Епин А., Епин Н.

«СОЛОН»

Локальная СУБД своими руками

Учимся на примерах

Пошаговая разработка проектов «с нуля»
Как совместить компактность и универсальность СУБД
Секреты визуального программирования
Как создать тестирующую программу
Анализатор SQL-запросов - сделай сам
Разработка библиотек классов и своего Class Wizard'а

Библиотека Профессионала

УДК 621.396.218
ББК 32.884.1
Е63

А. В. Енин, Н. В. Енин
Е63 Локальная СУБД своими руками. Учимся на примерах. — М.: СОЛОН-ПРЕСС, 2010. — 464 с.: ил. — (Серия «Библиотека профессионала»).

ISBN 5-98003-272-X

— У вас есть множество идей и вы чувствуете в себе силы разработать оригинальную программную систему, но не знаете, с чего начать?

— Вы хотели бы создать свой язык программирования, но полагаете, что это слишком сложно?

— Вас всегда интересовало, как работают различные «волшебники», или «Wizard'ы», автоматически создающие заготовки приложений и частей приложения, но думаете, что создать свой вам не по силам?

— Вы устали от чтения книг со множеством небольших учебных примеров использования классов MFC и хотели бы видеть на практике, как все возможности MFC объединить в настоящий проект?

— Вы прочли ряд книг по проектированию баз данных, но всегда хотели знать, каково их внутреннее устройство?

Эта книга — для вас.

Авторы попытались осветить все вышеназванные вопросы на примере создания локальной системы управления базами данных.

С помощью множества поясняющих схем и ряда работающих проектов, расположенных на прилагаемом к книге CD, шаг за шагом вы, вместе с авторами, создатите ядро СУБД, позволяющее создавать таблицы базы данных и поля в них, добавлять, изменять и удалять записи в этих таблицах, выбирать в таблицах необходимые наборы записей и осуществлять навигацию по этим наборам. В процессе создания ядра вам придётся создать небольшой язык SQL-запросов к базе данных.

Затем, на основе созданного вами кода, будет создана библиотека классов, позволяющая использовать СУБД в объектно-ориентированном виде. И, наконец, будет создана небольшая визуальная среда по работе с СУБД и автоматический генератор C++ классов (Wizard).

Каждому этапу разработки соответствуют один или несколько проектов, т. е. происходит имитация постепенной разработки проекта самим читателем.

УДК 621.396.218
ББК 32.884.1

По вопросам приобретения обращаться:
ООО «АЛЬЯНС-КНИГА КТК»
Тел: (495) 258-91-94, 258-91-95, **www.abook.ru**

Сайт издательства СОЛОН-ПРЕСС: **www.solon-press.ru.**
E-mail: **solon-avtor@coba.ru**

ISBN 5-98003-272-X

Введение

В этой книге мы вместе с вами попробуем написать небольшую, но «настоящую» систему управления базами данных (СУБД). Почему именно эта тема явилась предметом рассмотрения? На то есть целый ряд причин. Существует большое количество литературы, обучающей «проектированию баз данных». Под данным термином подразумевается, что у вас уже есть какая-либо промышленная СУБД и вам необходимо разработать для своих прикладных нужд структуру таблиц и полей базы данных, их взаимосвязь между собой в соответствии с решаемой вами задачей. И действительно, ознакомившись с одной из подобных книг, вы вполне сможете использовать имеющуюся у вас СУБД.

А как быть, если вам захочется узнать, «как это все устроено», каков движущий механизм современных СУБД, какие принципы и алгоритмы обработки информации лежат в их основе, наконец, ознакомиться с примерами программ на подобную тему? Такого типа литературу найти достаточно проблематично, если не сказать невозможно. В лучшем случае необходимая информация будет разбросана по немалому количеству книг, к тому же обычно основной упор в них делается на теоретическую часть проблемы и при попытке самому начать писать что-либо подобное эти книги окажут вам не много помощи.

На то есть свои причины. Во-первых, разработчики ПО не заинтересованы в вашем образовании. Лишние конкуренты им не нужны. Пусть все покупают их продукцию, тратят массу времени на обучение работе с ней, а на большее не рассчитывают. Ведь это так трудно — написать что-то свое, это под силу только большим коллективам гениальных программистов, собранных в фирмах-производителях ПО, а нам всем предлагается пользоваться лишь результатами их нелегкого, недоступного вашему пониманию труда.

Во-вторых, система управления базами данных — действительно достаточно сложный продукт. Сложность эта заключается в том, что при ее разработке используются практически все так называемые современные технологии.

Прежде чем вы напишете первую строку своей программы, вам необходимы некоторые начальные знания по технологиям COM и ATL, протоколу TCP/IP, принципам написания языков программирования и текстовых редакторов, основам устройства файловых систем, знакомство с технологией «клиент — сервер». Можно представить себе то количество специальной литературы, которое необходимо внимательно прочесть в поисках необходимых знаний. Тем более, что большинство авторов не только страдает академической сухостью изложения материала, но и проблемы освещает преимущественно в теоретическом свете, почти не уделяя внимания их практическому применению.

Мы попробуем на практике показать, что создать свою систему с нуля вполне по силам любому мыслящему программисту. И такой системой в нашем случае является СУБД именно по той причине, что в процессе ее создания вы получите практические знания по многим так называемым современным технологиям. Кроме того, есть надежда, что результат нашего труда вы будете использовать в своей повседневной работе при создании небольших приложений.

В настоящее время существует множество СУБД, под разные платформы и операционные системы, различного типа и качества. Судя по публикациям в компьютерной прессе, лучшей считается СУБД Oracle, «мировой лидер в области СУБД», «преимущества которой в полной мере проявляются при количестве записей более ста тысяч», и т. д. Наверное, все это так и есть, и великолепными возможностями Oracle пользуется огромное количество больших фирм, сетевых гигантов, серьезных правительственных и околоправительственных учреждений.

В то же время не совсем понятно, что делать нам, простым пользователям и программистам, пишущим относительно небольшие и далеко не всегда коммерческие программы, которые в процессе своей работы вынуждены использовать возможности СУБД. Использование разрекламированных высокотехнологичных продуктов типа Oracle, SQL-server и др. наталкивается, как минимум, на три проблемы:

- несомненная сложность продукта в эксплуатации и процессе разработки приложений. Как нам кажется, любой выпускаемый в настоящее время продукт (и не только СУБД) в своем развитии проходит три фазы: первоначальная версия, имеющая упрощенные возможности; оптимальная версия, имеющая достаточно богатый набор возможностей и стабильно работающая; версия-переросток, приобретающая массу дополнительных возможностей и, в конце концов, неимоверно раздувающаяся в размерах. К сожалению, большинство фирм не может себе позволить (или не хочет) остановиться на оптимальном варианте продукта.

  В то же время до 70 % разрекламированных возможностей почти никогда не используются большинством разработчиков, что не избавляет вас от необходимости приобретать и изучать толстые книги, описывающие порядок работы с продуктом.

