Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования
ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

В. Б. Пикулев

Разработка баз данных.
Практика Microsoft Access & VB.NET

Учебное пособие

Петрозаводск
Издательство ПетрГУ
2010

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

В настоящее время информация стала фактором, определяющим эффективность любой сферы человеческой жизнедеятельности. И умение её хранить, систематизировать и использовать становится определяющим элементом в любом процессе принятия решения: от мирового и государственного управления и стратегического планирования до ведения личных дел.

Под термином *база данных* (database) обычно понимают набор логически согласованных между собой данных вместе с описанием их структуры, содержащий сведения о конкретной предметной области. В эпоху электронных средств хранения и обработки информации термин «базы данных» неразрывно связан с понятием *СУБД*, т.е. с *системой управления базами данных* (DBMS – Database Management System).

СУБД – это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии, поиска в них информации и осуществления к этой информации контролируемого доступа. Основными функциями СУБД, таким образом, являются обеспечение целостности и непротиворечивости данных в базе, предотвращение несанкционированного доступа к информации, организация эффективной (параллельной многопользовательской) работы с данными (см.: [1, 2]).

История развития баз данных начинается с 70-х годов XX века, когда разработанные схемы хранения и управления массивами данных на «больших машинах» IBM начали отличаться от понятия «файловая система». Существовавшие при этом модели представления данных были весьма интересными, но не до конца проработанными.

Основоположником теории реляционных баз данных считается сотрудник фирмы IBM доктор Э. Кодд (Codd), опубликовавший в 1970 году статью «A Relational Model of Data for Large Shared Data Banks». Кодд предложил использовать для обработки данных аппарат теории множеств и предикативной логики для того, чтобы внести в область управления базами данных строгие математические принципы. Сейчас реляционные СУБД используются практически повсеместно, занимая более 90% рынка всех систем баз данных [3].

В прикладном смысле *реляционной* считается такая база данных, в которой все данные представлены для пользователя в виде прямоугольных таблиц, называемых *отношениями* (relation), и все операции над базой данных сводятся к манипуляциям с таблицами. В более строгом изложении реляционная модель определяется тремя частями, описывающими разные аспекты реляционного подхода: структурной, манипуляционной и целостной.

В структурной части модели фиксируется, что единственной структурой данных, используемой в реляционных БД, является *нормализованное отношение*[1]. В манипуляционной части модели утверждаются два фундаментальных механизма манипулирования реляционными БД: *реляционная алгебра* и *реляционное исчисление*. В целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД: *требование целостности сущностей* и *требование целостности по ссылкам* [4, 5].

Несмотря на преимущественное развитие централизованных и распределённых моделей доступа к данным, в 90-х годах XX века в связи с бурным развитием рынка персональных компьютеров широкую популярность стали приобретать однопользовательские СУБД. Однако не только обработка личных архивов и обслуживание деятельности мелких фирм определили для таких СУБД широкие перспективы их использования. Важной оказалась возможность подключения к созданной базе данных собственных приложений, разработанных на языках высокого уровня, а также возможность таких СУБД подключаться в качестве полнофункционального клиента к сетевым базам данных. Очевидно, что на рынке операционных систем Windows было бы странно не увидеть в фаворитах сектора персональных СУБД программные продукты от компании Microsoft (MS).

Реляционная СУБД MS Access известна с 1992 года. Традиционно она входит в состав пакета MS Office. Для работы предполагается использование версии MS Office не ниже 2003.

Каждая база данных в формате MS Access обычно хранится в одном файле с расширением *mdb*. Обеспечивается хранение тек-

[1] Подробнее см. раздел 2.1.

стовой информации в формате unicode, гиперссылок, дат, бинарной и числовых величин с различной точностью. Основным языком программирования является VBA (Visual Basic for Applications), основным языком манипуляции данными является SQL (Structured Query Language).

Язык SQL [6, 7], разработка которого начиналась в корпорации IBM, с 1986 года является официальным и фактическим стандартом структурного языка для работы с реляционными базами данных. В настоящее время известно восемь вариантов его стандартов ANSI/ISO, начиная от SQL-86 до SQL-2008, причём каждый последующий включает в себя возможности предыдущего. Наиболее широко используемым считается стандарт SQL-92 (он же SQL-2), возможности которого достаточны для воспроизведения базовых операций реляционной алгебры, а синтаксис – для формирования наиболее часто встречающихся выборок данных.

SQL существенно отличается от классических языков программирования высокого уровня: например, он не позволяет осуществлять действия на уровне доступа к ячейкам таблиц как к элементам массива. Напротив, язык ориентирован на работу со множествами, и сутью его является формирование *запросов* к нормализованным таблицам базы данных, связанных внешними ключами. Работа с данными в рамках SQL апеллирует к понятию *транзакция*, которое можно раскрыть как последовательность операций над БД, рассматриваемых СУБД как единое целое.

Хорошо проработанный пользовательский интерфейс MS Access позволяет без особого труда создавать простейшие базы данных, а также формы и отчёты для доступа к данным даже неподготовленным пользователям. Разработка сложных элементов пользовательского интерфейса невозможна без знания основ программирования на языке VBA. Сложность обычно составляет не синтаксис и правила языка Basic (вряд ли найдётся более удобный для изучения язык программирования высокого уровня) и не работа с объектами VBA (система помощи содержит максимально подробные сведения), а программирование интерфейса, управляемого событиями. В частности, для реализации необходимой функциональности, помимо правильно оформленного запроса к данным, необходимо корректно отобразить результаты

выборки, от которых обычно зависит состояние нескольких визуальных элементов в формах и отчётах.

В этой связи в данном пособии предлагается разобрать учебный пример создания базы данных, начиная от проектирования структуры до организации пользовательского интерфейса, точно удовлетворяющих поставленным перед разработчиком задачам. Рецепты, предложенные автором, отнюдь не являются единственным вариантом реализации проекта, иные подходы можно посмотреть в рекомендованной или актуальной на данный момент литературе, посвящённой СУБД Microsoft Access [8, 9].

Выбор в пользу MS Access был также сделан в связи с удобной возможностью создавать собственные приложения, которые используют созданную в среде этой СУБД базу данных вместо (или вместе с) пользовательским интерфейсом MS Access. Используя данный подход, можно реализовать доступ к одному файлу базы данных нескольких клиентов, каждый из которых может быть написан на своём языке программирования и решать свой круг задач (т.н. технология файлового сервера) [10].

В нашем случае в качестве «внешнего» клиента предлагается создать отдельное приложение (фактически – исполняемый exe-файл, компонуемый с библиотеками NET.Framework), разработанный в максимально совместимой по языку и модулям с базами данных Access интегрированной среде разработки (IDE) MS Visual Studio на языке Visual Basic .NET [11]. В пособии приведены инструкции по разработке приложения, ориентированные на версию продукта «2008». В качестве интерфейса доступа к данным предлагается использовать ADO .NET [12].

Глава 1. ОСНОВНЫЕ ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

1.1. Основные понятия

Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели. В общем случае можно выделить следующие этапы проектирования [1–5]:

1. Системный анализ и словесное описание информационных объектов предметной области и связей между ними. Как результат на данном этапе формулируется техническое задание на разработку базы данных.
2. Проектирование инфологической модели предметной области в терминах некоторой семантической модели. *Инфологическая модель* данных – обобщённое, не привязанное к какой-либо ЭВМ или СУБД описание предметной области.
3. Выбор конкретной СУБД и даталогическое или логическое проектирование БД, декомпозиция отношений.
4. Физическое проектирование БД, т.е. выбор эффективного размещения БД на внешних носителях и способа доступа к ней.

Поскольку в контексте данного пособия выбор СУБД предопределён, пункты 2–4 фактически объединяются и выполняются с помощью интерфейса проектирования структуры базы данных в MS Access.

Инфологическое моделирование прежде всего связано с попыткой представления семаники предметной области в модели БД. Основным инструментом разработки инфологических моделей (прежде всего для реляционных баз данных) в настоящий момент являются диаграммы «сущность-связь» (ER – Entity-Relationship).

Конструктивными элементами инфологического моделирования являются сущность, атрибут сущности и ключ сущности.

Сущность (Entity) – объект либо совокупность объектов, имеющих существенное значение для рассматриваемой предметной области, информация о которых подлежит хранению. С по-

мощью сущности описывается класс однотипных объектов, т.е. предполагается, что в системе существует множество экземпляров данной сущности. Экземпляры сущности должны иметь некоторые уникальные признаки, позволяющие отличать их друг от друга.

Атрибут сущности – этот некоторая характеристика экземпляра сущности, определяющая свойства данного представителя класса. При этом набор атрибутов должен быть таким, чтобы можно было различать конкретные экземпляры сущности. Например, атрибутами сущности «книга» могут являться «название», «количество страниц», «издательство» и т.д.

Ключ сущности – неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности. Неизбыточность заключается в том, что удаление любого атрибута из ключа должно приводить к нарушению его уникальности.

Между сущностями могут быть установлены *связи* (relationship), показывающие, каким образом сущности соотносятся или взаимодействуют между собой. Связь может существовать между двумя разными сущностями или между сущностью и ею же самой (т.н. *рекурсивная связь*).

Связи делятся на три типа по *множественности*:

- *один-к-одному* (1..1 либо 0..1) – экземпляр одной сущности связан только с одним экземпляром другой сущности;
- *один-ко-многим* (1..* либо 0..*) – один экземпляр сущности может быть связан с несколькими экземплярами другой сущности;
- *многие-ко-многим* (*..*) – один экземпляр первой сущности связан с несколькими экземплярами второй сущности, а один экземпляр второй сущности при этом связан с несколькими экземплярами первой сущности.

Связи также делятся на два типа по модальности: «обязательная» и «возможная». Связь является обязательной (1..*), если в ней должен участвовать каждый экземпляр сущности; возможной (0..*) – если некоторые экземпляры сущности не участвуют в связи. При этом связь может быть обязательной со стороны одной сущности и возможной со стороны другой.

Характерный пример на приведенные определения – сущность «авторы» и сущность «книги». Между этими сущностями должна быть установлена связь «многие-ко-многим», поскольку некоторые книги написаны несколькими авторами, а некоторые авторы написали более одной книги. В рамках предметной области «библиотека» эту связь можно считать обязательной, поскольку нельзя быть автором книги, не написав хотя бы одну книгу, и практически не бывает книг без авторства (редактора или составителя будем считать автором).

Тип связи «многие ко многим» является временным типом связи, допустимым на ранних этапах разработки инфологической модели. При разработке даталогической модели такой тип нельзя установить для поддержки целостности в реляционной базе данных. Наличие в модели такого типа указывает, что одна или несколько промежуточных сущностей ещё не выявлены и требуют конкретизации. В нашем случае для разрешения связи (*..*) необходима дополнительная сущность, которую можно условно назвать «титульная страница», содержащая в числе прочих атрибуты «название книги» и «фамилия автора», тем самым связывая между собой сущности «авторы» и «книги» по двум внешним ключам в отношении (1..*).

В результате графического построения модели предметной области в виде набора сущностей и связей (*ER-диаграмма*) получается связный граф. В полученном графе не должно быть циклических связей – они выявляют некорректность модели.

Инфологическое моделирование по определению не привязано к реляционным или каким-либо иным базам данных, однако в нашем случае, проектируя структуру базы данных для СУБД Microsoft Access, можно экстраполировать понятия инфологической модели в даталогическую следующим образом:

- сущностью будет являться отношение, которое с точки зрения реляционной алгебры является подмножеством декартова произведения множеств, а с точки зрения прикладного проектирования представляет собой двумерную таблицу в базе данных;

- атрибутом сущности будет являться домен[1], входящий в отношение. Имя атрибута эквивалентно названию столбца в таблице. При этом атрибуты, значения которых однозначно идентифицируют кортежи, называют ключевыми.
- связи в реляционных базах данных создаются с помощью *внешних ключей* (foreign key), связывающих несколько отношений.

Дадим несколько дополнительных определений, необходимых для понимания механизма поддержки целостности данных в реляционных базах. Под *целостностью* реляционной БД понимают соответствие информационной модели предметной области, хранимой в базе данных, объектам реального мира и их взаимосвязям в каждый момент времени.

Возможным (потенциальным) ключом отношения называется набор атрибутов, однозначно определяющий кортеж[2] отношения. В общем случае в отношении может быть несколько возможных ключей. Среди всех возможных ключей отношения обычно выбирают один, который считается главным и который называют первичным ключом отношения.

