эти компоненты, но я постараюсь дать необходимую информацию, чтобы можно было написать профессиональные приложения для работы с базами данных.

14.2 Создание первой базы данных Access

Сейчас я постараюсь подробно рассказать, как создавать и использовать базы данных Access. Для последующей работы необходимо, чтобы на твоём компьютере был установлен MS Office и его компонент MS Access. Именно в нём и будут создаваться базы, а вот работать с ними мы будем уже из Delphi.

Запусти Access и выбери в меню Файл->Создать. В мастере создания базы выбери пункт "База данных" и нажми "ОК" (рисунок 14.2.1). Тебе предложат выбрать имя базы и место расположения, укажи что угодно, а назвал свой файл Database.mdb .

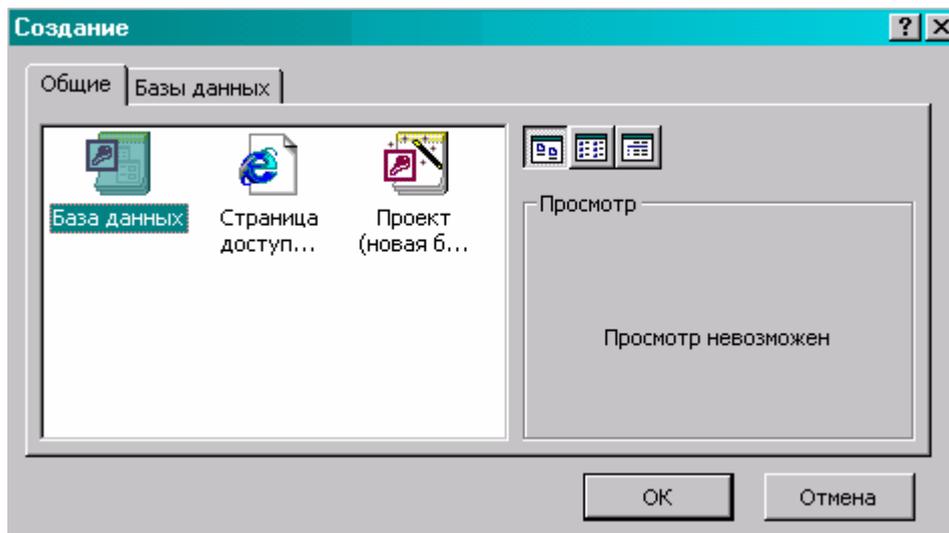


Рис 14.2.1 Окно создания новой базы данных

После этого Access создаст базу и сохранит её по указанному пути. А ты увидишь окно как на рисунке 14.2.2, в котором и происходит работа с базой. С левой стороны окна находится колонка выбора объектов, с которыми ты хочешь работать. Первым находится пункт "Таблицы" (он выделен по умолчанию) который и будет нас интересовать. Если этот объект у тебя не выделен, то выдели его. В окне справа находится три пункта:

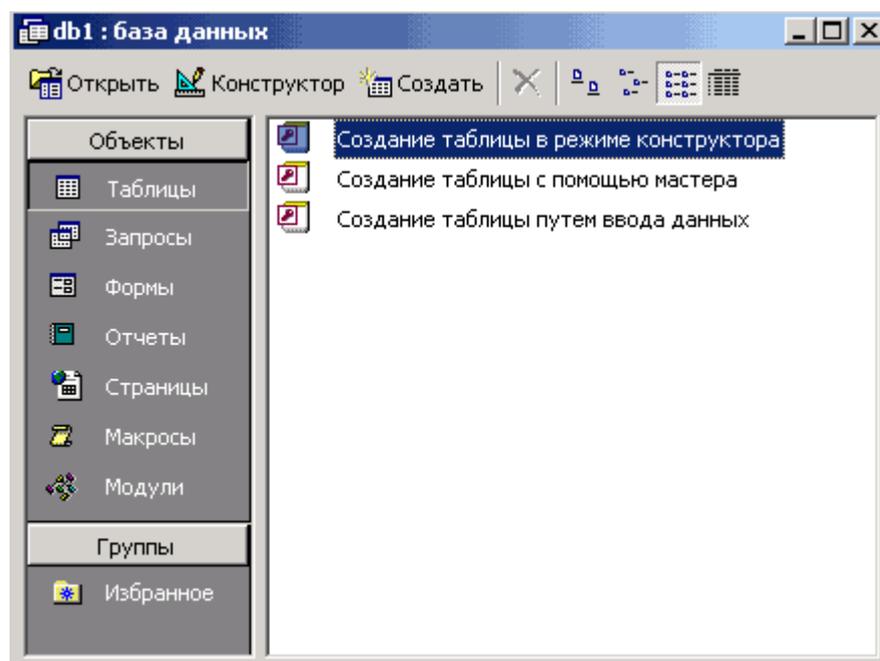


Рис 14.2.2 Окно создания новой базы данных

1. Создание таблицы в режиме конструктора
2. Создание таблицы с помощью мастера
3. Создание таблицы путём ввода данных

С помощью этих команд можно создать таблицы внутри созданной базы данных, Access, которая может хранить в одном файле несколько таблиц.

Все данные в базах данных хранятся в виде двумерных таблиц. На рисунке 14.2.3 ты можешь видеть пример простой базы данных, состоящей из семи колонок и множества строк.

Код1	Код	Материал	Наименовани	Ед	Производитель	Всего
1020	39000180	Автомат.выкл.		ШТ		5
1025	39000150	Автомат.выкл.		ШТ		3
1026	39000160	Автомат.выкл.		ШТ		5
1027	39000170	Автомат.выкл.		ШТ		5
1028	39000190	Автомат.выкл.		ШТ		5
1029	29006654	Автоматич.вык		ШТ		10
1030	29006651	Автоматич.вык		ШТ		10
1031	29006649	Автоматич.вык		ШТ		27
1032	29006492	Автоматич.вык		ШТ		11
1033	71200610	Амортизатор 0		ШТ		6

Рис 14.2.3 Пример простой таблицы

Колонки в таблицах называются полями, и по ним определяется, какие именно данные хранятся в таблице. Давай попробуем создать базу данных телефонного справочника, чтобы увидеть всё на практике. Щёлкни по "Создание таблицы в режиме конструктора" чтобы создать новую таблицу в базе данных. Перед тобой откроется окно, как на рисунке 14.2.4.

Имя поля	Тип данных	Описание
Key1	Числовой	
Фамилия	Текстовый	
Имя	Текстовый	
Телефон	Текстовый	
e-mail	Текстовый	
Город	Текстовый	

Свойства поля

Общие | Подстановка

Размер поля: Длинное целое

Формат поля:

Число десятичных знаков: Авто

Маска ввода:

Подпись:

Значение по умолчанию: 0

Условие на значение:

Сообщение об ошибке:

Обязательное поле: Нет

Индексированное поле: Да (Совпадения не допускаются)

Имя поля может состоять из 64 знаков с учетом пробелов. Для справки по именам полей нажмите клавишу F1.