  Универсальность продукта, позволяющая использовать его с разными языками программирования (или вовсе без оных), при всех своих кажущихся достоинствах хороша, на наш взгляд, для успешной продажи продукта на рынке ПО, а в действительности универсальность достигается за счет ограничения предоставляемых продуктом возможностей;
- необходимость наличия используемой СУБД на компьютере пользователя, работающего с созданной вами программой;
- и, наконец, все эти продукты, как правило, небесплатны, что при разработке некоммерческих приложений и приложений для внутреннего использования, как нам кажется, немаловажно.

А ведь существует масса задач, не требующих для своего решения такого развитого инструментария, как промышленные СУБД. Вот мы и займемся созданием своего скромного и полезного продукта.

1. Пример программы-пятиминутки

1.1. Введение

Теперь, когда известна цель проекта, нелишне на каком-нибудь примере показать, что мы собираемся сделать. Сделаем мы это с помощью офисного приложения MS Access. В этом примере мы создадим БД с тремя таблицами, где будут храниться данные о книгах и их авторах. Конечно же, в первой будут храниться данные о книгах, во второй — данные об их авторах, а вот в третьей будут соединены данные из обеих таблиц. Далее мы сформируем проект на основе DAO-классов с помощью Wizard'a. Итак, приступим!

1.2. Создаем базу данных

При запуске Access появится окно, где можно выбрать уже созданную БД или создать новую. Мы создаем новую и называем ее «Test1.mdb». Во вновь появившемся окне щелкаем на кнопке создания новой таблицы в режиме конструктора. В разных версиях Access интерфейс может отличаться, но все же он настолько прост на этапе создания БД и таблиц, что запутаться трудно, даже в первый раз. Первое поле, созданное нами, называется «ID» — оно имеет тип — «счетчик». Счетчик выполняет функцию идентификатора любой созданной записи в этой таблице. Следующие два текстовых поля «Title» и «Category» нужны для хранения названия и категории книги: книга «Мастер и Маргарита» — категория «Классика». В последнем поле будут содержаться рейтинги книг в числовом виде: «5», «30», «217». Щелкаем правой кнопкой мыши на поле «ID» и в появившемся контекстном меню выбираем строку «Ключевое поле». В Access таблица не может существовать без ключевого поля. После закрытия окна конструктора Access запрашивает имя таблицы, и мы вводим название — «Книга». Аналогично создаем таблицу «Автор» с полями: идентификатор — «ID», имя автора — «Name», отчество — «OldName», фамилия — «Family». Например, автор «Михаил» «Афанасьевич» «Булгаков». В последней таблице «Соединение автора с книгой», объединяющей две остальных, будет всего три поля: идентификатор — «ID»; число, соответствующее идентификатору книги в первой таблице — «BookID»; число, соответствующее идентификатору автора во второй таблице — «AuthorID». В этом случае налицо связь первой и второй таблицы через третью, что показано на схеме. Далее заполняем таблицы авторов и книг произвольными значениями и сопоставляем значения их идентификаторов в таблице «Соединение автора с книгой».