Правило ссылочной целостности (declarative referential integrity) может быть сформулировано следующим образом: база данных не должна содержать значений внешних ключей, для которых не существует соответствующих значений потенциальных ключей. Поддерживаются следующие принципы управления кортежами связанных отношений:

- кортежи подчинённого отношения уничтожаются при удалении кортежа основного отношения, связанного с ним. Такой способ поддержки ссылочной целостности называют *каскадным удалением* записей;
- кортежи подчинённого отношения модифицируются при изменении ключа основного отношения, связанного с ним, так что условие связи по внешнему ключу сохраняется. Такой способ соответственно называют *каскадным изменением* записей;

---

[1] Некоторое конечное множество элементов

[2] Кортежем в реляционной алгебре называют декартово произведение элементов множеств.

- кортежи подчинённого отношения модифицируются при удалении кортежа основного отношения, связанного с ним, при этом на месте ключа родительского отношения ставится значение NULL. В этом случае информация в базе не удаляется, однако логика представления данных для предметной области искажается.

1.2. Основы теории нормализации

Классическая методика проектирования реляционных баз данных связана с *теорией нормализации*, основанной на анализе функциональных зависимостей между атрибутами отношений. *Функциональной зависимостью* (functional dependency) набора атрибутов **B** отношения **R** от набора атрибутов **A** того же отношения называют такое соотношения проекций[1] **R[A]** и **R[B]**, при котором в каждый момент времени любому элементу проекции **R[A]** соответствует только один элемент проекции **R[B]**, входящий вместе с ним в какой-либо кортеж отношения **R**. Часто применяется обозначение: **R.A** $\rightarrow$ **R.B**. В данном случае **A** называют *детерминантом* функциональной зависимости [4, 6].

Атрибуты могут группироваться в отношения с образованием реляционной схемы на основе либо собственного опыта разработчика базы данных, либо посредством последовательного вывода реляционной схемы из исходной ER-диаграммы. При использовании любого из этих двух подходов часто требуется применять определенный формальный метод, способный помочь проектировщику базы данных найти оптимальную группировку атрибутов для каждого отношения в схеме. Таким методом является *нормализация* – процесс проектирования схемы базы данных с использованием декомпозиции.

Основная цель проектирования реляционной базы данных заключается в группировании атрибутов в отношения таким образом, чтобы минимизировать избыточность данных и сократить объем информации, хранящийся в таблицах, тем самым уменьшая количество возможных ошибок при вводе и обработке дан-

[1] Проекция – это копия отношения, в которую не включены один или несколько атрибутов исходного отношения.

ных. Проектирование схемы базы данных обычно осуществляется путём *декомпозиции* (разбиения) исходного набора отношений, при этом полученные отношения являются проекциями исходных.

Каждой нормальной форме соответствует некоторый определённый набор ограничений. Чаще всего нормализация осуществляется в виде нескольких последовательно выполняемых этапов, каждый из которых соответствует определенной нормальной форме, обладающей известными свойствами. В ходе нормализации структура отношений становится все более фиксированной (строгой), а содержание – менее восприимчивым к аномалиям обновления.

Изучение принципов нормализации требует серьёзного и подробного рассмотрения. В нашем случае ограничимся кратким обзором правил для первых четырёх нормальных форм, на которые чаще всего требуется тестировать структуру базы данных на практике.

1. Отношение находится в первой нормальной форме (1NF) тогда и только тогда, когда значения всех его атрибутов *атомарны*. Это основной (и в то же время самый спорный) постулат теории реляционных баз данных. Иными словами, речь идет о таблице, в которой на пересечении каждой строки и каждого столбца содержится одно и только одно значение, при этом реляционные операции могут действовать на это значение целиком, а не на какую-либо его часть. Обычно требование 1NF автоматически соблюдается при разработке модели данных непосредственно с помощью интерфейса реляционной СУБД.

2. Отношение находится во второй нормальной форме (2NF) тогда и только тогда, когда оно находится в первой нормальной форме и не содержит неполных функциональных зависимостей непервичных атрибутов от атрибутов первичного ключа. Следует уточнить, что функциональная зависимость **R.A → R.B** называется *полной*, если набор атрибутов **B** функционально зависит от **A**, но не зависит функционально от любого подмножества **A**. То есть удаление какого-либо атрибута из **A** приводит к утрате функциональной зависимости.

3. Отношение находится в третьей нормальной форме (3NF) тогда и только тогда, когда оно находится во второй нормальной

форме и не содержит транзитивных функциональных зависимостей[1]. Иными словами, требование 3NF сводится к тому, чтобы все неключевые поля зависели только от первичного ключа и не зависели друг от друга.

4. Отношение находится в нормальной форме Бойса – Кодда (BCNF), если оно находится в третьей нормальной форме и каждый детерминант отношения является возможным ключом этого отношения. То есть отношение находится в BCNF, если любая функциональная зависимость между его атрибутами сводится к полной функциональной зависимости от возможного ключа.

Четвёртая нормальная форма и пятая нормальная форма *проекции/соединения* в данном пособии не рассматриваются. В литературе часто поднимается вопрос об обоснованности применения строгой нормализации, поскольку увеличение количества таблиц наряду с повышением качества описания предметной области и уменьшением количества возможных ошибок при вводе и обработке данных в то же время приводит к увеличению сложности создаваемых запросов и к появлению связей, реально не существующих в описываемой предметной области. Это в некоторых случаях приводит к снижению производительности исполняющей системы и, в большинстве случаев, к увеличению числа ошибок при программировании соответствующих запросов. Разумным практическим компромиссом для большей части моделей данных является приведение схемы отношений к третьей нормальной форме, либо к BCNF.

Возможным способом практической разработки схемы данных (инфологической модели) является предварительный анализ предметной области с целью выявления сущностей *словарно-справочного характера* и сущностей, непосредственно связанных с *манипуляциями* в предметной области. Обычно первая категория сущностей представляет собой переферийное окружение, связываемое с центральными сущностями из второй категории.

---

[1] Если для атрибутов **A**, **B** и **C** некоторого отношения существуют зависимости вида $\mathbf{A} \rightarrow \mathbf{B}$ и $\mathbf{B} \rightarrow \mathbf{C}$, это означает, что атрибут **C** *транзитивно* зависит от атрибута **A** через атрибут **B**, при условии что атрибут **A** функционально не зависит ни от атрибута **B**, ни от атрибута **C**.

Анализ связей между сущностями необходим для приведения всех связей к виду «один-ко-многим» либо «один-к-одному». В результате этого анализа могут появиться (или исчезнуть) ранее не учтённые сущности. На следующем этапе анализа следует внимательно оценить полученную ER-диаграмму на предмет циклических связей. Если имеется хотя бы одна такая связь, схему данных либо не удастся создать, либо (что хуже) созданную БД будет весьма проблематично заполнять и практически невозможно обрабатывать.

Глава 2. РАЗРАБОТКА УЧЕБНОГО ПРОЕКТА СРЕДСТВАМИ MICROSOFT ACCESS

2.1. Постановка целей и общее содержание учебного проекта

Создадим систему управления малой библиотекой (например, личной библиотекой книг, дисков или библиотекой организации), которая будет содержать собственно базу данных для хранения информации о книгах (при выполнении учебных заданий можно выбрать иные единицы хранения) и пользовательский интерфейс, который будет реализован:

1) в виде форм и отчётов MS Access, обеспечивающих удобное наполнение и управление базой данных (учёт, поиск, сортировка по различным критериям, формирование итоговых диаграмм и сводных таблиц);

2) в виде внешних программ, созданных на языке Visual Basic для создания интерфейса библиотечного работника по приёму-выдаче книг читателям.

Для реализации проекта необходимо последовательно выполнить следующие действия:

- Разработать структуру базы данных, создать таблицы и связи между ними.
- Наполнить необходимое количество таблиц первичным содержанием.
- Разработать формы для ввода, поиска и сортировки информации о книгах.
- Разработать в среде Visual Basic программу-интерфейс для регистрации приёма-выдачи книг.
- Разработать в MS Access отчёты по итогам работы библиотеки (анализ востребованности книг, загруженность библиотеки и т.п.).

Перед началом выполнения предложенной практической работы следует изучить и, по возможности, иметь под рукой литературу по работе с MS Access и Visual Basic.NET [4–10], а также использовать любые другие источники информации, дающие квалифицированные советы и указания на уровне программиста-разработчика баз данных.

Следует также иметь в виду, что программные продукты, на которые ориентирована данная книга, находятся в состоянии непрерывного совершенствования, так что описание изложенных здесь рекомендаций через определённое время может оказаться устаревшим, чаще всего из-за изменившегося интерфейса среды разработки или совершенствования объектной структуры языка. Другая часть рекомендаций, наоборот, будет реализована уже без неприятных сюрпризов со стороны программных средств. Автор не останавливается на проблемах продуктов Microsoft, подстерегающих разработчика, не без оснований полагая, что нередко непонятные ошибки возникают из-за неуверенных действий самого программиста, а наиболее неприятные проблемы обычно исправляются с выходом новых обновлений или новых версий рассматриваемых продуктов. Наконец, в данном пособии автор показывает создание универсального программного продукта, иллюстрирующего доступные начинающим возможности технологий управления данными, намеренно не вступая в область профессионального библиотечного дела.

2.2. Организация структуры базы данных

Схема структуры базы данных приведена на рис. 2.1–2.3, а способ формирования входящих в неё таблиц проиллюстрирован серией рисунков 2.4–2.15.

База данных состоит из следующих основных таблиц:

- **Books** – содержит описание единиц хранения (т.е. книг) в библиотеке. Здесь описываются все книги, принадлежащие библиотеке, в т.ч. не рекомендованные к выдаче читателям или, наоборот, не возвращённые читателями.
- **Users** – содержит список посетителей библиотеки (в который могут входить, помимо читателей, поставщики литературы либо лица, по каким-либо причинам уже выписанные из библиотеки).
- **Actions** – таблица действий, совершаемых библиотекарем. Список действий содержится в так называемой справочной таблице *action_status*, среди которых: «Выдача книги читателю», «Приём книги от читателя», «Продление срока выдачи книги» и «Регистрация новой книги».

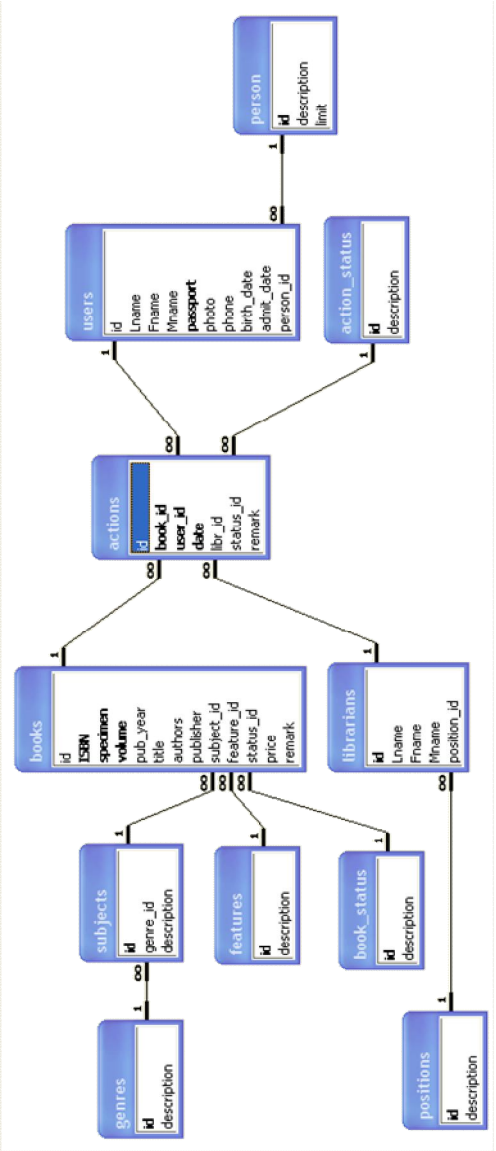


Рис. 2.1. Схема базы данных «Library»