Рис 14.2.4 Окно создания таблицы

Сверху находится сетка, в которой ты вводишь поля таблицы, их тип и описание (последнее не обязательно). Когда ты вписал в сетку имя нового поля и указал тип, внизу окна появляются свойства нового поля. В зависимости от типа поля изменяется и количество свойств. Вот самые основные:

- Максимальная длина поля. Для текстового поля размер не может быть больше 255. Если текст длинее, то надо использовать "Поле Мемо".

- Формат поля. Здесь ты можешь указать внешний вид данных. Например, поле может выглядеть как "Yes/No" для логических полей, или например "mm уууу" для поля даты.
- Маска ввода. Здесь мы вводим маску, которая отвечает за отображение поля при редактировании. Если ты щёлкнешь на кнопке с точками "..." в строке "Маска ввода", то увидишь мастер создания маски.
- Значение по умолчанию. Умолчание, оно и в аффрике по умолчанию.
- Обязательное поле. Если пользователь не введёт сюда значение, то появится сообщение об ошибке. Такое поле не может быть пустым.
- Пустые строки. Похоже на предыдущий, потому что это поле тоже не может быть пустым.
- Индексированное поле. Может быть неиндексированным, индексированным с допуском совпадений, и индексированным без допуска совпадений. Основной индекс всегда без допуска совпадений. Остальные желательно с допуском.
- Сжатие Юникод - позволяет сжать данные в соответствии с Юникод.

Создай шесть полей:

1. Имя поля - *Key1*. Тип - счётчик. Это у нас будет ключик. Размер поля - "Длинное целое". Индексированное поле - "Да (Совпадения не допускаются)".
2. Имя поля – *Фамилия*. Тип - текстовый. Размер поля - 50. Индексированное поле - "Да (Допускаются совпадения)".
3. Имя поля – *Имя*. Тип - текстовый. Размер поля - 50. Индексированное поле - "Да (Допускаются совпадения)".
4. Имя поля – *Телефон*. Тип - текстовый. Размер поля - 10. Индексированное поле - "Да (Допускаются совпадения)".
5. Имя поля - *e-mail*. Тип - текстовый. Размер поля - 20. Индексированное поле - "Да (Допускаются совпадения)".
6. Имя поля – *Город*. Тип - числовой. Размер поля - Длинное целое. Индексированное поле - "Нет". Почему город не строковый, ведь названия городов – это текст? Пока я не буду объяснять этого феномена и оставляю его на потом. Чуть позже я покажу, почему город должен быть числовым.

Помимо этого, у всех полей значение "*Обязательно поле*" стоит в "*Нет*", и "*Пустые строки*" выставлено в "*Да*". Если ты сделаешь поле обязательным, то во всех строках обязательно должно быть заполнено соответствующее поле. Если ты запретишь пустые строки (поставишь «*Нет*»), то в указанном поле должно быть обязательно что-то введено, иначе произойдёт ошибки. В реальных условиях, если какое-то поле обязательно должно иметь значение, то лучше сделать его обязательным. Не надо надеяться на добропорядочность пользователя, потому что они слишком часто подводят. Пусть лучше база данных следит за правильностью данных.

Теперь выдели первое поле (*Key1*), щёлкни правой кнопкой мыши и выбери пункт "Ключевое поле" (рисунок 14.2.5). Задание ключевого поля является обязательным действием, если ты этого не сделаешь, то таблица не сможет редактироваться, а это значит, что в неё нельзя будет добавить строки.

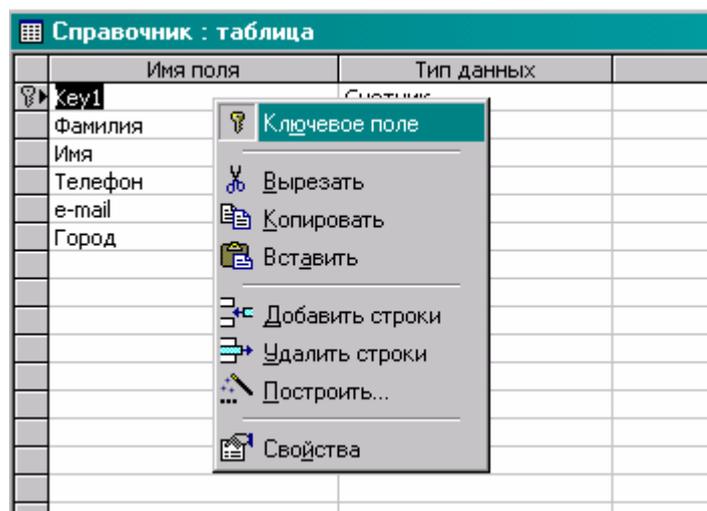


Рис 14.2.5 Задание ключевого поля

Всё, теперь таблицу можно сохранять и закрывать. На вопрос: «Сохранить таблицу» отвечай положительно и сохраняй под именем «Справочник».

Наша первая база данных готова. Закрывай её и переходи к следующей части моей книги, где мы напишем первый пример для работы с нашей базой и таблицей.

14.3 Пример работы с базами данных

Мы пишем программу, которая будет работать с базой данных MS Access. Я уже говорил, что для разработок от MS лучше всего использовать ADO. Давай напишем наше первое приложение для работы с базой данных.



- ADOConnection.

Создай новый проект. Теперь брось на форму компонент *ADOConnection* с закладки ADO палитры компонентов. Теперь настроим соединение с сервером, которое должно быть прописано в свойстве *ConnectionString*. Для этого надо дважды щёлкнуть по строке *ConnectionString* и перед нами открывается окно, как на рисунке 14.3.1.

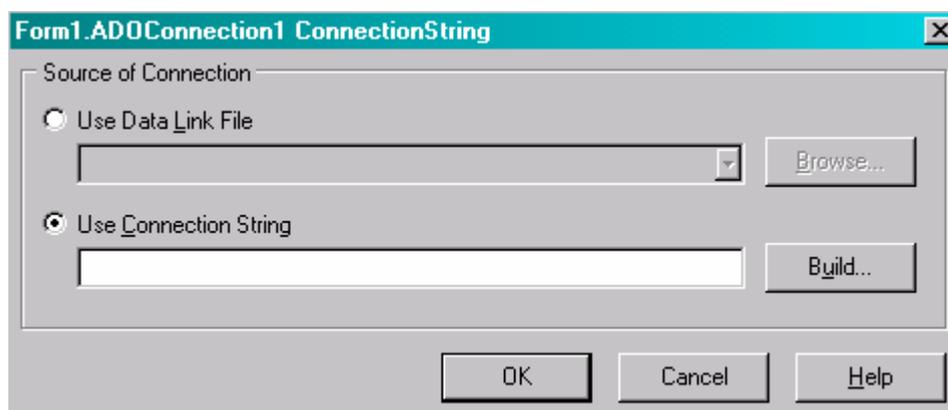


Рис 14.3.1 Окно создания подключения к базе

Здесь перед нами стоит выбор:

1. Использовать специальный файл (*Use Data Link File*);
2. Использовать строку подключения (*Use Connection String*).

Второе, на мой взгляд, более предпочтительно, поэтому я покажу, как создать строку подключения. Для этого щёлкаем кнопку Build и перед нами открывается ещё одно окно, показанное на рисунке 14.3.2.