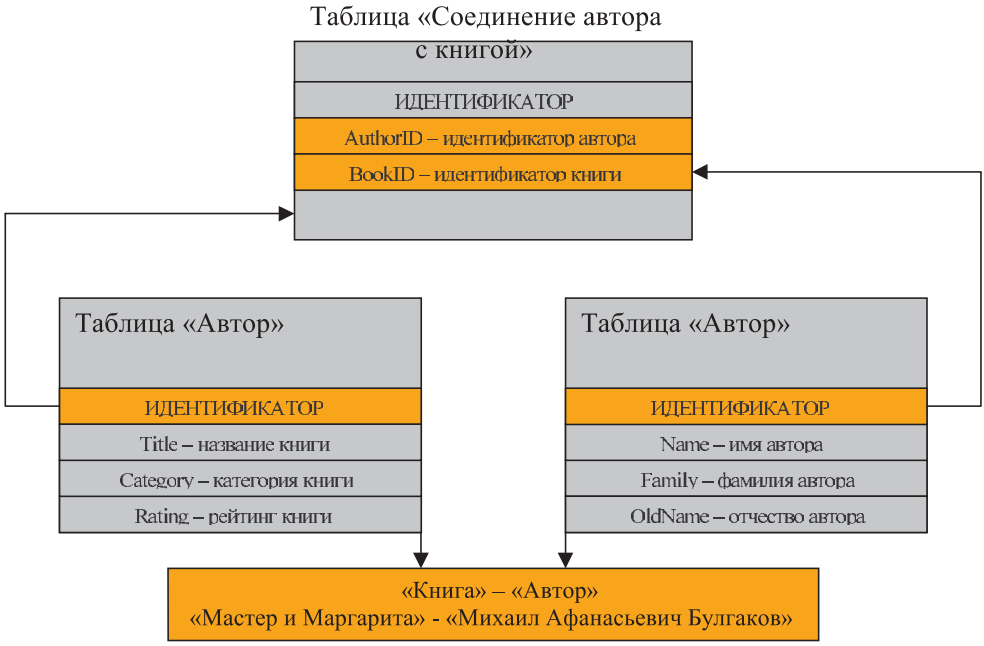

Схема 1.1. Связь двух таблиц через третью

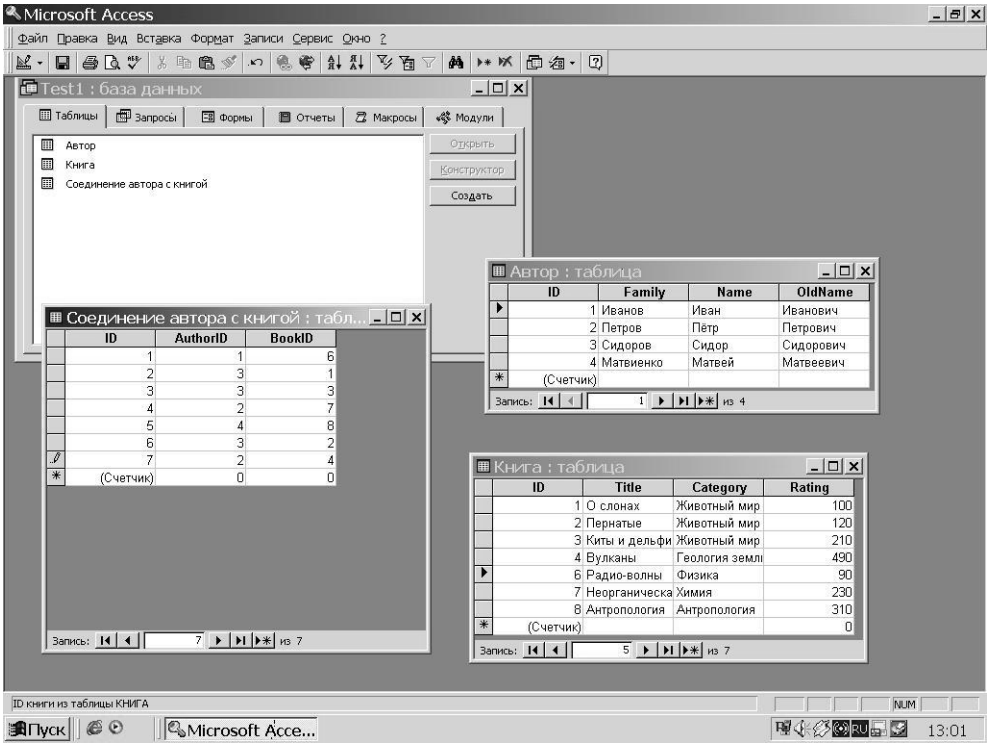

Рис. 1.1

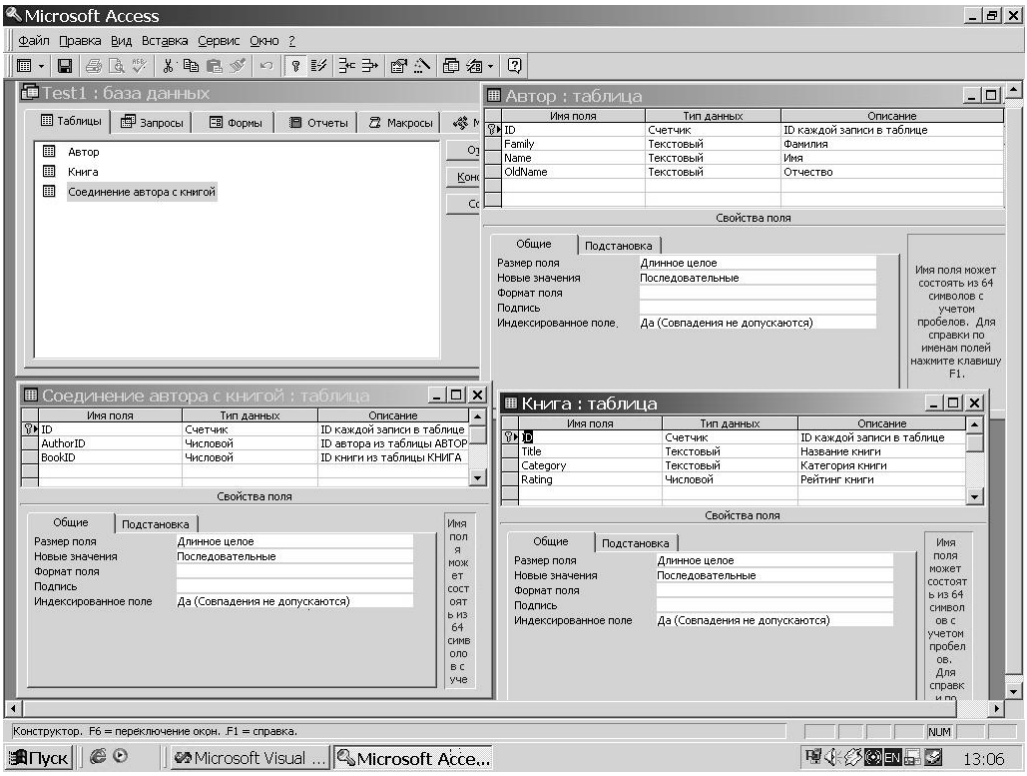

Рис. 1.2

1.3. Создание проекта

Теперь запускаем MS Visual C++. Чтобы создать новый проект, необходимо, как и в других приложениях Microsoft, выбрать в меню «File» пункт «New...». В появившемся диалоговом окне в поле редактирования «Project name» набираем имя нашего проекта: «TestDAO». В списке выделяем «MFC AppWizard (exe)». В новом окне с заголовком «MFC AppWizard — Step 1» выделяем вместо «Multiple document» «Single document» и нажимаем кнопку «Next». В следующем окне выделяем вместо «None» — «Database view without file support» и щелкаем на кнопке «Data Source...». В новом окне выбираем «DAO», после чего нажимаем на кнопку, рядом с комбинированным списком, с названием «...». При этом появится окно открытия БД. Мы указываем в нем путь к нашей базе данных «Test1.mdb» и нажимаем кнопку «OK». После чего появляется окно выбора таблицы, на основе которой будет создано приложение: к примеру, таблицу «Автор». Теперь для завершения работы Wizard'a нажимаем кнопку «Finish».

По завершении работы Wizard'a у нас имеется рабочее приложение на основе базы данных. Для того чтобы проверить, что это приложение действительно работает, нужно в меню «Build» выбрать подменю «Execute TestDAO.exe», после чего появится диалоговое приложение.

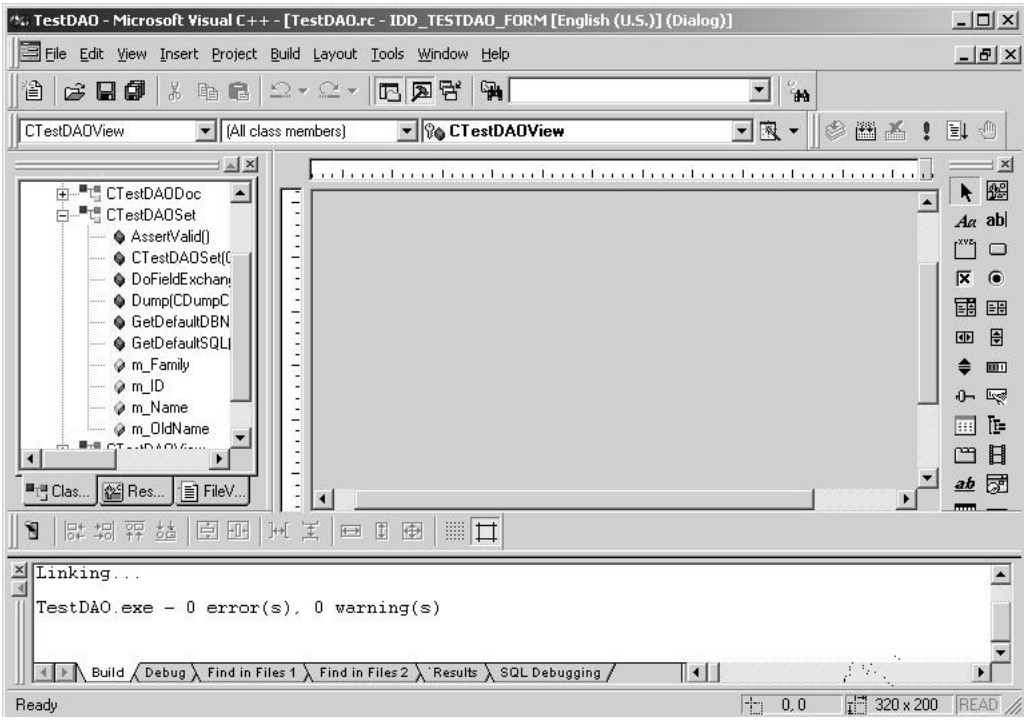
TestDAO - Microsoft Visual C++ - [TestDAO.rc - IDD_TESTDAO_FORM [English (U.S.)] (Dialog)]
File Edit View Insert Project Build Layout Tools Window Help
CTestDAOView
(All class members)
CTestDAOView
CTestDAODoc
CTestDAOSet
AssertValid()
m_Family
m_ID
m_Name
m_OldName
Clas...
Res...
FileV...
Linking...
TestDAO.exe - 0 error(s), 0 warning(s)
Build
Debug
Find in Files 1
Find in Files 2
Results
SQL Debugging
Ready
0, 0
320 x 200
READ