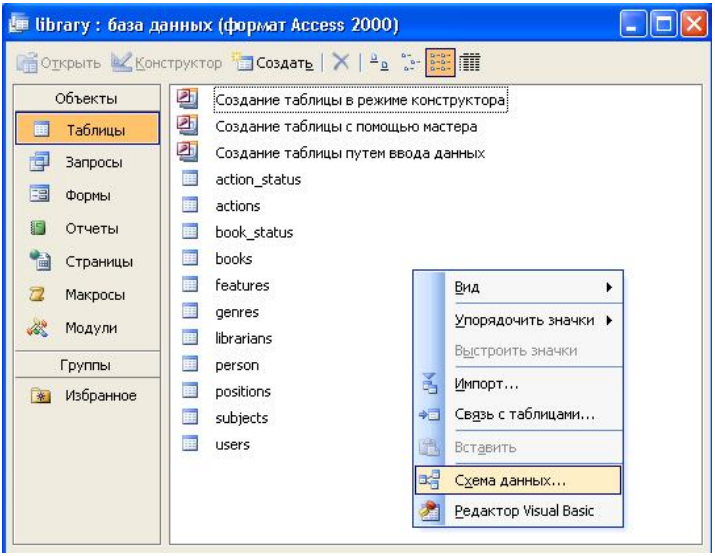


Рис. 2.2. Переход к схеме данных может быть осуществлён из выпадающего меню

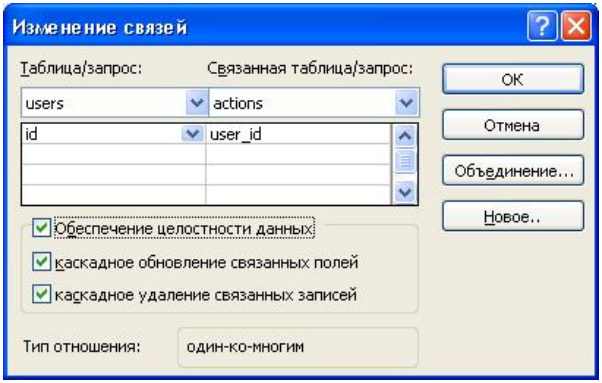


Рис. 2.3. В проекте по умолчанию для всех связей между таблицами предполагается обеспечение целостности данных (каскадное обновление и удаление)

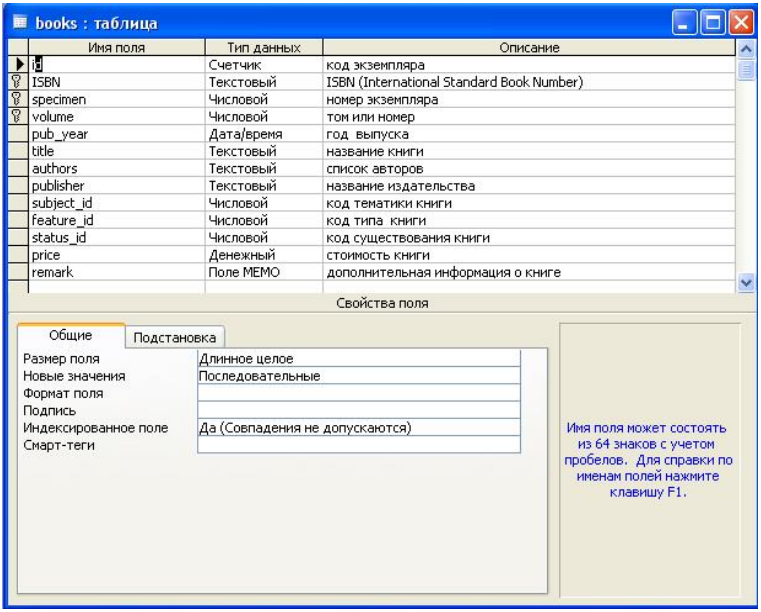


Рис. 2.4. Структура таблицы *books*

Следует обратить внимание, что составной первичный ключ (рис. 2.4) выбирается с целью не допустить повтора в таблице одной и той же книги (или экземпляра книги, или номера журнала) дважды. В этой связи поле *id* используется как альтернативный потенциальный ключ и должно принимать только уникальные значения. В противном случае связь между таблицами создать не удастся.

Ситуация, при которой ISBN книги неизвестен, может быть решена в рамках конкретной предметной области различными способами. Проще всего ввести собственный классификатор, для чего использовать то же самое поле ISBN, либо, что лучше, увеличить число символов этого поля для установки признаков новой (или собственной) классификации при сохранении преемственности с прежними схемами.

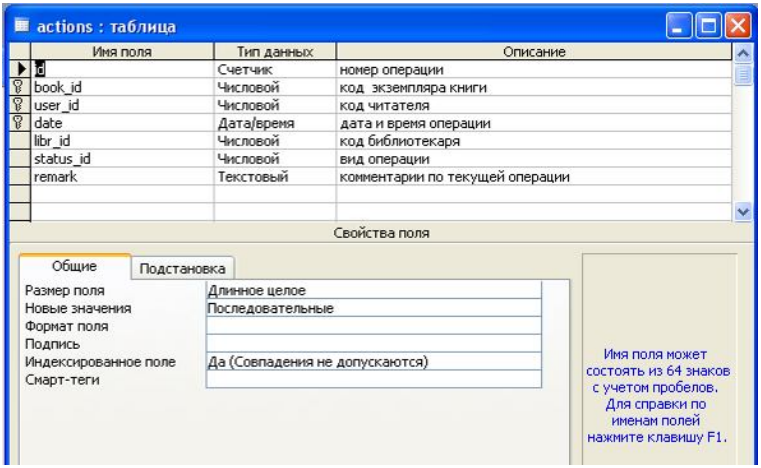


Рис. 2.5. Структура таблицы *actions*

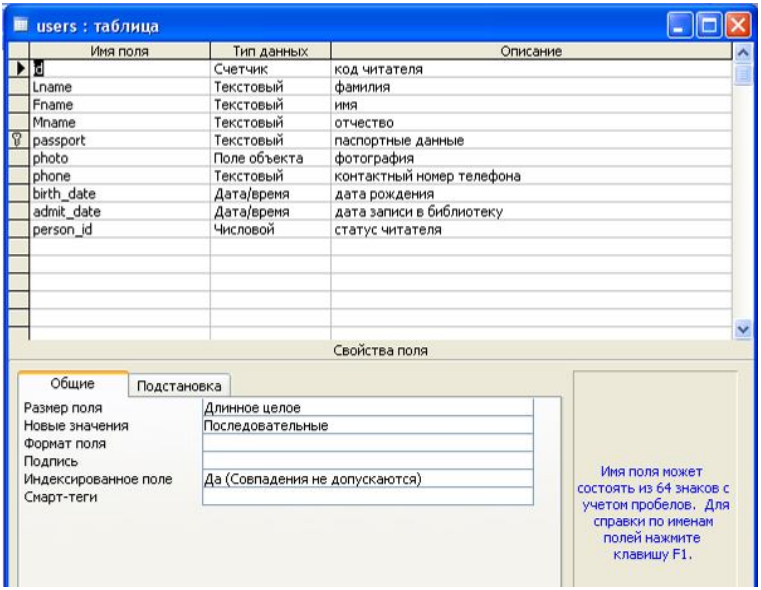


Рис. 2.6. Структура таблицы *users*

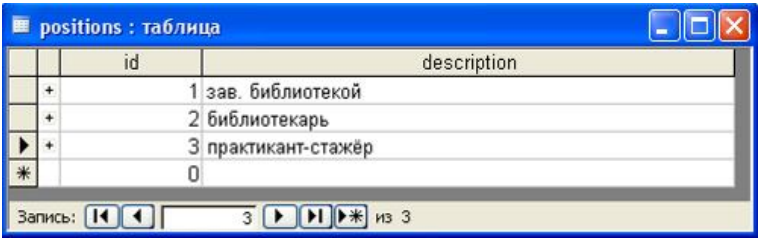


Рис. 2.7. Примерное содержание таблицы *positions*. Здесь и далее поле *id* имеет тип «длинное целое», а поле *description* – текстовый тип максимальной для MS Access длины (255 символов)

Рис. 2.8. Примерное содержание таблицы *librarians*

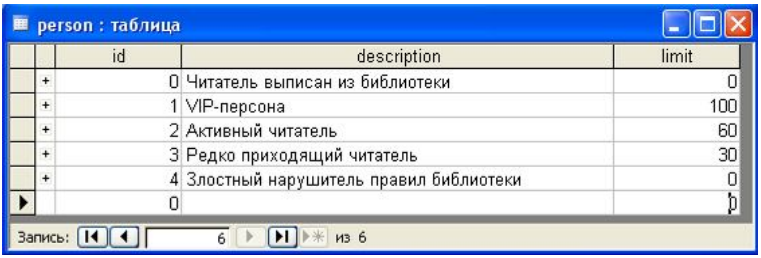


Рис. 2.9. Примерное содержание таблицы *person*. Поле *limit* имеет смысл количества дней, которые читатель может держать книгу «на руках», не нарушая правила библиотеки

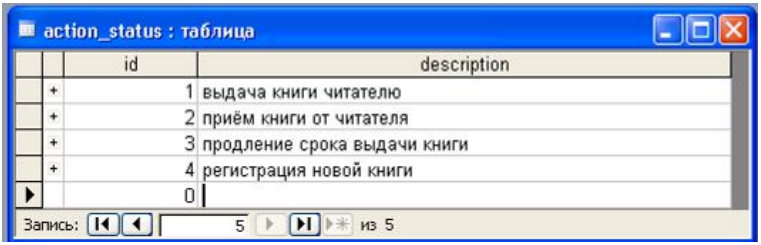


Рис. 2.10. Примерное содержание таблицы *action_status*

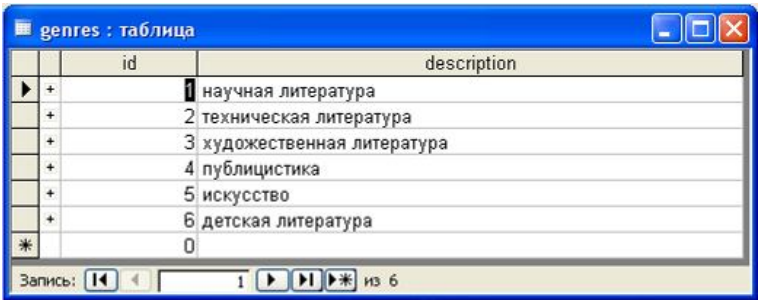


Рис. 2.11. Примерное содержание таблицы *genres*

При наполнении созданной структуры таблиц вначале заполняются все периферийные справочные таблицы (см. рис. 2.1), затем таблицы, к которым имеются внешние ключи от справочных таблиц. Нужно обратить внимание, что таблицы *books* и *actions* будут заполняться в последнюю очередь – тогда, когда вы создадите формы пользовательского интерфейса для их заполнения. Хотя в процессе отладки, естественно, вводить и редактировать записи в этих таблицах придётся вручную. Для заполнения, редактирования и просмотра таблицы *users* впоследствии необходимо будет самостоятельно разработать подходящую форму, на данном этапе рекомендуется внести в эту таблицу несколько записей для читателей библиотеки и поставщика литературы, пользуясь штатными средствами Microsoft Access.