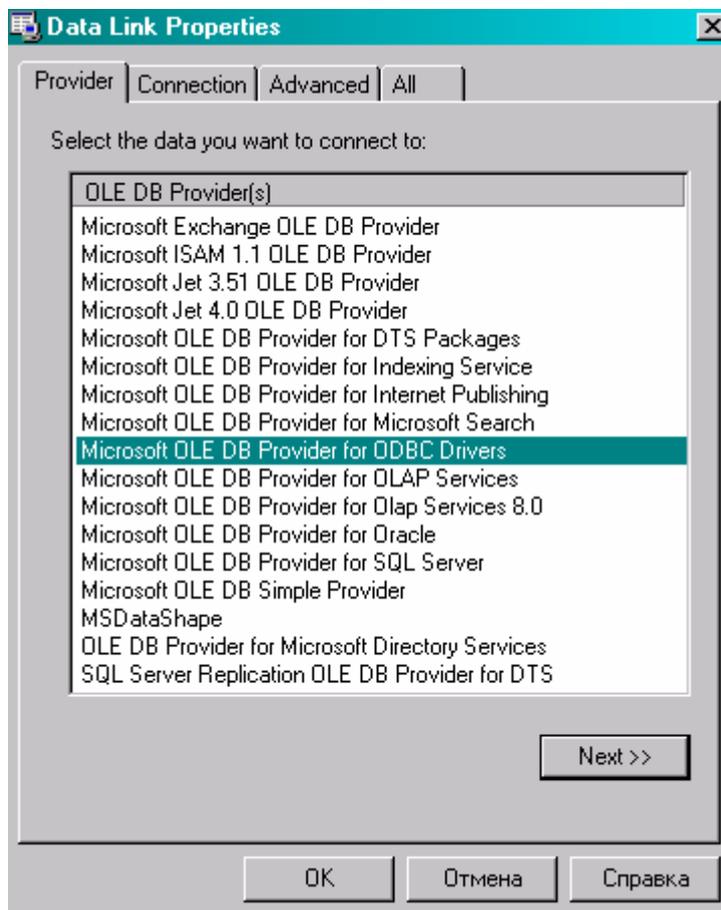


Рис 14.3.2 Окно создания строки подключения

На закладке *Provider* перечислены все доступные ADO драйверы доступа к базам данных. Если какого-то драйвера нет, то можно попробовать выделенный по умолчанию «*Microsoft OLE DB Provider for ODBC Drivers*». Этот драйвер позволяет получить доступ к базе данных через ODBC драйвер, которые есть к большинству существующих баз данных (единственное, он может быть не установленным на твоём компьютере).

В нашем случае, для доступа к базам данных MS Access используется драйвер «*Microsoft Jet OLE DB Provider*». Такой драйвер обязательно устанавливается на машину вместе с MS Office, а в последних версиях Windows он устанавливается по умолчанию.

На моей машине установлено сразу две версии этого драйвера, поэтому я выберу более новый - «*Microsoft Jet 4.0 OLE DB Provider*». После этого нажимаем кнопку Next, или переходим на закладку «*Connection*».

Вид закладки *Connection* зависит от выбранного драйвера. В нашем случае она должна выглядеть, как показано на рисунке 14.3.3.

Первым делом, в этом окне надо ввести имя (если надо то и путь) базы данных в строку «*Select or enter a database name*». Если база данных будет располагаться в той же директории, что и запускной файл, то путь указывать не надо. Я вообще советую хранить базы в одной директории с запускными файлами. Если ты будешь держать файлы

отдельно от запускового, то тебе придётся указывать полный путь, а это может вызвать проблемы при переносе программы на другой компьютер. Ведь там программа будет искать базу по указанному пути, который может измениться. Если хочешь держать файлы в другой директории, то указывай относительный путь относительно текущей директории.

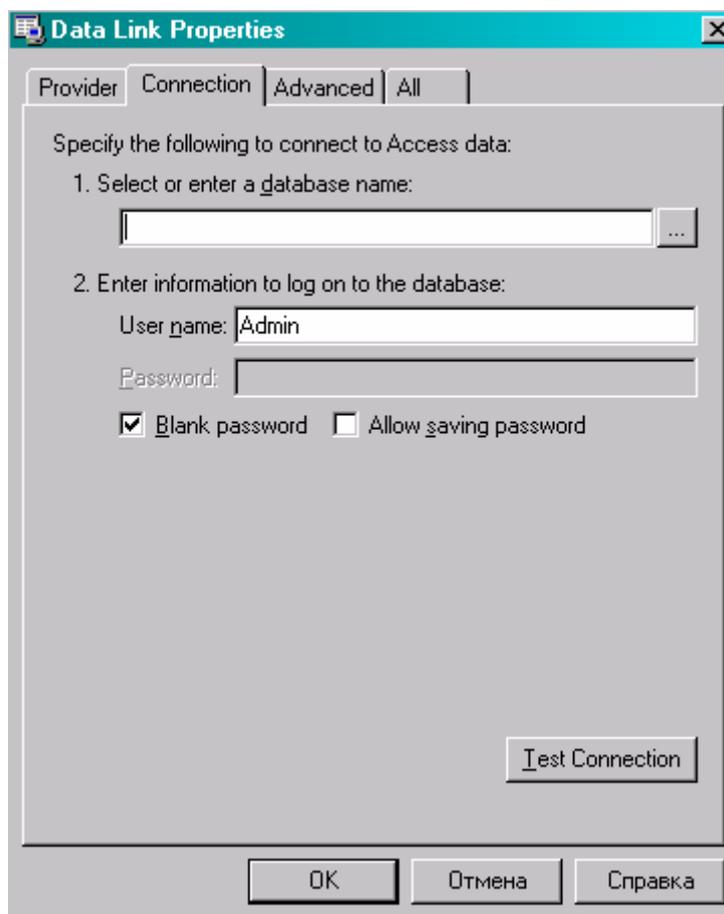


Рис 14.3.3 Закладка Connection

Чтобы легче было выбрать файл базы данных необходимо щёлкнуть по кнопке с точками справа от строки ввода.

Помимо этого нам надо заполнить следующие поля:

1. Имя пользователя (*User name*), можно оставить по умолчанию, если не заданно иное при создании базы в MS Access;
2. Пароль (*Password*) – если база имеет пароль, то его необходимо указать;
3. Пустой пароль (*Blank password*) – если пароль не нужен, то здесь желательно поставить галочку;
4. Позволять сохранять пароль (*Allow saving password*). Если здесь поставить галочку, то пароль может быть сохранён.

Как только выберешь базу данных, нажми кнопку *Test Connection*, чтобы протестировать соединение. Если всё указано правильно, то ты должен увидеть сообщение «*Test connection succeeded*». Всё, можно нажать OK, чтобы закрыть окно создания строки подключения и ещё раз OK, чтобы закрыть окно редактора строки подключения (тот, что был на рисунке 14.3.1).

Теперь в свойствах компонента *ADODConnection* отключи свойство *LoginPrompt*, выставив его в *False*. Это нужно для того, чтобы при каждом обращении к базе нас не

грузили окном ввода пароля. А теперь выставим свойство *Connected* в *True*, чтобы произошло соединение с базой.