Рис. 1.3

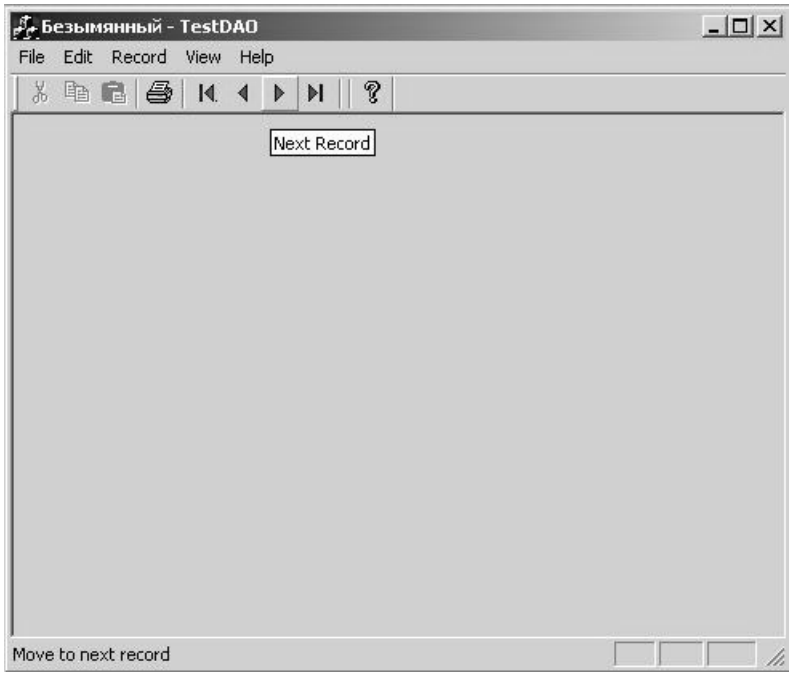
Безымянный - TestDAO
File Edit Record View Help
Next Record
Move to next record

Рис. 1.4

Результатом наших действий является приложение, которое может брать из БД различные данные в виде записей, но они еще не выводятся на экран и поэтому нельзя оценить работу, которую сделал за нас Wizard.

Чуть позже мы рассмотрим, как можно визуализировать работу нашей программы с базой данных, а пока неплохо было бы узнать, что именно сделал для нас Wizard.

В окне «Workspace», которое располагается слева от рабочей области приложения Visual C++, во вкладке «ClassView» находится древовидный список. Открываем в нем класс CTestDAOSet, там находятся переменные, соответствующие полям таблицы «Автор»: ID — m_ID, Family — m_Family, Name — m_Name, OldName — m_OldName. В самом же классе и его предках находятся функции, осуществляющие интерфейс между приложением и базой данных. Заполняя поля и вызывая соответствующие функции, можно добавлять, изменять и удалять записи из таблицы «Автор».

Приложение, которое мы создали, базируется на DAO-классах. Но что же представляет собой «DAO»?

DAO — Data Acess Objects — объекты доступа к базам данных. Объектно-ориентированный интерфейс для ядра управления базами данных Microsoft Jet. С Версии Office 95 Jet стал Automation и поэтому может выполняться в любой программе, которая является клиентом Automation.

В Office 97 включена версия DAO 3.5. Она позволяет соединяться с источником данных, используя ODBC через ODBCDirect. Это позволяет, работая со стандартным интерфейсом, иметь доступ не только к базам данных Access (.mdb), но и к базам данных ISAM и SQL. В итоге DAO — универсальный механизм доступа к данным.

Что это дает? Ну, например, так как доступ универсальный, разработчик имеет возможность тестировать приложения, используя Access без загрузки SQL-сервера, а установить соединение с сервером только после того, как приложение будет готово и протестировано. Также DAO позволяет работать с SQL-серверами таким распространенным офисным приложениям, как Excel и Word. Кроме того, есть возможность работать по технологии баз данных с форматами файлов, которые не рассматривались как БД, например, текстовые файлы или листы Excel.

В DAO 3.5 две объектных модели. Модель зависит от того, используется ODBC или нет. Модель задается при создании соединения (рабочей области) указанием констант.

Для нас уже создано шесть классов, каждый из которых выполняет свои функции. Класс CAboutDlg выводит окно помощи, класс CMainFrame создает главное окно, класс CTestDAOApp отвечает за приложение, класс CTestDAODoc работает с документом (рабочей областью главного окна), класс CTestDAOSet работает с таблицей «Автор» из базы данных «Test1.mdb», класс CTestDAOView отвечает за визуальное отображение элементов приложения. Подробнее мы рассмотрим их дальше, когда будем создавать тестирующую программу для начальных этапов разработки будущего проекта.

1.4. Первичный анализ и добавление классов

Теперь изменим исходную программу так, чтобы мы смогли просматривать записи. Более широко об этом будет рассказано в других главах.

Создание переменных и определение функций происходит в *TestDAOSet.h*, т.н. файле-заголовке (file header). Конструктор класса по умолчанию — *CTestDAOSet()*, в нем происходит инициализация переменных и объектов, а также зачастую выделение памяти. А освобождается память и удаляются объекты в деструкторе, он обозначается как *~CTestDAOSet()*. Функция *GetDefaultDBName()* возвращает полное имя базы данных, т. е. путь к ней. Функция, как и все остальные, создана Wizard'ом по умолчанию. Функция *GetDefaultSQL()* возвращает имя таблицы, на основе которой создан класс. Функция *DoFieldExchange()* как раз и производит обмен данными между базой данных и переменными класса таблицы. Это происходит посредством работы макросов DFX, которые получают на входе имена полей и имена соответствующих им переменных.