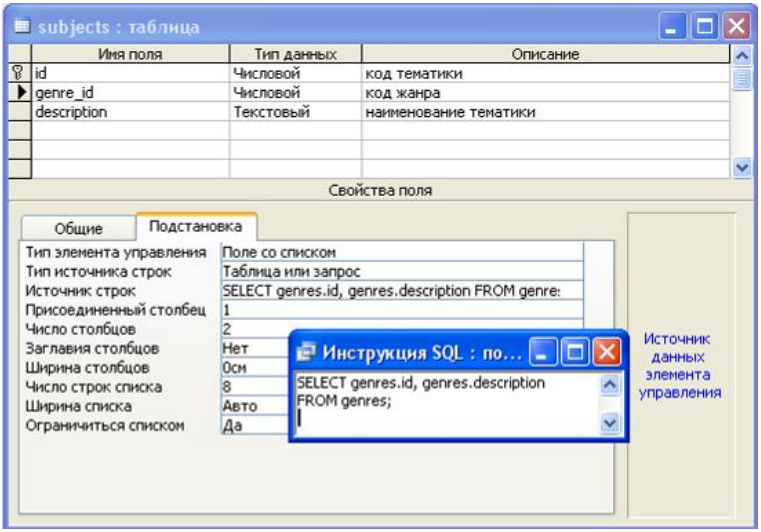


Рис. 2.12. Структура таблицы *subjects.* На рисунке показано, как организуется поле подстановки для более удобного ввода данных в таблицу, используя штатные средства Microsoft Access

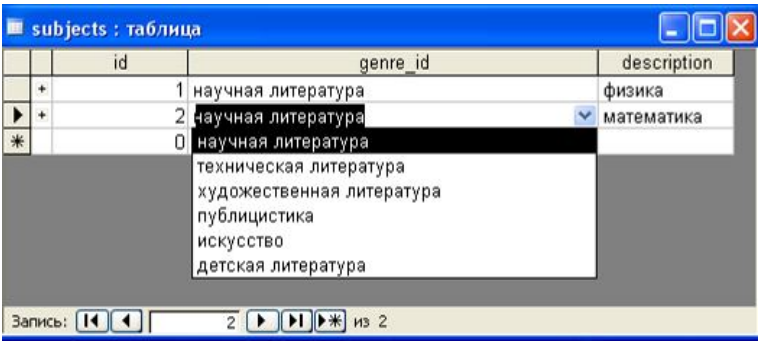


Рис. 2.13. Результат использования поля подстановки

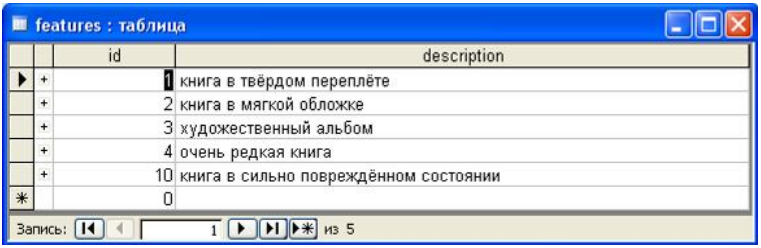


Рис. 2.14. Примерное содержание таблицы *features*

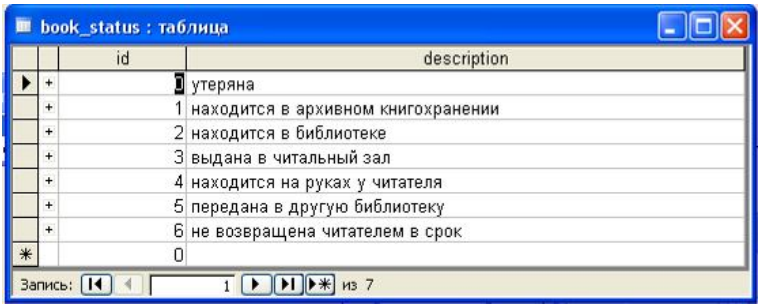


Рис. 2.15. Примерное содержание таблицы *book_status*. Очевидно, что адаптация данного учебного варианта базы данных к конкретной предметной области приведёт к изменению структуры базы данных и содержанию таблиц

2.3. Разработка формы для регистрации новых книг

Назначением формы является организация удобного полнофункционального пользовательского интерфейса для регистрации новой литературы в библиотеке. Интерфейс должен быть дружественным к неподготовленному пользователю и реализовывать по возможности максимальную защиту от ошибок ввода данных. Помимо того, что в результате регистрации новой книги заполняется таблица *books*, информация о регистрации является неким стандартным действием библиотекаря, и потому персонифицированный акт регистрации должен быть отмечен и в таблице *actions*.

Примерный вид формы для ввода данных представлен на рис. 2.16. Для создания представленной формы предлагаемый в Access мастер может быть полезен только на первом этапе, хотя в ряде других случаев ему следует отдавать предпочтение. Данную форму мы не будем напрямую связывать с какой-либо таблицей базы данных, поскольку в нашем случае ввод данных будет осуществляться путём выполнения запросов на изменения данных в таблицах при нажатии на кнопку «Ввести данные в базу». Это даст возможность манипулировать содержимым полей ввода данных по своему усмотрению, поскольку ни один активный элемент формы не будет привязан к атрибутам таблиц. Помимо сказанного, такую форму легче отлаживать.

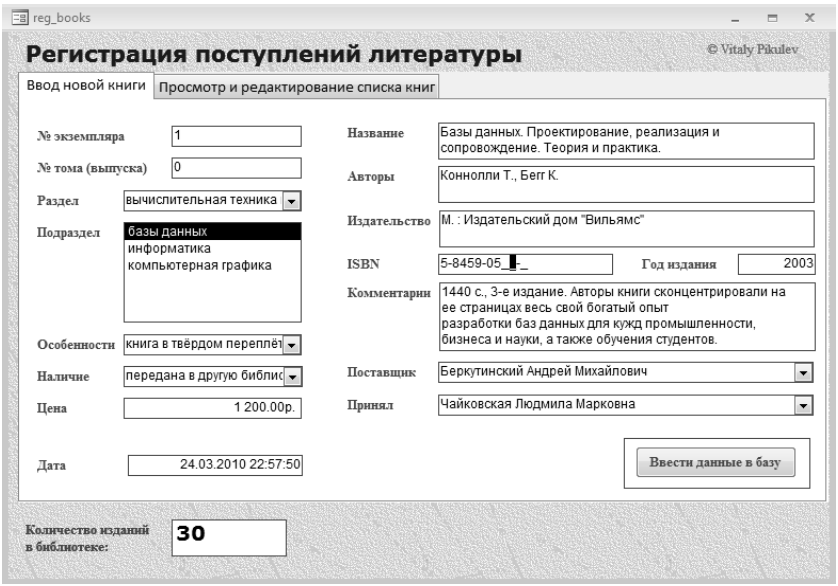


Рис. 2.16. Форма для ввода информации о новых книгах

Начальные настройки формы представлены на рис. 2.17. На вкладке *Данные* свойств формы источник записей не определён. Наиболее рациональным способом создания такой формы является первоначальное использование мастера по таблице *books*, но после завершения работы мастера все поля ввода следует отсо-

единить от таблицы (т.е. сделать свободными), ввести недостающие поля («Поставщик», «Принял», «Дата»), нарисовать кнопку «Ввести данные в базу», а саму таблицу *books* из источника записей формы убрать. Другим хорошим вариантом является создание всех полей ввода вручную, без использования мастера.

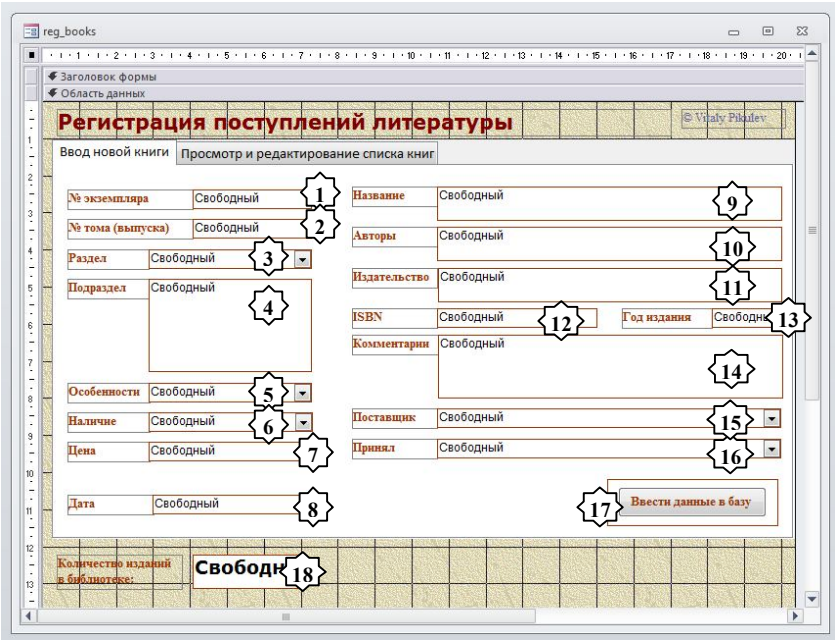


Рис. 2.17. Режим конструктора форм.

Цифрами на рисунке обозначены элементы со следующими именами:

1) var_specimen;
2) var_volume;
3) var_genre;
4) var_subject;
5) var_feature;
6) var_status;
7) var_price;
8) var_date;
9) var_title;
10) var_authors;
11) var_publisher;
12) var_ISBN;
13) var_year;
14) var_remark;
15) var_user;
16) var_librarian;
17) var_ready;
18) var_count.

Для имен полей ввода в формах в данном пособии принят префикс *var_*, чтобы формально отличать их от наименований полей в таблицах. На рис. 2.18 представлен вариант защиты от ввода некорректных данных для поля *var_price*. Для большинства полей формат вывода данных обычно не указывается, для *var_price* рекомендуется указать «Денежный».

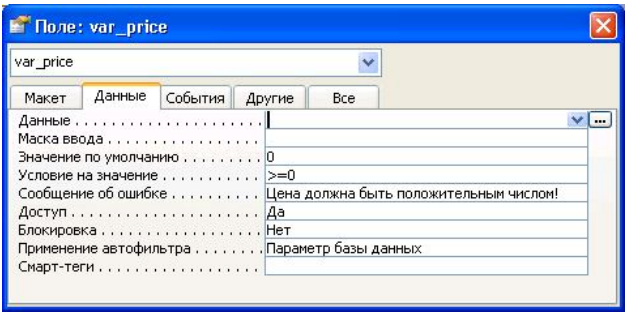


Рис. 2.18. Свойства одного из полей ввода в форме

Поскольку практически никогда не бывает безошибочного ввода данных, в представленной форме предусмотрены две вкладки (*Tab Control*) с названиями «Ввод новой книги» и «Просмотр и редактирование списка книг». Сейчас практически все элементы управления следует размещать на первой из этих вкладок. На второй вкладке позже будет размещена подчинённая форма с представлением данных в виде таблицы с возможностью редактирования полей (см. Задание 2 для самостоятельной работы). После оформления первоначального дизайна формы неплохо будет откорректировать последовательность смены фокуса между визуальными элементами формы с помощью меню *Вид → Последовательность перехода...*

Источник данных для поля со списком *var_genre* «Раздел» указывается так, как показано на рис. 2.19. Для создания и отладки запроса щёлкните на символ многоточия в строке «Источник строк» и выберите либо интерактивный построитель запросов (см. рис. 2.20, по умолчанию), либо *режим SQL*. Можно переключаться между этими режимами и режимом отладки (*Режим таблицы*), однако при этом возможно, что набранный вами текст

будет модифицирован самой СУБД. Поэтому рекомендуется инструкции разработанных запросов (особенно когда они станут сложными) сохранять отдельно в текстовом файле.

После создания запроса следует откорректировать макет поля в соответствии с числом возвращаемых запросом столбцов (рис. 2.21). В нашем случае: установить два столбца данных, а ширину первого столбца принять равной нулю, поскольку он содержит код *id* и не нуждается в отображении на экране. Ширина списка откорректирована таким образом, чтобы длинные названия жанров не оказались урезанными.

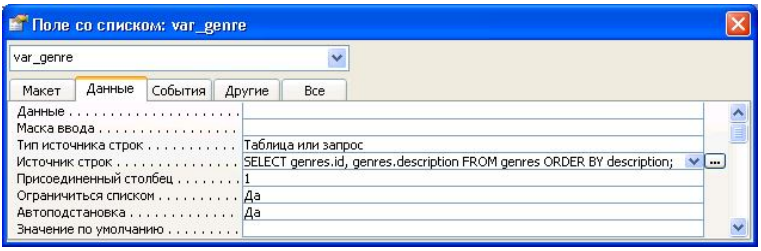


Рис. 2.19. Источник строк для поля со списком

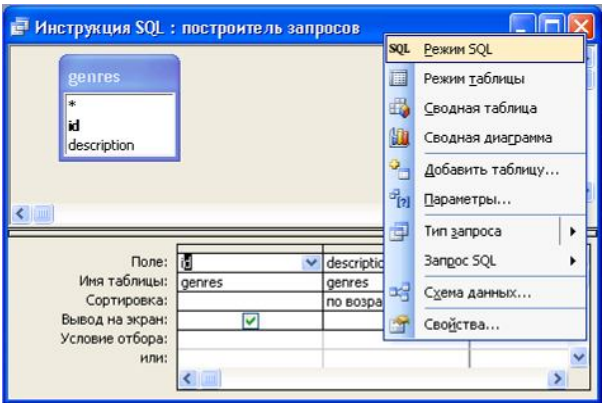


Рис. 2.20. Построитель запросов

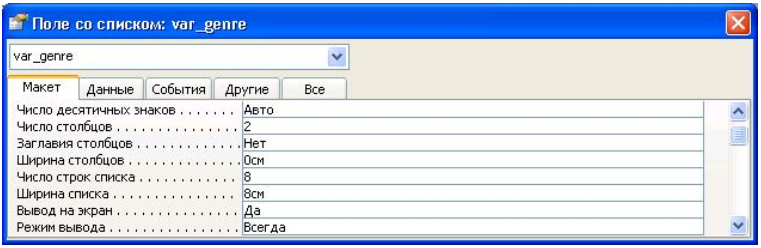


Рис. 2.21. Особенности построения макета поля со списком

Выпадающие списки *var_feature* «Особенности» и *var_status* «Наличие» определяются аналогичным образом по своим справочным таблицам. Поле-список *var_subject* «Подраздел» также строится аналогично полю «Раздел», но запрос учитывает зависимость данных в этом поле от значений поля *var_genre*. Текст запроса выглядит следующим образом (рис. 2.22).