На этом соединении можно считать оконченным. Теперь нам надо получить доступ к созданной нами таблице «Справочник». Для этого брось на форму компонент *ADOTable* с закладки *ADO* палитры компонентов. Сразу измени его свойство *Name* на *BookName*.

 - TADOTable.

В этом компоненте тоже есть свойство *ConnectionString* и его так же можно настраивать. Почему «можно»? Да потому что, чтобы этого не делать мы поставили на форму компонент *ADOConnection*. Теперь мы можем указать у нашего компонента *BookName* в свойстве *Connection*, созданный нами компонент соединения с базой данных. Щёлкни по выпадающему списку в свойстве *Connection* и выбери там единственный пункт *ADOConnection1*. Теперь нам не надо заполнять свойство *ConnectionString*.

Теперь в свойстве *TableName* нужно выбрать имя нашей таблицы (*Справочник*). Всё, таблица и соединение указаны, можно подключаться. Для этого выставь свойство *Active* в *true*.

 - TDataSource

Для отображения данных из таблицы надо ещё установить на форму компонент *DataSource* с закладки *Data Access* палитры компонентов. Теперь этому компоненту надо указать, какую именно таблицу он должен отображать. Для этого в свойстве *DataSet* нужно из выпадающего списка выбрать нашу таблицу *BookTable*.

 - DBGrid

Всё!!! Все приготовления готовы, можно приступать к реальному отображению данных. Самый простой способ отобразить таблицу – установить компонент *DBGrid*. Это компонент-сетка, которая может отображать данные в виде таблицы. В этом же компоненте можно добавлять, удалять и редактировать строки нашей таблицы.

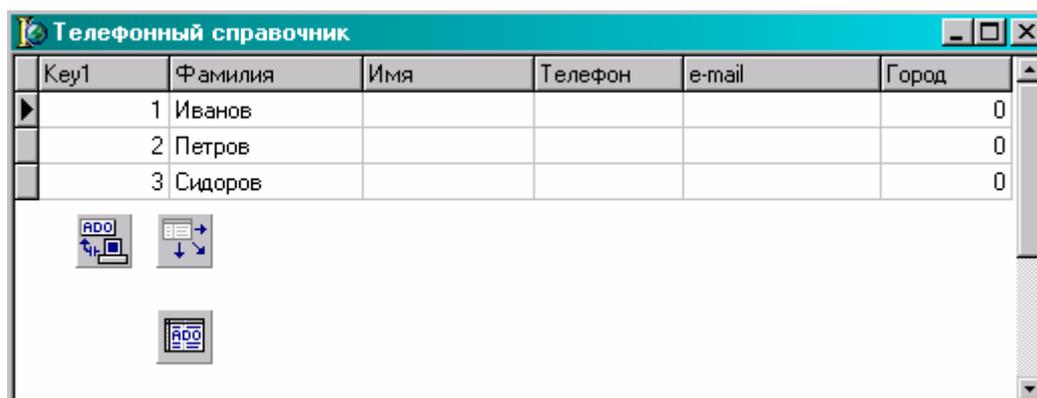


Рис 14.3.4 Форма нашего приложения

И последний этап создания нашего приложения – связывание компонента сетки с компонентом отображения таблицы. Для этого в свойстве *DataSource* компонента *DBGrid* нужно указать созданный нами компонент *DataSource1*.

Вот теперь наше приложение готово!!! Может ты не заметил, но мы не написали ни одной строчки кода :). Вот до какой степени Delphi упрощает процесс программирования баз данных, что даже программировать ничего не надо.

Попробуй запустить этот пример и создать несколько строк, отредактировать уже существующие и удалить что-нибудь. Для вставки строки используй клавишу **Ins**, а для удаления **Ctrl+Del**.

 На компакт диске, в директории \Примеры\Глава 14\Database1 ты можешь увидеть пример этой программы.

14.3.1 Свойства компонента TADOTable

Компонент *TADOTable* имеет множество полезных свойств. Большинство из них просты в использовании, поэтому чтобы не писать множество примеров на их использование, я здесь кратко опишу основные из них. В дальнейшем с какими-то мы познакомимся на практике, а какие-то останутся для тебя как дополнительная информация к размышлению.

MasterSource - в этом свойстве указывается главная, по отношению к текущей таблица. Мы рассмотрим это свойство достаточно подробно и на практике, когда будем рассматривать связанные таблицы.

ReadOnly – если это свойство равно *true*, то таблицу нельзя редактировать. В этом случае данные только отображаются. Обязательно устанавливай это свойство для тех таблиц, где данные не должны изменяться и пользователь не должен вносить в них изменения.

TableDirect – это свойство отображает какой будет происходить доступ к таблице. Если этот параметр равен *true* то будет происходить прямой доступ к таблице по имени. Если *false* то незаметно для тебя будет происходить специальный SQL запрос к базе данных (о SQL запросах читай ниже). Не все базы данных позволяют работать через прямой доступ, поэтому это свойство по умолчанию равно *false*.

TableName – имя таблицы, данные которой мы хотим обрабатывать.

CacheSize – размер кэш памяти. Если здесь установить число 50, то при первом подключении к таблице компонент выберет первые 50 строк и поместит их в локальной памяти, что ускорит доступ к ним.

CanModify – свойство похоже на *ReadOnly* и указывает на возможность редактирование данных таблицы.

CommandTimeout – время ожидания выполнения команды. Когда компонент направляет команду базе данных, то он запускает таймер ожидания, по истечению которого (если команда не выполнялась) происходит сообщение об ошибке.

Connection – здесь указывается компонент *TADOConnection*, через который происходит подключение.

ConnectionString – строка подключения к базе данных.

CursorLocation – расположение курсора, который считывает данные и указывает текущую позицию в таблице. Курсор может находиться на сервере или на машине клиента.

CursorType – тип курсора. Тут возможен один из следующих вариантов:

- *ctUnspecified* расположение курсора не указано

- *ctOpenForwardOnly* – курсор может двигаться только вперёд.

- *ctKeyset* при этом курсоре изменения внесённые одним пользователем не видны остальным пользователям подключённым к этой таблице. Если с одной таблицей

работают одновременно несколько пользователей, то при таком курсоре для отображения изменений других пользователей нужно отключиться от базы и подключиться к ней снова.

- *ctDynamic* динамический курсор, при котором изменения одного пользователя видят все остальные.

- *ctStatic* статический курсор. Изменения одного пользователя не видны остальным



Внимание!!! Если курсор расположен на клиенте, то можно использовать только статический курсор. Не все типы курсоров могут работать с определённой базой данных. Одна база данных может поддерживать один тип, а другая может поддерживать всё.

Filter – строка фильтра.

Filtered – является ли таблица фильтруемой. Если здесь установить *false* то строка фильтра (*filter*) игнорируется.

IndexFieldNames – имя индексированной колонки. Индексы используются для сортировки данных или для связи между таблицами.

RecNo - номер текущей выделенной строки.

RecordCount – количество строк в таблице.

Sort - строка, в которой указывается тип сортировки. Например, для сортировки по полю «Телефон» сюда нужно записать строку: *ADOQuery1.Sort := 'Телефон ASC'*. Оператор *ASC* говорит о том, что надо сортировать в порядке возрастания. Оператор *DESC* говорит о сортировании в порядке убывания.