Листинг 1.1. Класс CTestDAOSet

```
// TestDAOSet.h : interface of the CTestDAOSet class
      long        m_ID;       // Переменные, соответствующие полям
      CString     m_Family;   // в базе данных
      CString     m_Name;
      CString     m_OldName;


// TestDAOSet.cpp : implementation of the CTestDAOSet class
CTestDAOSet::CTestDAOSet(CDaoDatabase* pdb)
      : CDaoRecordset(pdb)
{
      //{{AFX_FIELD_INIT(CTestDAOSet)
      m_ID = 0;
      m_Family = _T("");
      m_Name = _T("");      // Инициализация буферных переменных
      m_OldName = _T("");   // для данных полей записи БД
      m_nFields = 4;
      //}}AFX_FIELD_INIT
      m_nDefaultType = dbOpenDynaset;
}

CString CTestDAOSet::GetDefaultDBName()
{
      return _T("C:\\Documents and Settings\\Администратор\\Мои документы\\Test1.mdb");
}

CString CTestDAOSet::GetDefaultSQL()
{
      return _T("[Автор]"); // Возврат имени таблицы
}

void CTestDAOSet::DoFieldExchange(CDaoFieldExchange* pFX)
{
 // Полю сопоставляется переменная класса, для обмена данными
      //{{AFX_FIELD_MAP(CTestDAOSet)
```

```
      pFX->SetFieldType(CDaoFieldExchange::outputColumn);
      DFX_Long(pFX, _T("[ID]"), m_ID);
      DFX_Text(pFX, _T("[Family]"), m_Family);
      DFX_Text(pFX, _T("[Name]"), m_Name);
      DFX_Text(pFX, _T("[OldName]"), m_OldName);
      //}}AFX_FIELD_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CTestDAOSet diagnostics

#ifdef _DEBUG
void CTestDAOSet::AssertValid() const
{
      CDaoRecordset::AssertValid(); // Вызов функции предка
}

void CTestDAOSet::Dump(CDumpContext& dc) const
{
      CDaoRecordset::Dump(dc); // Вызов функции предка
}
#endif //_DEBUG
```

Кроме того, как мы уже видели, можно ходить по записям при помощи кнопок на панели инструментов или выбирая соответствующие им пункты меню. Так вызываются функции класса CDaoRecordset: MoveFirst — переход в начало, MoveLast — переход в конец, MoveNext — переход на следующую запись, MovePrev — переход на предыдущую.

А теперь визуализируем нашу программу. Для начала необходимо создать еще два класса на основе других таблиц и являющиеся потомками класса CDaoRecordset. Для этого нужно в окне проекта щелкнуть на заголовке «Test-DAO classes» и в появившемся контекстном меню выбрать пункт «New Class...», после чего возникнет окно создания нового класса (рис. 1.5).

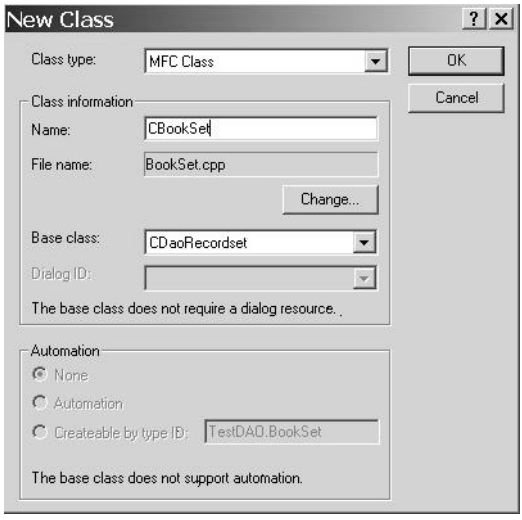

Рис. 1.5

В поле «Name» этого окна следует ввести название класса «CBookSet», а в поле «File name» автоматически появляется имя файла, в котором, собственно, и будет находиться код класса «BookSet.cpp». В комбинированном списке «Base class» выбираем класс «CDaoRecordset». После нажатия кнопки «OK» нужно повторить действия по указанию пути к БД и выбора таблицы, только вместо таблицы «Автор» выбираем таблицу «Книги».

Перед добавлением элементов в окно IDD_TESTDAO_FORM (кнопок, полей редактирования, надписей) нужно его настроить на русский язык, иначе потом в окне вместо надписей и названий кнопок будут выводиться иероглифы сомнительного назначения. Для этого в окне проекта во вкладке «ResourseView» находим заготовку окна IDD_TESTDAO_FORM, щелкаем на нем и в его контекстном меню выбираем пункт «Properties». И в окне, показанном на рис. 1.6, в комбинированном списке меняем английский язык на русский.

Теперь мы изменяем в редакторе ресурсов форму «IDD_TESTDAO_FORM» в соответствии с рис. 1.7.

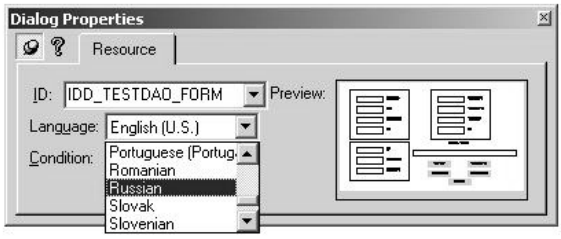

Рис. 1.6

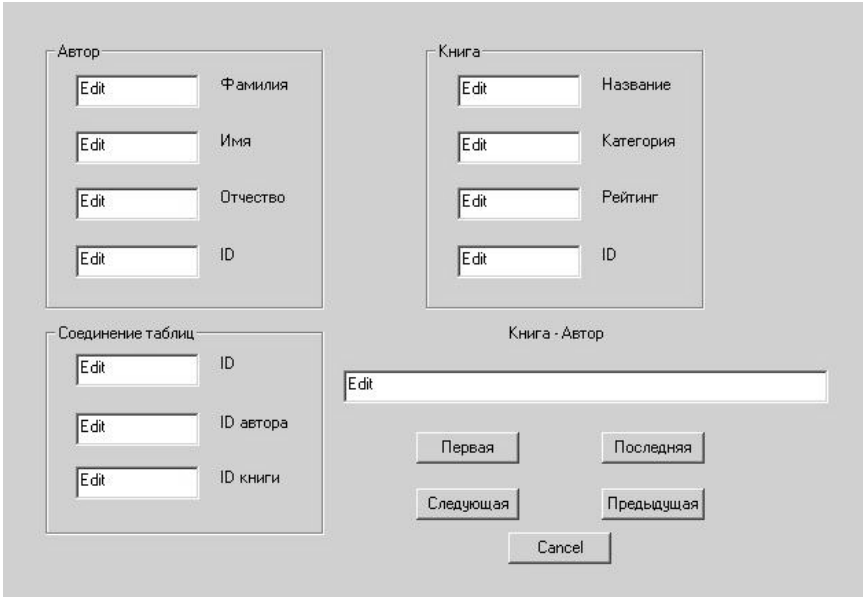

Рис. 1.7

После добавления всех элементов нужно изменить идентификаторы полей и кнопок окна, а также создать для полей редактирования переменные в классе CTestDAOView. Для каждого элемента окна, кроме надписей, в окне «Properties» изменяем поле «ID» на соответствующий по значению идентификатор. Для создания переменных элементов окна надо щелкнуть на одном из них правой кнопкой мыши и выбрать в контекстном меню пункт «ClassWizard...». Во вкладке «Member Variables» появившегося окна выделяем идентификатор поля редактирования, нажимаем кнопку «Add Variable...» и вводим имя новой переменной. Ниже приведен список названий полей редактирования, их идентификаторы и соответствующие им имена переменных.