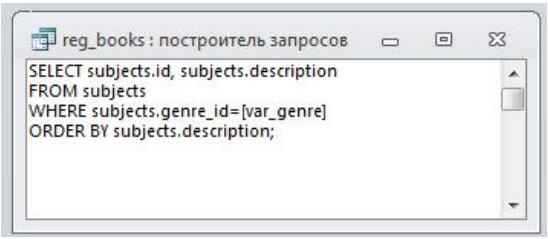


Рис. 2.22. SQL-запрос со значением из поля активной формы

Представленным способом можно вводить значения полей формы (точнее, следует сказать: Forms!My_object.value) в текст SQL-инструкции.

Для того чтобы автоматически обновлять состояние поля «Подраздел» при изменении значения поля «Раздел», требуется программно обработать событие обновления данных для принудительного обновления остальных полей формы. В свойствах поля «Раздел» на вкладке *События* щёлкните мышкой на многоточие в конце поля *После обновления*, в окне *Построитель* выберите *Программы*. Откроется редактор Visual Basic for Access. Вве-

дите код так, как показано на рис. 2.23. Проверьте согласованное изменение данных в соответствующих полях.

Поле *var_count* «Количество изданий в библиотеке» на самом деле – список, единственное значение которого строится по запросу, приведенному на рис. 2.24. Поле необходимо для того, чтобы пользователь видел, как наполняется таблица в результате регистрации новых книг.

Значения, введённые в текстовые поля *var_title* «Название», *var_authors* «Авторы», *var_publisher* «Издательство», *var_remark* «Комментарии» с помощью запроса, о котором пойдёт речь в конце раздела, будут добавлены в соответствующие столбцы таблицы *books*.

Для текстовых полей *var_ISBN* и *var_year* предлагается использовать маски ввода. Оба эти поля могут содержать только цифры, но значения в *var_ISBN* удобно держать в текстовом формате, а полю *var_year* подойдёт целочисленный (но только не интуитивно предполагаемый формат *дата-время*).

Рис. 2.23. Окно VBA. Стрелкой указано место, куда следует вписать команду

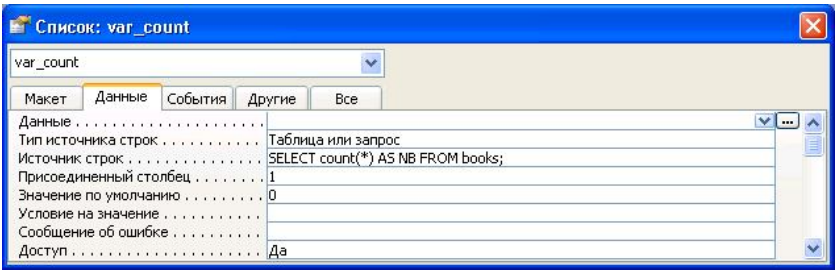


Рис. 2.24. Примерный вариант сбора статистики в форме

Один из вариантов создания маски ввода для кодов ISBN представлен на рис. 2.25 (см. ранее на рис. 2.16 вид поля с маской). При этом в поле таблицы будет сохранена только последовательность из 10 цифр. Аналогично маска в поле *var_year* должна позволить ввести только четыре цифры года.

Информацию для выпадающего списка *var_user* «Поставщик» получаем из таблицы *users*. Если ввести в таблицу *Person* некую строку с *id* = 5 «Поставщик литературы», то легко организовать запись в таблице *users* для человека (или организации) с соответствующим идентификатором. Запрос на заполнение поля «Поставщик» представлен на рис. 2.26. После выбора элемента из списка значение *var_user* будет соответствовать *id* пользователя из таблицы *users*.

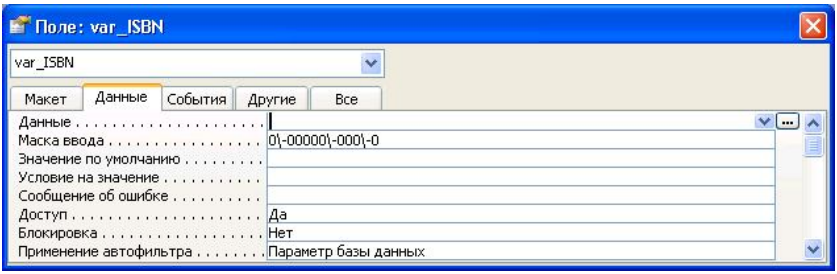


Рис. 2.25. Маска ввода для поля

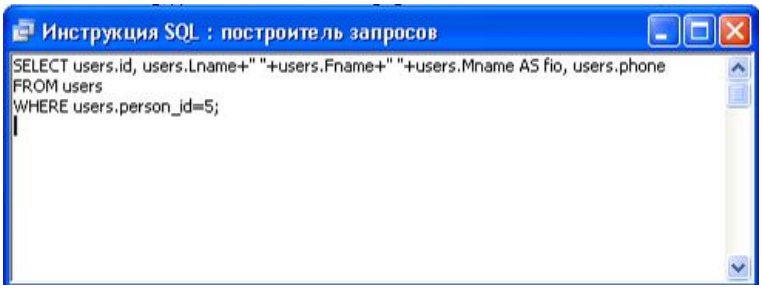


Рис. 2.26. Источник записей для списка *var_user*

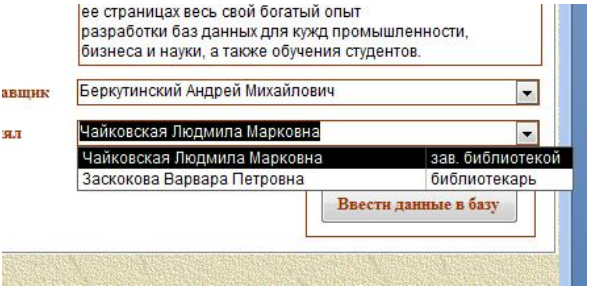


Рис. 2.27. Пример работы с полем с выпадающим списком

Выпадающий список *var_librarian* «Принял» формируется по таблице *librarians*. Работа со списком показана на рис. 2.27.

Формат Поля *var_date* «Дата» указан как «Полный формат даты», а значение по умолчанию – функция *Now()*, возвращающая текущую дату и время. Значение этого поля, как и двух перечисленных выше, должно быть записано в таблицу *actions* в случае, если удалось ввести новую книгу в таблицу *books*.

Последним визуальным элементом формы является кнопка *var_ready* «Ввести данные в базу». При создании кнопки обычно запускается мастер MS Access, который позволяет выбрать функцию, реализуемую кнопкой. В нашем случае выбранная функция может быть любой, поскольку она всё равно будет изменена, но программный код, окружающий функцию, будет использован при написании собственных фрагментов программы.

Программа, выполняемая кнопкой, прикрепляется к событию «Нажатие кнопки». Примерный текст соответствующей процедуры приведен ниже.

```
Private Sub var_ready_Click()
On Error GoTo Err_var_ready_Click
   var_i = var_count.Column(0, 0) 'текущее число записей в таблице books
   DoCmd.SetWarnings False 'прекращаем выдачу уведомлений от СУБД
   DoCmd.OpenQuery "insert_book", acViewNormal, acAdd
            'выполняем запрос на добавление данных в books
   DoCmd.SetWarnings True 'вновь разрешаем выдачу уведомлений
   Me.Refresh 'обновляем содержимое окна активной формы
   If var_count.Column(0, 0) > var_i Then 'изменилось ли число строк в books?
```

```
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "insert_action", acViewNormal, acAdd
        'если да, выполняем запрос на добавление данных в actions
    DoCmd.SetWarnings True
    MsgBox "Информация успешно сохранена в базе!", vbInformation, ¬
"Данные введены"
  Else
  MsgBox "Запись невозможна! Вероятно, такие данные уже ¬
существуют в базе.", vbCritical, "Проблема"
  End If

Exit_var_ready_Click:
  var_date.Value = Now() 'изменяем значение поля, ¬
здесь же можно инициализировать другие поля формы
  Exit Sub

Err_var_ready_Click:
  MsgBox Err.Description
        'СУБД информирует, что выполнение команды завершилось ошибкой
  Resume Exit_var_ready_Click

End Sub
```

Сами запросы на добавление данных в таблицы *books* и *actions* создаются как объекты базы данных (рис. 2.28). Рекомендуется приступать к созданию запроса в режиме конструктора, при этом выбрать тип запроса «Добавление» и перейти в режим SQL. В качестве аргументов строка запроса будет брать значения полей из формы *reg_books*, поэтому если форма не активна или соответствующие поля пусты, запрос не будет выполнен корректно.

Примерный текст запроса insert_book приведен ниже:

```
INSERT INTO books ( ISBN, specimen, volume, pub_year, title, authors, publisher,
subject_id, feature_id, status_id, price, remark ) VALUES
(Forms!reg_books.var_ISBN, Forms!reg_books.var_specimen,
Forms!reg_books.var_volume, "01.01."+Str(Forms!reg_books.var_year),
Forms!reg_books.var_title, Forms!reg_books.var_authors,
Forms!reg_books.var_publisher, Forms!reg_books.var_subject,
Forms!reg_books.var_feature, Forms!reg_books.var_status,
Forms!reg_books.var_price, Forms!reg_books.var_remark);
```

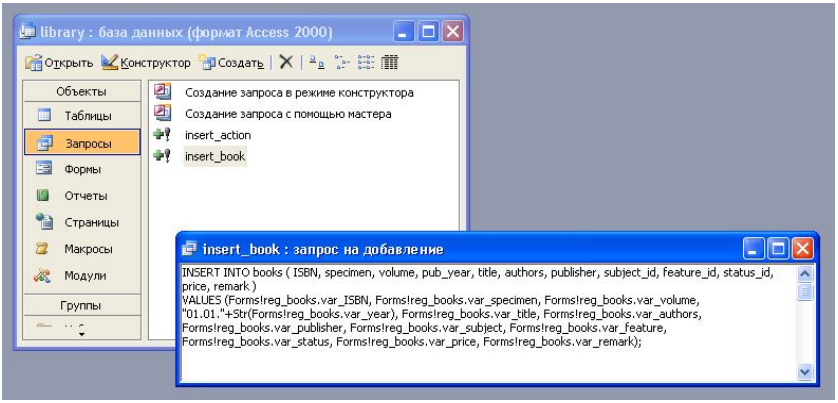


Рис. 2.28. Формирование запросов к данным

Рис. 2.29. Запрос на добавление данных

Запрос *insert_action* на добавление записи в таблицу *actions* несколько проще, однако требует дополнительных данных, которых нет в разработанной форме (рис. 2.29). Действительно, для создания записи в *actions* требуется знать *book_id* только что введённой книги, который присваивается элементу записи автоматически в таблице *books* после выполнения первого запроса. Поэтому второй запрос формально обращается к таблице *books*, забирая оттуда только max(*id*) (здесь важно, чтобы установленный для этого поля счётчик выдавал только последовательность возрастающих чисел). Три остальных поля берутся из формы *reg_books*, а последняя константа «4» указывает на вид операции «Регистрация новой книги» (см. рис. 2.10).

После этого можно протестировать полученную форму, осуществив наполнение базы данных информацией о литературе. Вкладку «Просмотр и редактирование списка книг» пока остав-

ляем чистой, к ней вернёмся позже. Рекомендуется в ходе работы над системой ввести не менее 30 наименований книг (или журналов) разных тематик.

Несомненное удобство разработанный формы заключается в том, что если книга регистрируется в библиотеке в нескольких экземплярах, то изменяется поле «№ экземпляра», а все остальные поля могут оставаться без изменений. Если регистрируется многотомное произведение или журнал, изменяется только поле «№ тома», все остальные поля остаются без изменений.