Active – если это свойство равно *true*, то таблица открыта.

AggFields – здесь хранятся все агрегатные поля.

AutoCalcFields – если здесь *true*, то надо автоматически пересчитывать поля.

Bof – на это свойство влиять нельзя, но если оно равно *true*, то мы находимся в начале файла.

Bookmark - здесь находится текущая закладка.

Eof - на это свойство влиять нельзя, но если оно равно *true*, то мы находимся в конце файла.

FieldCount – здесь хранится количество полей в таблице.

Fields – через это поле можно получить доступ к значениям полей. Допустим, что тебе надо узнать, какое значение хранится в 4-м поле. Для этого нужно написать *Table.Fields.Fields[4].AsString*. Метод *AsString* говорит о том, что нам надо получить значение в виде строки. Это простой метод, но я его не люблю использовать. В течении всей книги я буду обращаться к полям по именам.

FieldValues – с помощью этого свойства можно легко получить доступ к любому значению указанного поля. Имя поля нужно указывать в квадратных скобках. Например, *Table1.FieldValues['Телефон']:= '3346598'*;

FilterOption – настройки фильтра. Здесь можно указывать следующие параметры:

- *foCaseInsensitive* фильтр будет не чувствителен к регистру.

- *foNoPartialCompare* если стоит этот параметр, то сравнения будут происходить с точной копией указанного значения в фильтре. Если параметр не указан, то в фильтр будут попадать строки содержащие значение в фильтре, но не являющиеся его точной копией. Например, если в фильтре указано показывать слова «са», то в фильтр попадут все слова начинающиеся на «са» (самолёт, самокат).

Modified – если это свойство равно *true*, то в таблице были внесены изменения.

14.3.2 Методы компонента TADOTable

Как видишь, свойств очень много и большинство из них очень полезные. В течении этой главы я буду знакомить тебя с ними и мы увидим большинство свойств на практике. А теперь приготовься получить описание громадного количества методов. Они не менее полезны, и мы так же будем знакомиться с большинством из методов на практике

BookmarkValid – этот метод проверяет правильность закладки. В качестве единственного параметра нужно указать закладку типа **TBookmark** и если она является действительной, то результатом будет *true*.

CancelUpdates - отменить обновления сохранённые в кэш памяти

CompareBookmarks.- сравнение двух закладок. У метода два параметра типа **TBookmark**. Эти две закладки сравниваются. Если закладки равны, то результат равен нулю. Если первая меньше второй, то результат будет -1. Если первая больше второй, то результат равен единице.

DeleteRecords – удалить записи. У метода один параметр – какие записи удалять. Ты можешь указать следующие значения в качестве параметра:

- arCurrent удалить только текущую запись.
- arFiltered удалить записи удовлетворяющие установленному фильтру.
- arAll – все записи.
- arAllChapters удалить записи во всех разделах ADO.

Append – добавить новую запись в конец таблицы.

Cancel – отменить изменения текущей строки, если изменения ещё не были сохранены с помощью метода *Post*.

Close – закрыть таблицу.

Delete – удалить текущую строку.

Edit – перейти в режим редактирования. После этого можно изменять значения полей.

FieldByName – найти поле по имени. В качестве единственного параметра нужно указать имя поля в виде строки и в результате получаем ссылку на поле в виде объекта TField.

First – перейти на первую строку в таблице.

Insert – вставить новую строку в таблицу.

IsEmpty – если метод вернёт *true* то в таблице нет записей.

Last - перейти на последнюю запись в таблице.

Next – перейти на следующую запись.

Post – принять все изменения.

Prior - двигаться на предыдущую запись в таблице.

Refresh – обновить информацию о данных.

UpdateRecord – обновить текущую запись.

14.4 Управление отображением данных

В предыдущем примере всё работает прекрасно, только вот после Key пользователю видеть абсолютно не нужно. Это поле – счётчик и его значение увеличивается автоматически. А раз пользователь не может влиять на поле и оно не несёт в себе полезной для него информации, то и видеть это поле он не должен.

Чтобы прятать от пользователя не нужные поля и показывать только то, что мы хотим и в том виде, в котором хотим, нам необходимо научиться управлять отображением данных. Но прежде чем приступить к этому, давай создадим в нашей базе ещё два поля

«Дата» и «Мобильник». Загрузи нашу базу данных в Access, щёлкни по ней правой кнопкой и в появившемся меню выбери «Конструктор».

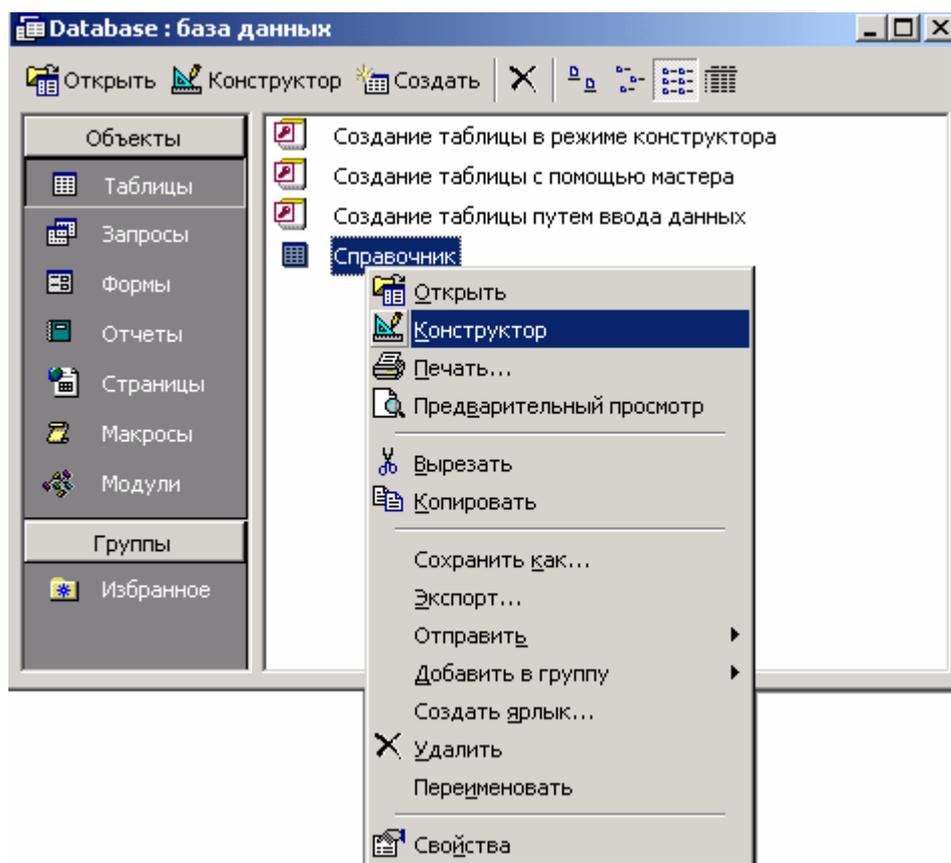


Рис 14.4.1 Редактирование таблицы

1. Добавь поле с именем «Дата», тип «Дата/время».
2. Добавь поле с именем «Мобильник», и тип «Логический». Если в строке находится мобильный телефон, то в этом поле будем ставить **true**, иначе «**false**».