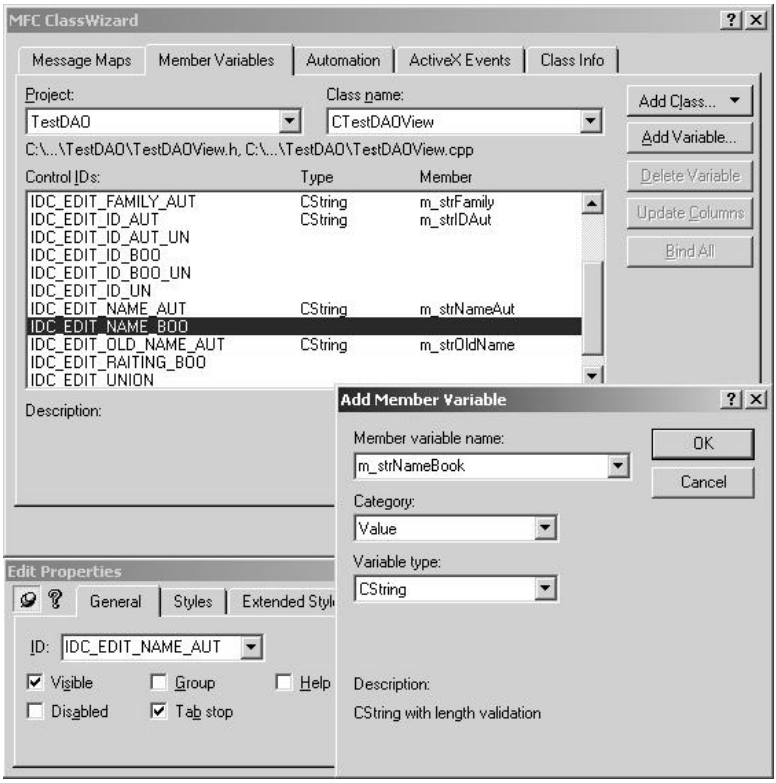

Рис. 1.8

Таблица 1.1. Идентификаторы и переменные

Таблица «АВТОР»		
Фамилия	IDC_EDIT_FAMILY_AUT	m_strFamilyAut
Имя	IDC_EDIT_NAME_AUT	m_strNameAut
Отчество	IDC_EDIT_OLD_NAME_AUT	m_strOldNameAut
Идентификатор	IDC_EDIT_ID_BOOK	m_strIDAut

Окончание табл. 1.1

Таблица «КНИГА»		
Название	IDC_EDIT_NAME_BOOK	m_strNameBook
Категория	IDC_EDIT_CATEGOR_BOOK	m_strCategoryBook
Рейтинг	IDC_EDIT_RAITING_BOOK	m_strIDBook
Идентификатор	IDC_EDIT_ID_BOOK	m_strIDBook
Таблица «СОЕДИНЕНИЕ ТАБЛИЦ»		
Идентификатор	IDC_EDIT_ID_UN	m_strIDUn
Идент. Автора	IDC_EDIT_ID_AUT_UN	m_strIDAutUn
Идент. Книги	IDC_EDIT_ID_BOO_UN	m_strIDBookUn
Книга — Автор	IDC_EDIT_UNION	m_strIDBookUn
КНОПКИ УПРАВЛЕНИЯ		
Первая	IDC_BUT_FIRST	
Последняя	IDC_BUT_LAST	
Следующая	IDC_BUT_NEXT	
Предыдущая	IDC_BUT_PREV	

1.5. Функция OnRecordFirst()

Дальше нам надо добавить обработчики сообщений от кнопок передвижения по записям. Делаем мы это с помощью ClassWizard'a, как показано на рис. 1.9.

Теперь создаем код этих четырех функций.

Все четыре функции очень схожи текстом. Мы разберем одну из них, а в остальных выделим отличия кода. Во-первых, в класс *CTESTDAOView* добавляем переменную *IDUnion*. Важно заметить, что эта переменная не обязательно должна совпадать с подлинным идентификатором активной записи.

Рассмотрим первую из них — функцию OnRecordFirst(). Она перемещает активную запись в первую позицию.

Действия функции:

1. Присваиваем переменной IDUnion значение 1, так как мы хотим попасть на первую запись таблицы.

2. Присваиваем переменной фильтрации запрос «ID = 1».

3. Открываем таблицу, при этом и происходит фильтрация записей. Алгоритм открытия довольно прост: если таблица уже открыта (*IsOpen()*), то открываем ее заново (*Requery()*), иначе открываем (*Open()*). Но это необходимо, так как повторное открытие уже открытой таблицы приводит к ошибке.

4. Переходим на первую запись функцией *MoveFirst()*.

5. Проверяем, есть ли в таблице хоть одна запись (*IsEOF()*).

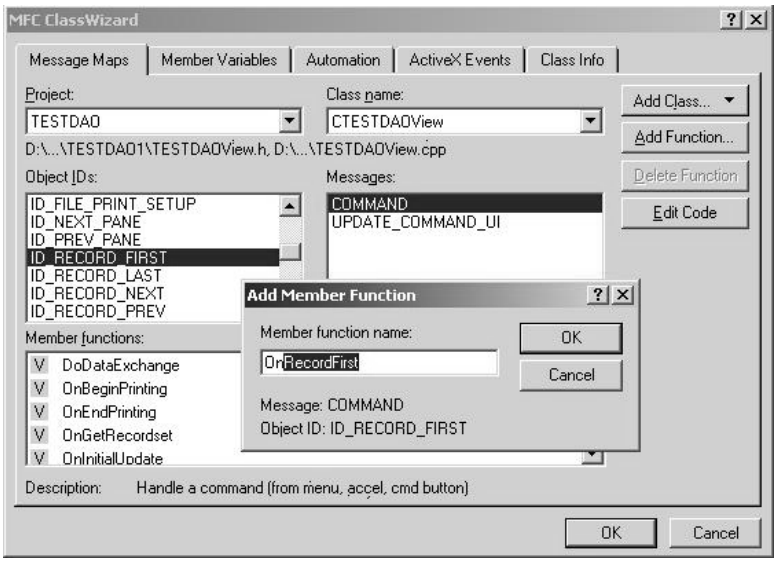

Рис. 1.9

6. По идентификатору из соединяющей таблицы фильтруем таблицу авторов.

7. Открываем таблицу авторов. Результатом открытия должна являться виртуальная таблица с одной искомой записью (если она есть), так как идентификатор уникален (является счетчиком).

8. Проверяем, есть ли в таблице хоть одна запись.

9. Присваиваем переменным полей редактирования нашего окна соответствующие данные из таблицы авторов.

10. Фильтруем таблицу книг по идентификатору из соединительной таблицы.

11. Открываем таблицу книги и получаем искомую запись (если она есть).

12. Если эта запись существует, присваиваем переменным полей редактирования окна соответствующие данные из таблиц соединения и книг.

13. Соединяем имя автора и название книги и тоже выводим в окне.