2.4. Разработка формы для просмотра, редактирования и поиска книг в библиотеке

Форма *show-books* является основным рабочим инструментом электронной библиотеки, поэтому количество функций, которые она может реализовать, определяется только фантазией разработчика (или реальным техническим заданием). Примерный вид формы в рабочем состоянии показан на рис. 2.30.

Форма реализует следующие функции:

- Просмотр полного списка наименований книг и их атрибутов, находящихся на учёте в библиотеке, с возможностью сортировки по любому полю.
- Выборочный просмотр книг с учётом одного или нескольких одновременно действующих критериев (например, вид переплёта, наличие в библиотеке, тематика книги).
- Поиск по любому из полей.
- Просмотр состояния каждого из экземпляров либо томов соответствующей книги.

Реализация формы приведена на рис. 2.31. По способу создания эта форма напоминает предыдущую: она свободная, т.е. не привязана к конкретной таблице базы данных. Однако главная форма содержит подчинённую, которая формируется на основе запроса. Поэтому после формирования первоначального дизайна первое, что следует сделать – это создать подчинённую форму.

Рис. 2.30. Форма для просмотра списка литературы

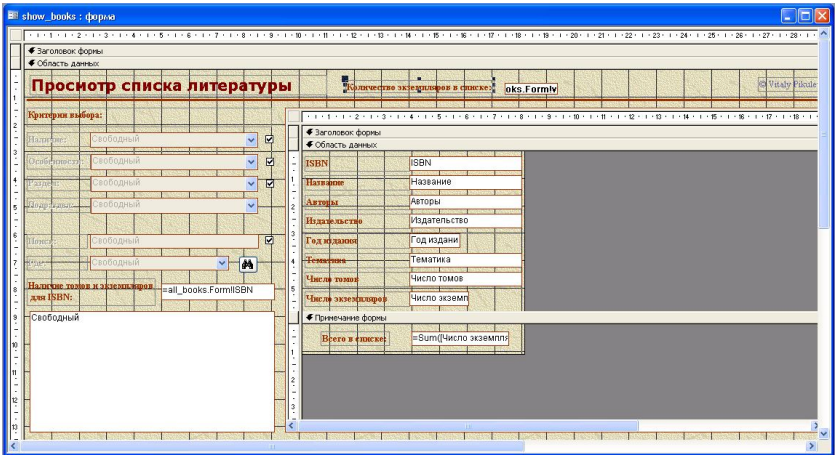


Рис. 2.31. Форма в режиме конструктора

Создание подчинённой формы можно начать с разработки и отладки запроса на выборку данных, который будет поставлять требуемые данные. Поскольку в запросе используется практически вся совокупность таблиц, он может получиться достаточно громоздким.

Вариант запроса *all_books* на выборку для подчинённой формы выглядит следующим образом:

```
SELECT books.ISBN AS ISBN, max(books.title) AS Название, max(books.authors) AS
Авторы, max(books.publisher) AS Издательство, max(year(books.pub_year)) AS
[Год издания], max(subjects.description+" ("+genres.description+")") AS Тематика,
iif(max(books.volume)=0,"одной книгой",count(books.volume)) AS [Число томов],
max(books.specimen) AS [Число экземпляров], max(books.status_id) AS status,
max(books.feature_id) AS feature, max(books.subject_id) AS subject
FROM features
        INNER JOIN (book_status
        INNER JOIN ((genres INNER JOIN subjects ON genres.id=subjects.genre_id)
        INNER JOIN books ON subjects.id=books.subject_id) ON
        book_status.id=books.status_id) ON features.id=books.feature_id
GROUP BY books.ISBN;
```

В чём особенность представленного запроса? Здесь можно увидеть группировку по полю ISBN и появление необходимых функций агрегирования у всех остальных выводимых полей. Дело в том, что в соответствии с правилами предметной области уместно показывать в таблице только книги, отличающиеся по библиотечному классификатору. Например, если в библиотеке присутствует 50 одинаковых экземпляров какого-либо методического пособия, перечислять их все в основной таблице нет необходимости. Про них можно будет узнать с помощью дополнительного поля-списка. Точно так же и перечисление всех томов одного издания в основной таблице излишне.

Вторая особенность запроса заключается в том, что наряду с полями, предназначенными для вывода на экран (они озаглавлены русскими наименованиями), в списке вывода перечислены поля-идентификаторы (*status_id, feature_id, subject_id*). Эти поля потребуются для работы системы фильтрации.

И третья особенность запроса состоит в использовании функции выбора *iif()*. Она здесь нужна, чтобы заменить неуместное для конечного пользователя цифровое значение на более содержательную текстовую информацию.

При разработке запроса не забывайте, что проверить его можно тотчас же, переключив окно запроса в режим «Таблица». Для создания подчинённой формы, безусловно, следует воспользоваться мастером, который автоматически запустится, когда вы выберете инструмент «Подчинённая форма/отчёт». В качестве источника данных следует указать только что разработанный запрос (рис. 2.32).

После завершения работы мастера будет автоматически сгенерирована подчинённая форма *all_books* и вставлена в основную, с возможностью редактирования её элементов в виде формы. Однако в режиме исполнения основной формы подчинённая будет всегда представляться в виде таблицы, как было показано на рис. 2.30.

Сортировка столбцов подчинённой формы выполняется встроенными функциями Access, поэтому специально программировать этот механизм не требуется (рис. 2.33).

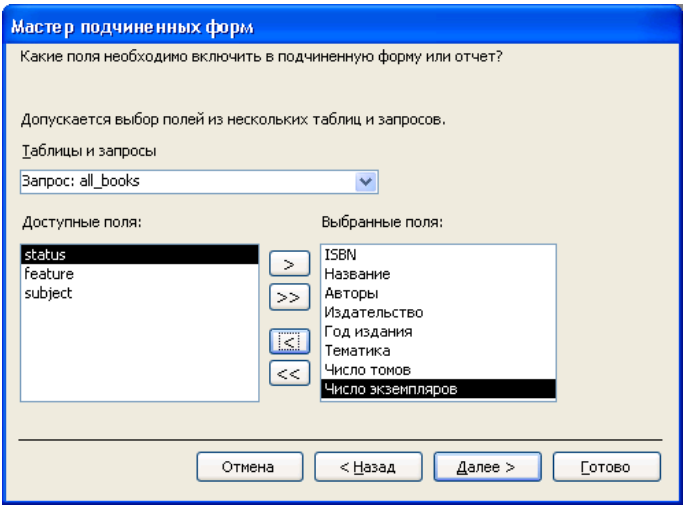


Рис. 2.32. Выбор источника строк для подчинённой формы

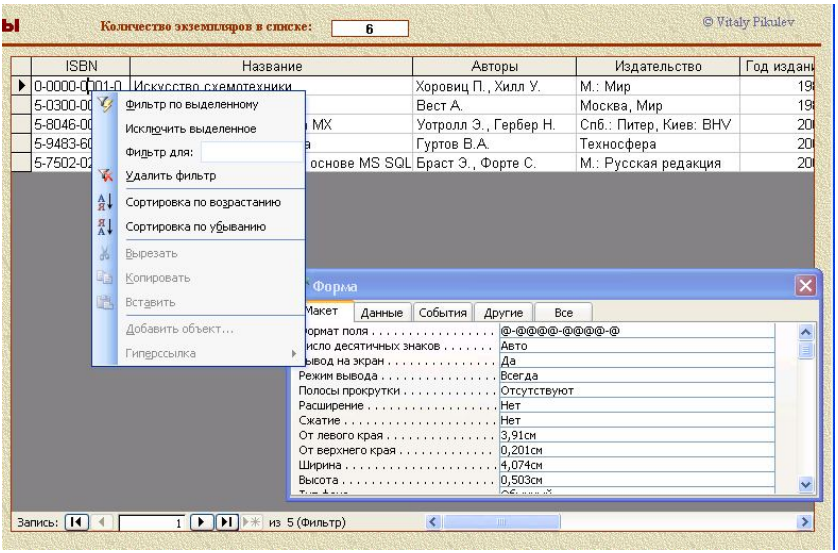


Рис. 2.33. Выпадающее меню сортировки.

Следующим этапом разработки необходимо ввести набор полей и соответствующих им флажков для реализации механизма фильтров для подчинённой формы. Поля с выпадающим списком реализованы аналогично тому, как это было сделано в предыдущей форме, поэтому дополнительные комментарии здесь излишни (рис. 2.34). Автор использовал следующие имена визуальных элементов:

Метка	*Имя поля*	*Имя флажка*
Наличие	var_status	f_status
Особенности	var_feature	f_feature
Раздел	var_genre	f_genre
Подраздел	var_subject	Нет

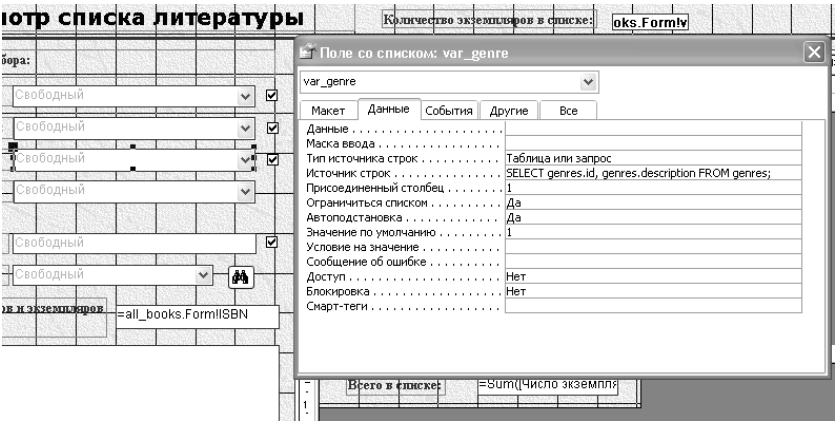


Рис. 2.34. Поля фильтра

Идеология работы фильтра заключается в следующем. Первоначально все флажки убраны и соответствующие им поля неактивны. Как только флажок установлен, поле становится доступным для выбора, а соответствующий фильтр начинает действовать на подчинённую форму. Если установлено несколько флажков, то должны действовать одновременно все условия на выборку. Как это реализовать? Теоретически всё просто: следует динамически (по выбранным значениям фильтра) формировать SQL-

инструкцию и вставлять её в свойство «Фильтр» подчинённой формы. Динамически – т.е. с помощью подпрограмм VBA, подключённых к обработке необходимых событий от указанных элементов управления.

Начнём реализацию фильтра с установки связей между флажками и полями с выпадающими списками. На рис. 2.35 показан фрагмент текста на языке VBA, реализующий соответствующую функциональность между элементами.