Закрой таблицу. Теперь переходим в Delphi и попробуем отобразить изменения в уже созданном примере.

Для начала давай перенесём компоненты доступа к базе данных в отдельное специальное окно. Выдели компоненты *ADOConnection1*, *DataSource1* и *BookTable*. Теперь выбери из меню *Edit* пункт *Cut*, что бы эти компоненты скопировались в буфер обмена и сразу удалились с формы.

Теперь выбери из меню *File->New->Data Module* (рисунок 14.4.2). Этим ты заставишь Delphi создать специальное окно *Data Module*, которое удобно подходит для хранения компонентов доступа к базам данных.

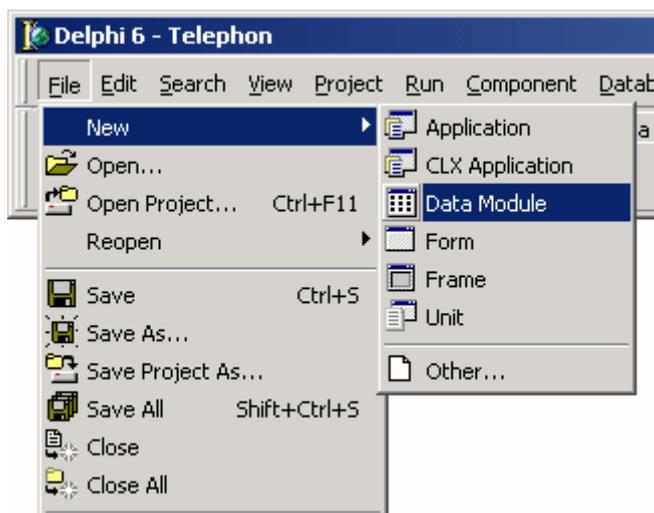


Рис 14.4.2 Создание модуля Data Module

Теперь выбери из меню *Edit* пункт *Paste*, чтобы вставить в это окно вырезанные нами компоненты. Расположи теперь эти компоненты в окне так, как тебе будет удобно. Я сделал это так, как показано на рисунке 14.4.3.

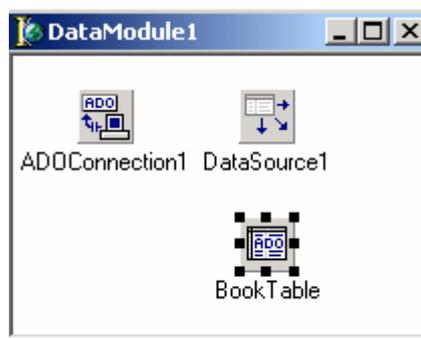


Рис 14.4.3 Окно Data Module

Теперь все компоненты, которые предназначены для доступа к базе данных будем располагать здесь, чтобы с ними удобно было работать. Сохрани новый модуль под именем *DataModuleUnit*.

Теперь открой менеджер проектом (в меню *Project* надо выбрать *Project Manager*) и расположи это окно так, чтобы тебе было удобно в любой момент получить к нему доступ. Я всегда располагаю его в правом нижнем углу экрана.

Теперь, когда тебе надо перейти из главной формы в модуль данных *DataModule* или обратно, ты легко можешь сделать это с помощью менеджера проектов, дважды щёлкнув по нужной форме.

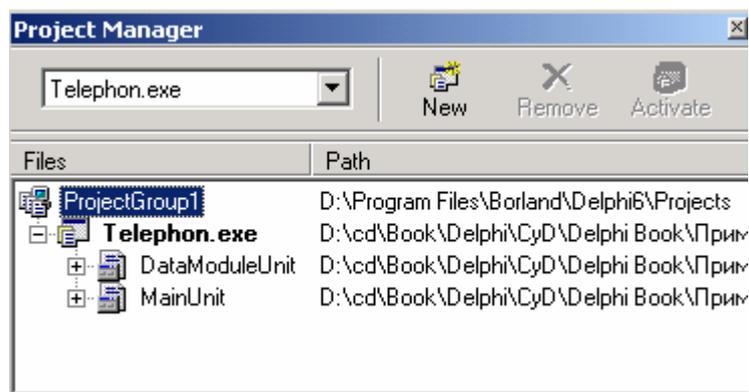


Рис 14.4.4 Менеджер проектов

Если ты хоть раз уже открывал какую-то форму из менеджера проектов и не закрывал, то её можно найти на закладках в окне редактора кода:

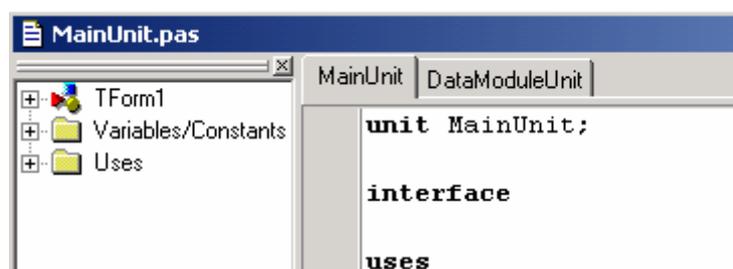


Рис 14.4.5 Закладки форм в редакторе кода

И ещё хочу напомнить, что переключаться между визуальной формой и её кодом очень удобно простым нажатием клавиши F12.

На этом инструкции по работе с оболочкой заканчиваю и пора двигаться дальше.

Перейди в главную форму и ты сразу увидишь, что в нашей сетке *DBGrid1* нет данных. Почему? Да потому что она потеряла связь с компонентами доступа к данным. Выдели сетку и щёлкни по свойству *DataSource*, и ты увидишь, что в выпадающем списке ничего нет. Это потому что все нужные компоненты мы убрали в отдельную форму и главная форма пока об этом не знает.

Чтобы форма узнала о существовании компонентов, ей нужно указать в разделе **uses** наш модуль *DataModuleUnit*. Это можно сделать вручную или выбрать из меню *File* пункт *Use Unit* (в этот момент должно быть выделено окно кода главной формы, потому что мы подключаем новый модуль именно к ней). В появившемся окне (рисунок 14.4.6) нужно выбрать имя нашего нового модуля *DataModuleUnit* (пока оно одно в списке) и нажать **OK**.

Проверь теперь в редакторе кода, чтобы после ключевого слова **implementation** появилось «*uses DataModuleUnit;*»:

implementation

uses DataModuleUnit;

{\$R *.dfm}

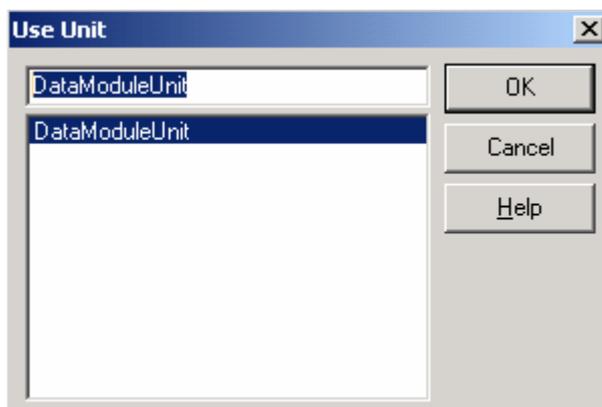


Рис 14.4.6 Окно подключения нового модуля

Вот теперь можно выделять нашу сетку *DBGrid1* и в свойстве *DataSource* указывать компонент *DataSource*, данные которого должны быть отображены в сетке (*DataModule1.DataSource1*).