14. Вызываем функцию *UpdateData()* с параметром *FALSE*, которая обновляет содержимое полей редактирования на экране. Если заменить *FALSE* на *TRUE*, то произойдет обратный процесс — извлечение данных из полей редактирования.

Листинг 1.2. Функция OnRecordFirst()

```
void CTESTDAOView::OnRecordFirst()
{
      IDUnion = 1; // Инициализация переменной

      m_pSet->m_strFilter.Format("ID = %d", IDUnion); // SQL-запрос

      if( m_pSet->IsOpen()) // Открываем таблицу соединения
            m_pSet->Requery();
      else
```

```
            m_pSet->Open();

      if( !m_pSet->IsEOF()) // проверяем найдена ли искомая запись
      {
// Набиваем SQL-запрос для таблицы авторов
            author.m_strFilter.Format("ID = %d", m_pSet->m_AuthorID);

            if( author.IsOpen()) // Открываем таблицу авторов
                  author.Requery();
            else
                  author.Open();

            if( !author.IsEOF()) // Проверяем найдена ли запись
            {
// Выводим в поля редактирования значения полей записи
                  m_strIDAut.Format("%d", author.m_ID);
                  m_strFamilyAut = author.m_Family;
                  m_strNameAut = author.m_Name;
                  m_strOldNameAut = author.m_OldName;
            }

// Набиваем SQL-запрос для таблицы книг
            book.m_strFilter.Format("ID = %d", m_pSet->m_BookID);

            if( book.IsOpen()) // Открываем таблицу книг
                  book.Requery();
            else
                  book.Open();

            if( !book.IsEOF()) // Проверяем найдена ли запись
            {
// Выводим в поля редактирования значения полей записи
            m_strIDBook.Format("%d", book.m_ID);
                  m_strNameBook = book.m_Title;
                  m_strCategoryBook = book.m_Category;
                  m_strRaitingBook.Format("%d", book.m_Rating);

                  m_strIDUn.Format("%d", m_pSet->m_ID);
                  m_strIDBookUn.Format("%d", book.m_ID);
                  m_strIDAutUn.Format("%d", author.m_ID);

            }
// Набиваем строку "Автор - Книга"
            m_strUnion = book.m_Title + "  -  " + author.m_Family;
            UpdateData( FALSE); // Обновляем диалоговое окно
      }
}
```

1.6. Функция OnRecordLast()

Следующая функция переходит на последнюю запись.

Действия функции:

1. Определяем переменную *i* и инициируем ее значением 1.
2. Открываем таблицу, при этом и происходит фильтрация записей.

3. По циклу *while* находим количество записей.

4. Так как в данном случае количество записей в таблице совпадает с идентификатором последней записи, сохраняем это значение в переменной *IDUion*.

5. Для избежания ошибок вызываем функцию перехода на последнюю запись *MoveLast()*.

6. Открываем таблицу авторов. Результатом открытия должна являться виртуальная таблица с одной искомой записью (если она есть), так как идентификатор уникален (является счетчиком).

7. Проверяем, есть ли в таблице хоть одна запись.

8. Присваиваем переменным полей редактирования нашего окна соответствующие данные из таблицы авторов.

9. Фильтруем таблицу книг по идентификатору из соединительной таблицы.

10. Открываем таблицу книги и получаем искомую запись (если она есть).

11. Если эта запись существует, присваиваем переменным полей редактирования окна соответствующие данные из таблиц соединения и книг.

12. Соединяем имя автора и название книги и тоже выводим в окне.

13. Вызываем функцию *UpdateData()* с параметром *FALSE*, которая обновляет содержимое полей редактирования на экране. Если заменить *FALSE* на *TRUE*, то произойдет обратный процесс — извлечение данных из полей редактирования.

Листинг 1.3. Функция OnRecordLast()

```
void CTESTDAOView::OnRecordLast()
{
      int i = 0; // Инициализация переменной

      m_pSet->m_strFilter = ""; // Пустой SQL-запрос

      if( m_pSet->IsOpen()) // Открываем таблицу соединения
            m_pSet->Requery();
      else
            m_pSet->Open();

            while( !m_pSet->IsEOF()) // По циклу ищем последнюю запись
            {
                  i++; // Инкрементация переменной
                  m_pSet->MoveNext(); // Переход на следующую запись
            }

            IDUnion = i; // Сохраняем позицию текущей записи

            m_pSet->MoveLast(); // Переходим на последнюю запись

      // Набиваем SQL-запрос для таблицы авторов
      author.m_strFilter.Format("ID = %d", m_pSet->m_AuthorID);

            if( author.IsOpen()) // Открываем таблицу авторов
                  author.Requery();
```

```
            else
                  author.Open();

            if( !author.IsEOF()) // Проверяем найдена ли запись
            {
// Выводим в поля редактирования значения полей записи
                  m_strIDAut.Format("%d", author.m_ID);
                  m_strFamilyAut = author.m_Family;
                  m_strNameAut = author.m_Name;
                  m_strOldNameAut = author.m_OldName;
            }

// Набиваем SQL-запрос для таблицы книг
            book.m_strFilter.Format("ID = %d", m_pSet->m_BookID);

            if( book.IsOpen()) // Открываем таблицу книг
                  book.Requery();
            else
                  book.Open();

            if( !book.IsEOF()) // Проверяем найдена ли запись
            {
// Выводим в поля редактирования значения полей записи
            m_strIDBook.Format("%d", book.m_ID);
                  m_strNameBook = book.m_Title;
                  m_strCategoryBook = book.m_Category;
                  m_strRaitingBook.Format("%d", book.m_Rating);

                  m_strIDUn.Format("%d", m_pSet->m_ID);
                  m_strIDBookUn.Format("%d", book.m_ID);
                  m_strIDAutUn.Format("%d", author.m_ID);

            }
// Набиваем строку "Автор - Книга"
            m_strUnion = book.m_Title + "  -  " + author.m_Family;
            UpdateData( FALSE); // Обновляем диалоговое окно

}
```

1.7. Функция OnRecordNext()

Функция OnRecordNext() переходит на следующую запись в таблице.

Действия функции:

1. Итерируем переменную *IDUnion*.

2. Открываем соединяющую таблицу, в параметры фильтрации мы вводим идентификатор следующей записи.

3. Проверяем, есть ли в таблице хоть одна запись (*IsEOF()*).

4. По идентификатору из соединяющей таблицы фильтруем таблицу авторов.

5. Открываем таблицу авторов. Результатом открытия должна являться виртуальная таблица с одной искомой записью (если она есть), так как идентификатор уникален (является счетчиком).

6. Проверяем, есть ли в таблице хоть одна запись.

7. Присваиваем переменным полей редактирования нашего окна соответствующие данные из таблицы авторов.

8. Фильтруем таблицу книг по идентификатору из соединительной таблицы.

9. Открываем таблицу книги и получаем искомую запись (если она есть).

10. Если эта запись существует, присваиваем переменным полей редактирования окна соответствующие данные из таблиц соединения и книг.