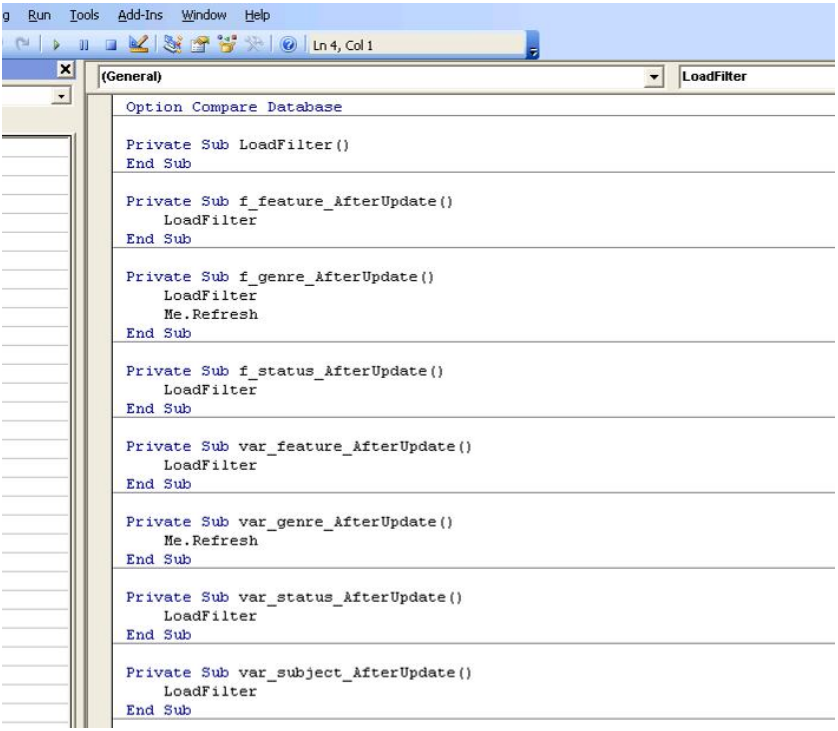


Рис. 2.35. Обработка событий от элементов фильтра

Процедура *LoadFilter* пока пуста. После отладки взаимодействия между визуальными элементами можно начинать писать содержимое этой функции:

```
Private Sub LoadFilter()
    If f_feature.Value = True Then
        var_feature.Enabled = True
        p_feature = " AND all_books.feature=" & var_feature
    Else
        var_feature.Enabled = False
        p_feature = ""
    End If
    If f_status.Value = True Then
        var_status.Enabled = True
        p_status = " AND all_books.status=" & var_status
    Else
        var_status.Enabled = False
        p_status = ""
    End If
    If f_genre.Value = True Then
        var_genre.Enabled = True
        var_subject.Enabled = True
        If var_subject.Value <> Empty Then
            p_subject = " AND all_books.subject=" & var_subject
        End If
    Else
        var_genre.Enabled = False
        var_subject.Enabled = False
        p_subject = ""
    End If
    all_books.Form.Filter = "1=1" & p_feature & p_status & p_subject
    all_books.Form.FilterOn = True
End Sub
```

Выражение «1=1» в текстовой строке имеет отнюдь не философский смысл – оно потребовалось, чтобы при произвольных критериях выбора (каждое начинается с «AND») и при включенном фильтре никогда не возникало бы ошибки формирования раздела WHILE полной SELECT-инструкции. После отладки процедуры *LoadFilter* созданный вами фильтр должен быть полностью работоспособным.

Задание 1 для самостоятельной работы:

Дополните форму поисковой системой, работающей так, как показано на рис. 2.36. Необходимую функциональность можно добавить непосредственно в процедуру *LoadFilter*.

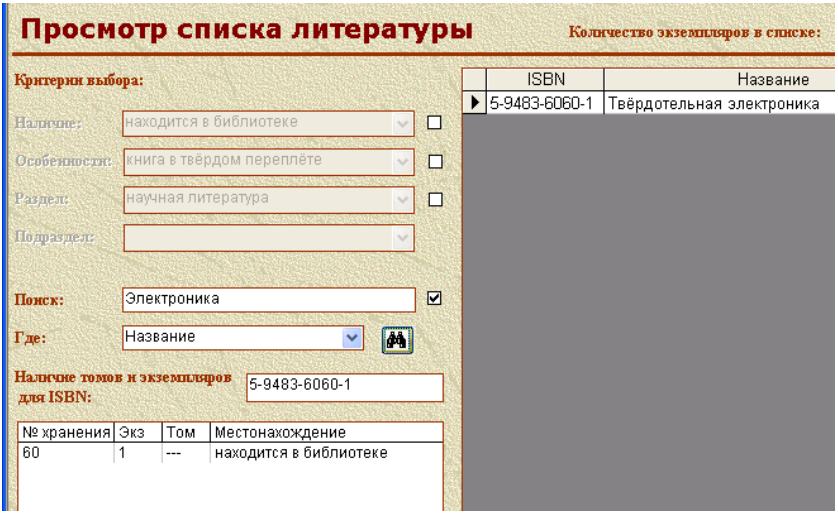


Рис. 2.36. Поисковая система в работе

Недостаток представленной системы фильтрации заключается в том, что критерии «Наличие» и «Особенности» относятся к каждому тому и экземпляру, а выборка в подчинённой форме агрегирует книги по томам и экземплярам. Поэтому если среди нескольких экземпляров одной книги есть один в плохом состоянии – совершенно не очевидно, что соответствующий ISBN будет представлен в результирующем списке.

Детализируем представление информации о книгах в нашей системе, введя дополнительную таблицу в виде многостолбцового списка. Хотелось бы, чтобы при выборе любого поля любой строки в подчинённой форме-таблице в этом списке отображались бы все тома и экземпляры для выбранного ISBN, а также сопутствующая информация (такая как «местонахождение» и «состояние»), см. рис. 2.37.

После создания свободного списка с именем *var_list* следует выбрать источник данных для него. Одним из наиболее простых вариантов является создание дополнительного поля ввода, в которое будет копироваться ISBN из выбранной строки подчинён-

ной формы. Дадим этому полю имя *sel_book*, и определим источник данных для *var_list* так, как показано ниже:

```
SELECT books.id AS [№ хранения], books.specimen AS Экз, IIf(books.volume=0,"---
",books.volume) AS Том, book_status.description AS Местонахождение, fea-
tures.description AS Состояние
FROM features INNER JOIN (book_status INNER JOIN books ON book_status.id =
books.status_id) ON features.id = books.feature_id
WHERE (((books.ISBN)=[sel_book]))
```

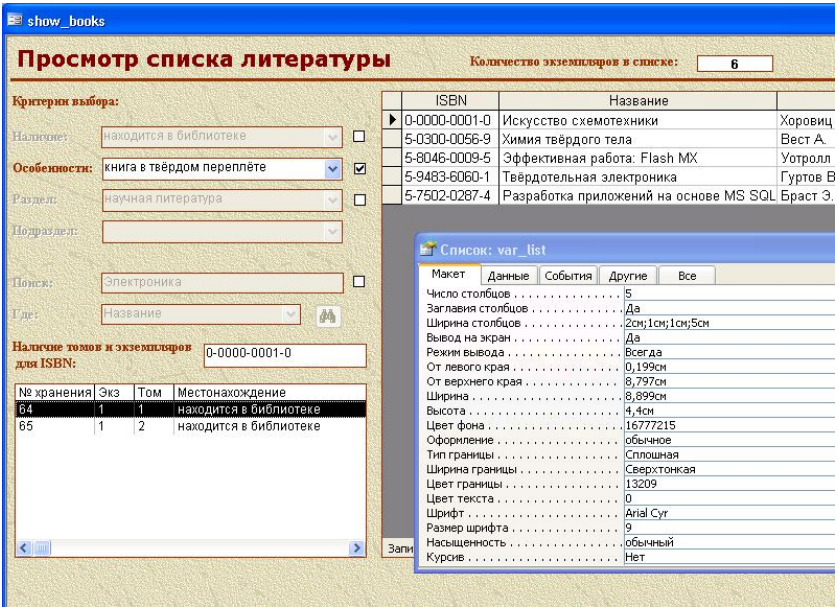


Рис. 2.37. Окно подробной информации о томах и экземплярах

В свою очередь источник данных для поля *sel_book* может выглядеть так, как показано на рис. 2.38. Чтобы точно установить имя поля в вашей базе данных, следует щёлкнуть на многоточие в графе «Данные» и воспользоваться мастером для выбора имён полей и функций.

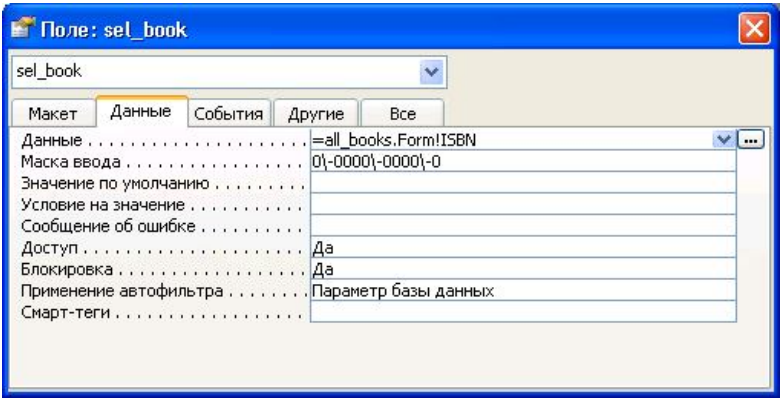


Рис. 2.38. Значения полей для элемента *sel_book*

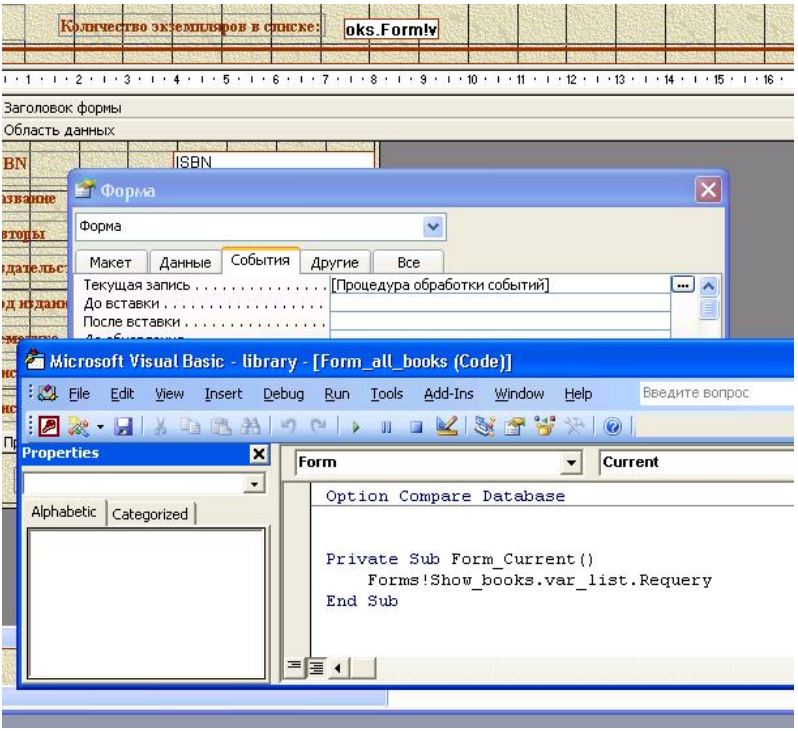


Рис. 2.39. Программирование в подчинённой форме

Однако после этого работа списка ещё не будет скоррелирована с выбором строки в подчинённой форме. Для корректного функционирования поля со списком требуется обработать событие «Текущая запись» подчинённой формы так, как показано на рис. 2.39. Вариант *Requery* для выбранного поля более корректен, нежели *Refresh* для родительской формы, поскольку в последнем случае возможно зацикливание очереди событий.

Введём некоторый элемент удобства в созданную форму: создадим поле «Количество экземпляров в списке», которое будет принимать значение количества книг, выводимых на экран в подчинённой форме, т.е. с учётом поиска и фильтрации. Для этого организуем в главной форме текстовое поле *var_ex*, источником данных для которого будет поле в подчинённой форме *all_books.Form!var_total*. Соответствующее поле в подчинённой форме (в разделе «Примечание формы») можно создать так, как показано на рис. 2.40.

Это дополнительное поле не будет видно в таблице при запуске формы, создание его в разделе «Примечание формы» желательно для правильной работы функции *Sum()*. Выбор *Sum*, а не *Count*, обусловлен тем, что считаются именно количество экземпляров книг, а не количество ISBN в таблице. На практике можно выбрать любой вариант подсчёта литературы в списке.

Предложенный вариант формы не допускает редактирования записей в сводной таблице, поскольку на содержимое действуют функции агрегирования. Для реализации функции редактирования записей можно создать довольно простую и удобную дополнительную форму с именем *edit_books*, воспользовавшись мастером. Форма будет создана с привязкой полей к таблице *books* и может быть настроена на запуск в режиме таблицы. Поля ввода, не имеющие связи с другими таблицами, следует оставить текстовыми, поля – внешние ключи следует заменить на поле с выпадающим списком, как уже не раз было проделано ранее. Добавление новых записей в таблицу в данном случае лучше запретить, поскольку для этого у нас уже есть форма *reg_books*.

В итоге созданная форма в рабочем состоянии должна выглядеть в виде таблицы, представленной на рис. 2.41. Легко проверить, что в ней редактируются все поля.