Теперь переходим в модуль *DataModule* и попытаемся настроить отображение данных. Дважды щёлкни по компоненту *BookTable* и перед тобой появится окно редактирования полей базы данных (рис 14.4.7).

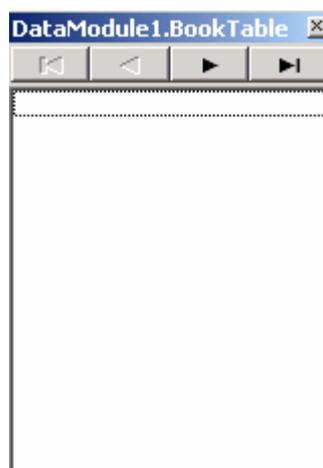


Рис 14.4.7 Окно редактирования полей базы данных

Пока что оно пустое и сюда нужно добавить все поля базы данных. Для этого щёлкни в нём правой кнопкой мыши и в появившемся меню выбери пункт *Add All Field* (Добавить все поля). Окно автоматически заполнится именами полей (рис 14.4.8).

Свойства можно переставлять местами двигая мышкой. При этом физическое расположение в базе данных не меняется, зато при отображении данных в сетке, они будут отображаться в том порядке, в котором они выстроены здесь. Так что ты в любой момент можешь изменить порядок отображения данных, не вмешиваясь в саму базу данных.

Ты можешь теперь выделять отдельные поля и в объектном инспекторе редактировать его свойства. Свойства у полей могут быть разные, в зависимости от типа поля. Я сейчас не буду их расписывать, уж лучше мы постепенно познакомимся с ними на практике и увидим их действие.

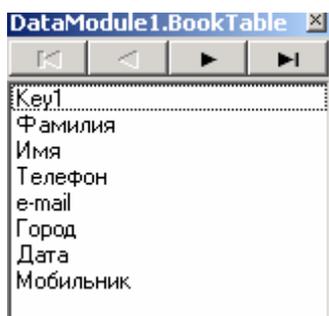


Рис 14.4.8 Заполненное окно редактирования полей базы данных

Первое, что мы должны сделать – убрать из видимости счётчик (поле *Key*). Мы уже договорились, что пользователю оно не нужно и он не должен его видеть. Выдели это свойство и в объектном инспекторе установи в свойстве *Visible* значение *false* (это свойство есть у всех полей). Сразу же можешь перейти в главную форму или запустить программу, чтобы убедиться в том, что поле *Key1* больше не отображается.

Теперь отредактируем длину отображения колонок. Для этого выдели свойство «*Фамилия*». В базе данных мы выделили под это поле 50 символов (на всякий случай). В сетке ширина колонки будет отображаться по умолчанию на всю длину. Но чаще всего фамилии не превышают 15 символов, поэтому нет смысла отображать всю длину. На много удобней отображать только 15 символов, а если что-то не поместится, то пользователь программы в любой момент сможет раздвинуть колонку и увидеть недостающие символы.

За ширину колонки отвечает свойство *DisplayWidth* (это свойство есть у всех полей). По умолчанию в нём стоит значение физической ширины поля, но мы укажем там 15. Опять же, на саму базу данных это не влияет и поле всё ещё имеет размер 50, но ширина отображаемой колонки в сетке будет 15. Точно так же сократи ширину поля «*Имя*».

Ещё несколько интересных свойств:

DefaultExpression – здесь можно указать значение по умолчанию. В дальнейшем, когда будут создаваться новые строки, то в поля будут сразу заноситься указанные здесь значения.

MaxValue – максимально допустимое значение. Если это числовое поле и оно должно изменяться в определённых рамках (например, от 0 до 100), то желательно указать эти ограничения здесь, чтобы сократить вероятность опечатки пользователем. Все люди склонны к ошибкам, так пускай программа автоматически сокращает вероятность таких ошибок.

MinValue – минимально допустимое значение.

ReadOnly – поле только для чтения. Если какой-то поле не должно изменяться, то установи у него в свойстве *ReadOnly* значение *true*. В этом случае ты обеспечишь программу от случайного изменения данного поля пользователем.

Required – если здесь *true*, то поле является обязательным и обязательно должно иметь какое-то значение. Если пользователь ничего не укажет, то программа сообщит об этом. Допустим, что какое-то поле у тебя участвует в расчётах. Если в этом поле не окажется данных, то программа может зависнуть. Есть два пути – при расчёте проверять наличие в поле данных или требовать, чтобы пользователь обязательно что-то ввёл. Второй путь предпочтительней, если это поле действительно важное. Представь запись в телефонном справочнике без телефона. Спрашивается, зачем тогда нужна эта запись если не указан телефон. Так что поле для номера телефона можно делать обязательным.

Tag – просто числовое значение, которое можно использовать по своему усмотрению.

Теперь в нашем окне помещается практически вся необходимая информация и не надо лишней раз использовать полосы прокрутки (рис 14.4.9).

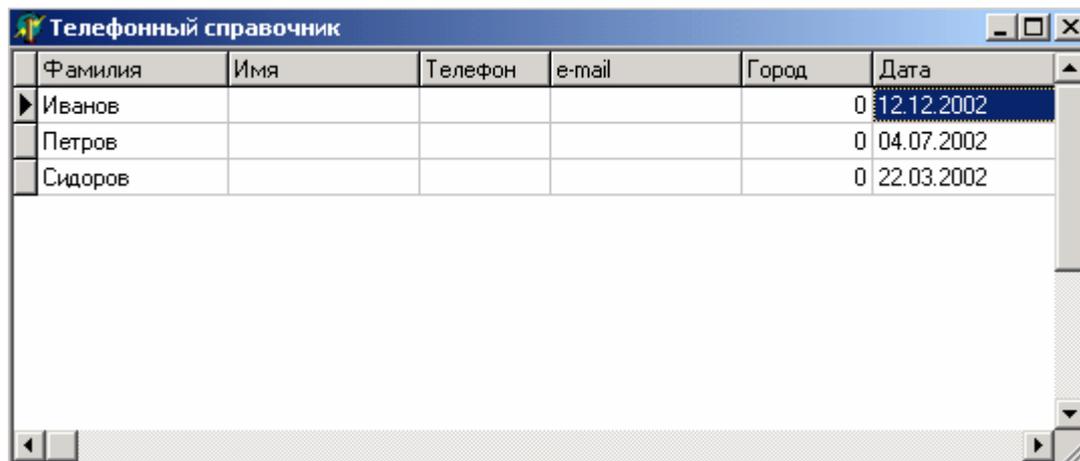


Рис 14.4.9 Улучшенное окно программы

Запусти программу и заполни поле «Дата» у всех записей любыми значениями. При заполнении будь внимателен и указывай реальные даты. Если ты введёшь недопустимое значение, то программа высветит ошибку.