11. Соединяем имя автора и название книги и тоже выводим в окне.

12. Вызываем функцию *UpdateData()* с параметром *FALSE*, которая обновляет содержимое полей редактирования на экране. Если заменить *FALSE* на *TRUE*, то произойдет обратный процесс — извлечение данных из полей редактирования.

13. Если в виртуальной таблице после фильтрования нет ни одной записи, то *IDUnion* уменьшаем на единицу.

Листинг 1.4. Функция OnRecordNext()

```
void CTESTDAOView::OnRecordNext()
{
      IDUnionxx; // инкрементируем переменную позиции

      if( IDUnion > 0) // Проверяем, не выходи ли он за пределы
      {
// Набиваем строку запроса таблицы соединения
            m_pSet->m_strFilter.Format("ID = %d", IDUnion);

            if( m_pSet->IsOpen()) // Открываем таблицу соединения
                  m_pSet->Requery();
            else
                  m_pSet->Open();

            if( !m_pSet->IsEOF()) // Проверяем на выход за пределы таблицы
            {
// Набиваем SQL-запрос для таблицы авторов
                  author.m_strFilter.Format("ID = %d", m_pSet->m_AuthorID);

                  if( author.IsOpen()) // Открываем таблицу авторов
                        author.Requery();
                  else
                        author.Open();

                  if( !author.IsEOF()) // Проверяем найдена ли запись
                  {
// Выводим в поля редактирования значения полей записи
                        m_strIDAut.Format("%d", author.m_ID);
                        m_strFamilyAut = author.m_Family;
                        m_strNameAut = author.m_Name;
                        m_strOldNameAut = author.m_OldName;
                  }

// Набиваем SQL-запрос для таблицы книг
                  book.m_strFilter.Format("ID = %d", m_pSet->m_BookID);
```

```
                if( book.IsOpen()) // Открываем таблицу книг
                      book.Requery();
                else
                      book.Open();

                if( !book.IsEOF()) // Проверяем найдена ли запись
                {
// Выводим в поля редактирования значения полей записи
                      m_strIDBook.Format("%d", book.m_ID);
                      m_strNameBook = book.m_Title;
                      m_strCategoryBook = book.m_Category;
                      m_strRaitingBook.Format("%d", book.m_Rating);

                      m_strIDUn.Format("%d", m_pSet->m_ID);
                      m_strIDBookUn.Format("%d", book.m_ID);
                      m_strIDAutUn.Format("%d", author.m_ID);

                }
// Набиваем строку "Автор - Книга"
                m_strUnion = book.m_Title × "  -  " × author.m_Family;
                UpdateData( FALSE); // Обновляем диалоговое окно
          }
          else
                IDUnion--; // При неполадке возвращаем старое значение
     }
}
```

1.8. Функция OnRecordPrev()

И последняя функция — OnRecordPrev() — переходит на предыдущую запись.

Действия функции:

1. Если *IDUnion* больше единицы, уменьшаем ее значение на 1.
2. Открываем соединяющую таблицу, в параметры фильтрации мы вводим идентификатор предыдущей записи.
3. Открываем таблицу, при этом и происходит фильтрация записей.
4. Проверяем, есть ли в таблице хоть одна запись (*IsEOF()*).
5. По идентификатору из соединяющей таблицы фильтруем таблицу авторов.
6. Открываем таблицу авторов. Результатом открытия должна являться виртуальная таблица с одной искомой записью (если она есть), так как идентификатор уникален (является счетчиком).
7. Проверяем, есть ли в таблице хоть одна запись.
8. Присваиваем переменным полей редактирования нашего окна соответствующие данные из таблицы авторов.
9. Фильтруем таблицу книг по идентификатору из соединительной таблицы.
10. Открываем таблицу книги и получаем искомую запись (если она есть).
11. Если эта запись существует, присваиваем переменным полей редактирования окна соответствующие данные из таблиц соединения и книг.

12. Соединяем имя автора и название книги и тоже выводим в окне.

13. Вызываем функцию *UpdateData()* с параметром *FALSE*, которая обновляет содержимое полей редактирования на экране. Если заменить *FALSE* на *TRUE*, то произойдет обратный процесс — извлечение данных из полей редактирования.

Листинг 1.5. Функция OnRecordPrev()

```
void CTESTDAOView::OnRecordPrev()
{
      if( IDUnion > 1) // Если номер больше единицы,
            IDUnion--; // то декрементируем переменную
      else
            return; // Иначе выходим из функции

// Набиваем строку запроса
            m_pSet->m_strFilter.Format("ID = %d", IDUnion);

            if( m_pSet->IsOpen()) // Открываем таблицу соединения
                  m_pSet->Requery();
            else
                  m_pSet->Open();

            if( !m_pSet->IsEOF()) // Проверяем на выход за пределы таблицы
            {
// Набиваем SQL-запрос для таблицы авторов
                  author.m_strFilter.Format("ID = %d", m_pSet->m_AuthorID);

                  if( author.IsOpen()) // Открываем таблицу авторов
                        author.Requery();
                  else
                        author.Open();

                  if( !author.IsEOF()) // Проверяем найдена ли запись
                  {
// Выводим в поля редактирования значения полей записи
                        m_strIDAut.Format("%d", author.m_ID);
                        m_strFamilyAut = author.m_Family;
                        m_strNameAut = author.m_Name;
                        m_strOldNameAut = author.m_OldName;
                  }

// Набиваем SQL-запрос для таблицы книг
                  book.m_strFilter.Format("ID = %d", m_pSet->m_BookID);

                  if( book.IsOpen()) // Открываем таблицу книг
                        book.Requery();
                  else
                        book.Open();

                  if( !book.IsEOF()) // Проверяем найдена ли запись
                  {
// Выводим в поля редактирования значения полей записи
                        m_strIDBook.Format("%d", book.m_ID);
                        m_strNameBook = book.m_Title;
```

```
                    m_strCategoryBook = book.m_Category;
                    m_strRaitingBook.Format("%d", book.m_Rating);

                    m_strIDUn.Format("%d", m_pSet->m_ID);
                    m_strIDBookUn.Format("%d", book.m_ID);
                    m_strIDAutUn.Format("%d", author.m_ID);

                }
// Набиваем строку "Автор - Книга"
                m_strUnion = book.m_Title + "  -  " + author.m_Family;
                UpdateData( FALSE); // Обновляем диалоговое окно
            }
}
```

1.9. Заключение

Теперь запустим приложение. Сразу же мы увидим содержимое первой записи. Можно ходить по записям и просматривать их содержимое. Конечно, это слишком упрощенная программа. В ней мало проверок на крайние значения, и она не выдержит даже первичного тестирования. Если удалить одну запись, то программа будет себя вести неадекватно. Но нашей задачей было показать на примере, что же мы собираемся создать в нашей книге далее, а именно — приложение-менеджер БД, иерархию классов работы с БД для MS Visual C++. Пока что не будем углубляться в подробности нашего будущего проекта, о нем мы поговорим немного позже.

Оглавление