При вводе данных учитывай разделитель чисел. В большинстве русскоязычных ОС Windows в качестве разделителя используется точка или знак косой черты «/». К тому же первым идёт число, потом месяц и потом год (в англоязычной версии первым может идти месяц). Пробелы не допустимы. Этот порядок и разделитель настраиваются в настройках ОС. Войди в «Панель управления» и запусти окно «Язык и стандарты» (рисунок 14.4.10). Здесь ты можешь изменить все настройки ввода даты, времени и чисел.

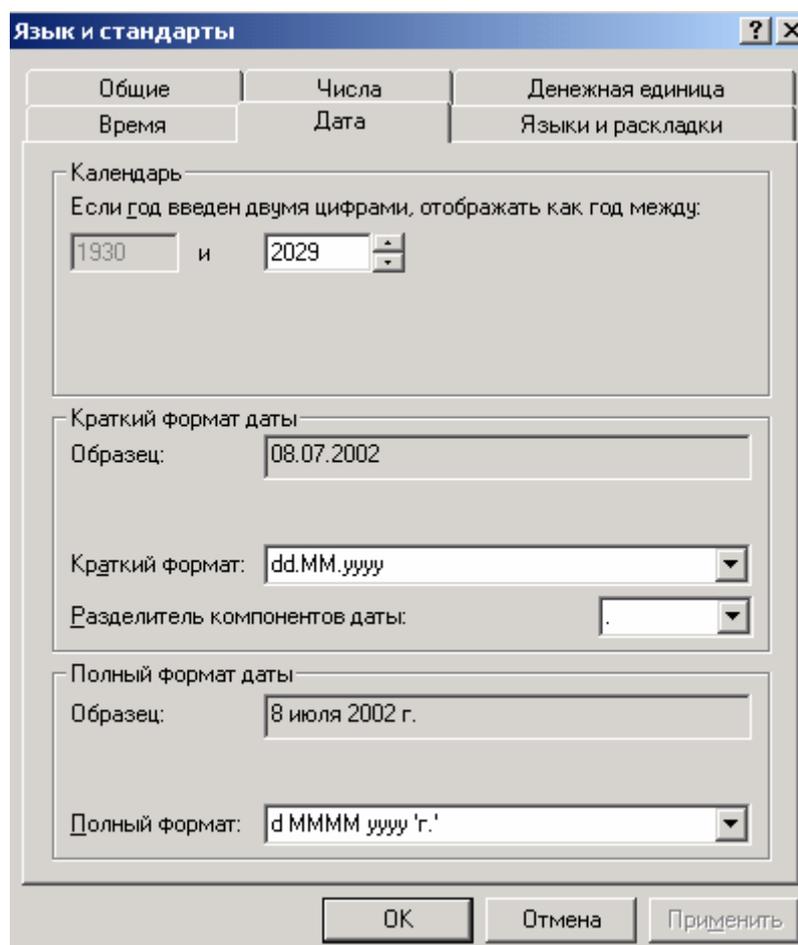


Рис 14.4.10 Настройка формата ввода и отображения даты

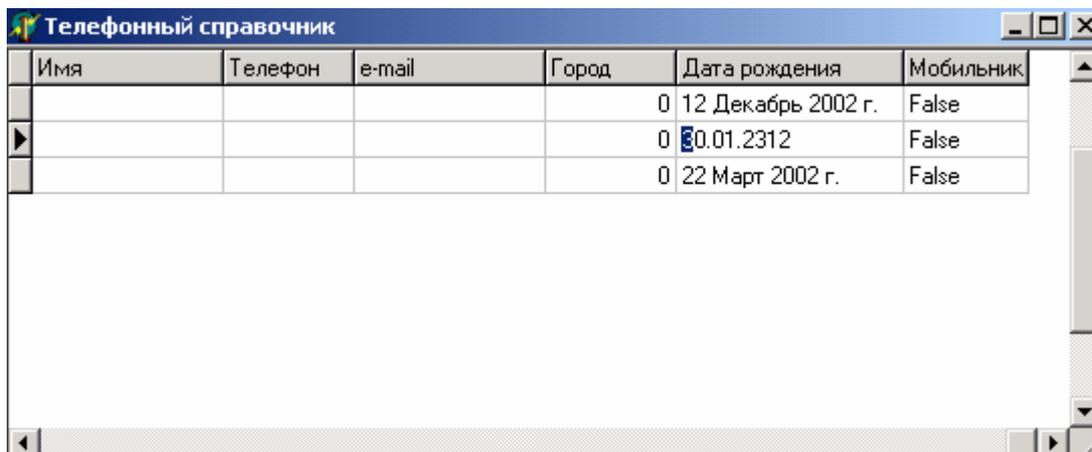
Вернёмся к Delphi. Выдели поле «Дата». Первое, что мы должны сделать – уточнить, какую именно дату здесь надо вводить. Так как это телефонный справочник, то я собираюсь здесь указывать дату рождения владельца телефона. Поэтому, вполне разумно будет в заголовке сетке отображать не просто «Дата», а «Дата рождения». Тут можно поступить двумя способами – отредактировать имя поля в базе данных (не совсем разумно) или просто заставить Delphi отображать в заголовке поля нужный текст.

За текст отображаемый в заголовке отвечает свойство *DisplayLabel* (это свойство есть у всех полей). Давай в нём введём текст «Дата рождения». Можешь проверить, что теперь в заголовке отображается нужный текст.

Теперь отредактируем формат отображения даты. За это отвечает свойство *DisplayFormat*. Тут можно указывать текстовый формат, в котором нужно отображать дату. Как отображать? Вспомни функцию *FormatDateTime* и её первый параметр (см главу 10.5 «Преобразование данных»). Вот именно это здесь и можно указывать. Лично я люблю использовать для отображения полный формат – «dddddd».

Ну и наконец нужно указать маску ввода для даты. Её нужно указывать в свойстве *EditMask* и так же, как мы это делали у компонента *TMaskEdit*. Для даты я всегда указываю маску ввода «99/99/9999».

Если ты теперь запустишь наш пример, то в поле «Дата рождения» все даты будут отображаться в полном формате (рис 14.4.11). Если щёлкнуть дважды по этому полю в любой строке (войти в режим редактирования строки), то дата сразу перейдёт в режим редактирования (см рис 14.4.11 вторая строка).



Имя	Телефон	e-mail	Город	Дата рождения	Мобильник
				0 12 Декабрь 2002 г.	False
				0 30.01.2312	False
				0 22 Март 2002 г.	False

Рис 14.4.11 Улучшенный вид поля «Дата рождения»

Последнее, что нам надо отредактировать – поле «Мобильник». Пока что здесь отображаются значения **true** или **false**. Но мы русские люди и для нас будет удобнее видеть родные **Да** или **Нет**. Выдели это поле и в объектном инспекторе найди свойство *DisplayValues*. Для булевых полей здесь нужно указать два значения в формате «True;False», т.е. сначала указываем положительное значение и после точки с запятой отрицательное (кавычки указывать не надо). Итак, я указал **Да;Нет**.

Теперь в поле «Мобильник» будут отображаться понятные слова, да и при редактировании теперь нужно вводить не **true** или **false**, а родные **Да** или **Нет**.

 На компакт диске, в директории \Примеры\Глава 14\Database2 ты можешь увидеть пример этой программы.