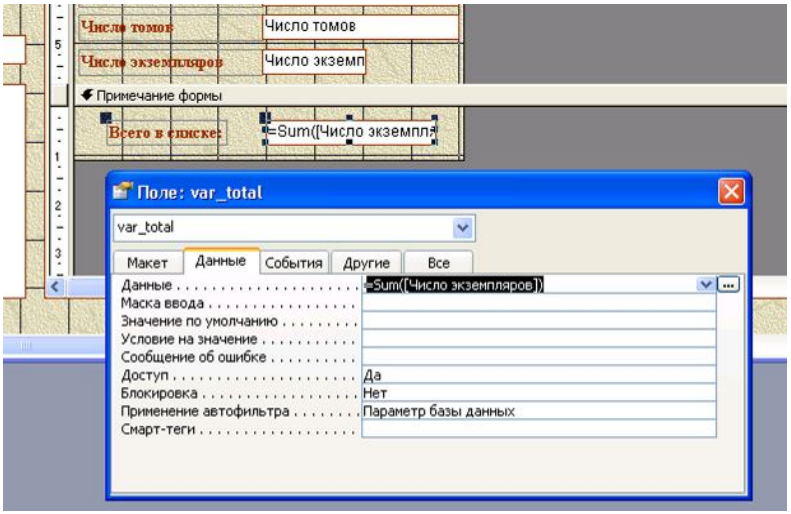


Рис. 2.40. Количество книг в библиотеке

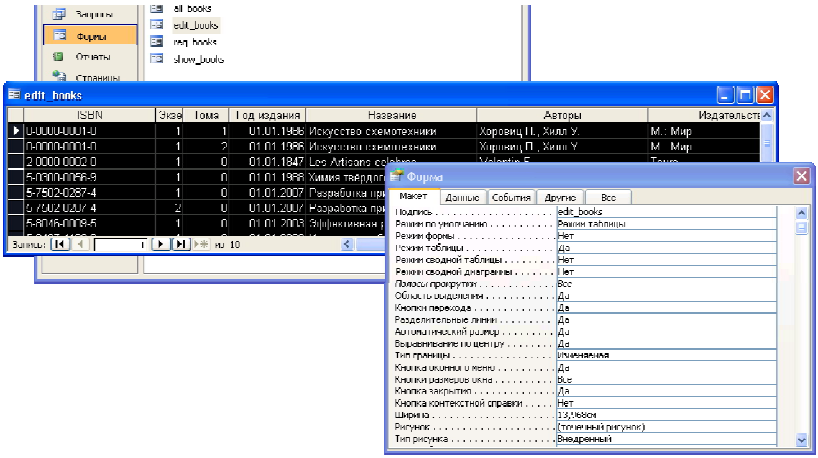


Рис. 2.41. Создание формы для редактирования информации о книгах

Соединим полученную таблицу с формой для просмотра данных. Пусть в форме *edit_books* будут показываться для редактирования только те книги, которые мы выбрали в списке экземпляров и томов двойным щелчком мышки. Обработаем событие двойного щелчка для элемента *var_list* так, как показано ниже:

```
Private Sub var_list_DblClick(Cancel As Integer)
   DoCmd.OpenForm "edit_books", acFormDS, , "books.ISBN='" &
all_books.Form!ISBN & "'"
End Sub
```

После этого разработанная нами форма полностью готова к работе (рис. 2.42).

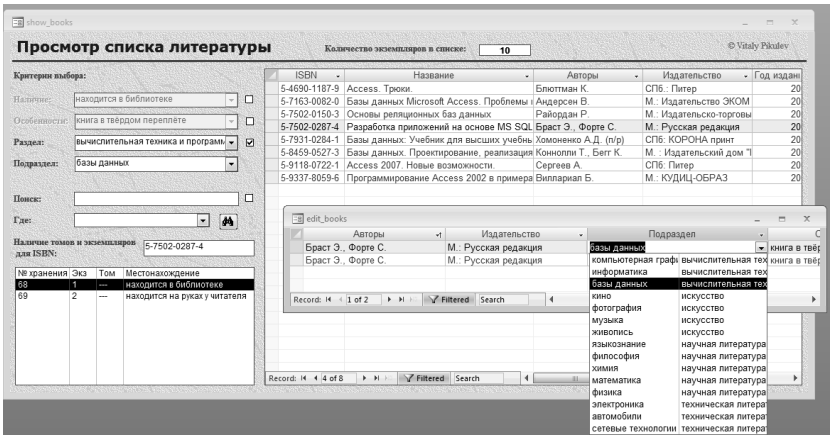


Рис. 2.42. Завершение работы над формой *show_books*

Задание 2 для самостоятельной работы:

Настало время вернуться к первой форме с именем *reg_books*, для которой мы, оказывается, только что разработали подчинённую форму *edit_books*. Итак, на вкладку «Просмотр и редактирование списка книг» следует поместить эту форму (рис. 2.43) и убедиться, насколько удобнее стало вводить и редактировать литературу.

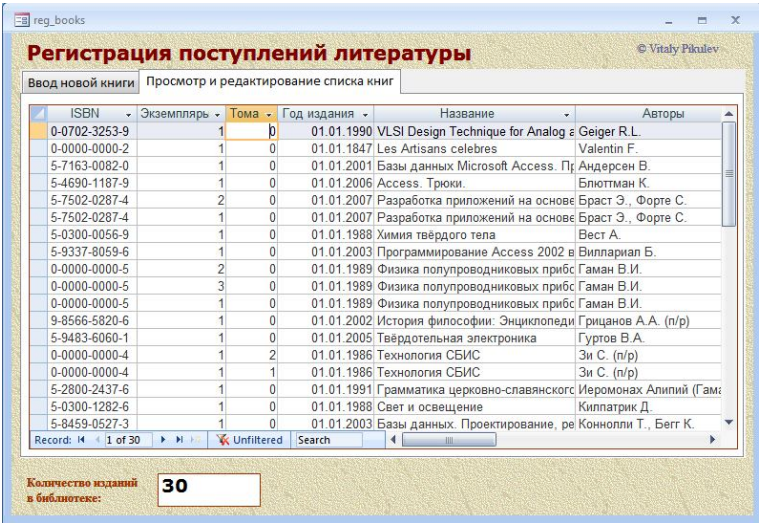


Рис. 2.43. Завершение работы над формой *reg_books*

Однако будет ещё лучше, если вы сделаете копию запроса и подчинённой формы *edit_books* непосредственно для работы с *reg_books*. Почему? Представим, что количество записей в таблице *books* превысило несколько тысяч. Удобно ли будет разбираться с этой таблицей, если вы хотели просто исправить последнюю только что введённую запись? Значит, имеет смысл ограничить вывод количества значений в таблице на вкладке «Просмотр и редактирование списка книг» последними двадцатью или, например, только записями, введёнными сегодня. А для этого потребуется модифицировать запрос.

Задание 3 для самостоятельной работы:

Сейчас, возможно, наступило удобное время поставить пароль на доступ к созданной базе данных и сделать главную разводную форму с кнопками, с помощью которых удобно будет вызывать отдельные формы. Например, так, как показано на рис. 2.44. Предполагается, что в опциях проекта базы данных вы укажете, что именно эта разводная форма будет автоматически запускаться при открытии файла базы данных.

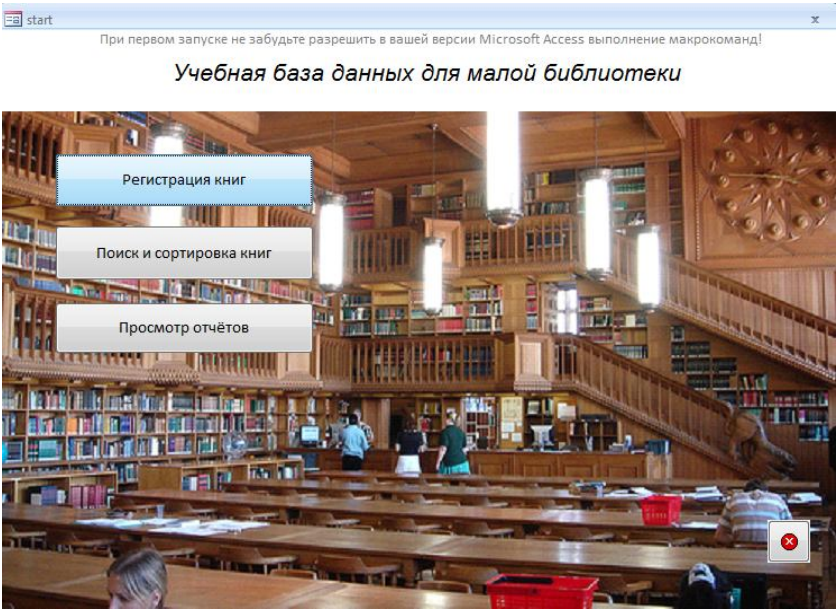


Рис. 2.44. Пример «главной кнопочной формы»

Следующим этапом разработки в Access явится создание набора отчётов, необходимых для получения статистики по функционированию библиотечной системы, однако сейчас нам потребуется временно оставить это направление и выйти за пределы среды разработки Microsoft Access, оставаясь в то же время в рамках разрабатываемой базы данных и в границах соответствующей предметной области.

Глава 3. РАЗВИТИЕ ПРОЕКТА В СРЕДЕ ПРОГРАММИРОВАНИЯ MICROSOFT VISUAL STUDIO

3.1. Общие сведения о создании приложений баз данных с использованием среды Visual Studio .NET

Какими бы изысканными не были возможности интерфейса СУБД, заинтересованные в конечном результате программисты стремятся к созданию приложений, обладающих большими функциональными возможностями, удобством, универсальностью и, по возможности, работающих без привязки к окну Access. В то же время вполне понятно желание потратить на рутинное программирование как можно меньше сил и времени. Разумным компромиссом в этой дилемме может стать использование возможностей среды программирования Visual Studio .NET, поскольку её элементы, очевидно, наиболее совместимы с идеологией программирования от Microsoft и максимально приспособлены для работы с БД MS Access и SQL Server.

Хорошо известно [12], что в состав библиотеки Microsoft NET.Framework входят библиотеки (и соответствующие классы) для подключения разнообразных источников данных, вместе именуемые ADO.NET (ActiveX Data Objects for NET technology). Как и предшествующие технологии, ADO.NET с равным успехом обеспечивает доступ как к удалённым серверам баз данных, так и к локальным или сетевым ресурсам, порой совсем не попадающим под определение баз данных. Наиболее известный пример – реализация обработки данных в XML-файлах с возможностью интеграции этих данных в реляционные структуры.

Поставщиком данных обычно является драйвер, предназначенный для взаимодействия с источником данных определенного типа, доступ к которому реализован через набор классов. Библиотеки .NET Framework включают два основных поставщика – OLE DB .NET Data Provider и SQL Client .NET Data Provider. Поставщик OLE DB .NET Data Provider позволяет взаимодействовать с различными хранилищами данных посредством универсальной технологии Objects Linking and Embedding (связывание и внедрение объектов в другие документы и объекты). Поставщик SQL

Client .NET Data Provider рассчитан исключительно на взаимодействие с БД SQL MS SQL Server, начиная с версии 7.

Важной особенностью использования доступа к БД, которую предоставляют эти библиотеки, является отсоединенный кэш данных, предназначенный для автономной работы с данными. Так, например, любые данные, выбранные из БД и помещенные в объект *DataTable*, могут считаться отсоединенными (disconnected) от источника (сервера). Содержимое *DataTable* можно просматривать, не генерируя сетевого трафика между ADO.NET и БД. Вместо того чтобы многократно обращаться к серверу и выбирать похожие наборы данных из нескольких таблиц, можно один раз перенести все необходимые таблицы в объект *DataSet*, который можно представить как контейнер объектов *DataTable* и иной инфраструктуры, позволяющей создавать и поддерживать ссылочную целостность между таблицами и определять регламент доступа к ним.

И, наконец, следует отметить языковую универсальность библиотек ADO.NET: поскольку Visual Studio .NET является общеязыковой средой разработки программ, то и модель доступа к данным не зависит от лингвистических особенностей исполнения программы или её отдельного модуля.

В среде Visual Studio существуют различные способы привязки элементов управления к данным, однако в любом случае вначале определяется источник данных, затем устанавливается соответствие между данными в источнике (обычно пишется SQL-запрос) и свойствами элементов управления, размещаемых на формах. В случае, если привязка данных осуществлена корректно, визуальные конструкторы среды практически автоматически генерируют правильные элементы управления. Неудачная привязка данных к элементу управления может нарушить функционирование всего проекта. В этой связи настоятельно рекомендуется архивировать текущие версии разработки вашего проекта.

3.2. Организация проекта Visual Studio и установление связи с файлом базы данных Access

В ходе работы за несколько этапов будет реализована интерфейсная программа, которую условно можно назвать «рабочим

местом библиотекаря». Основными функциями программы являются:

– просмотр и редактирование «карточки читателя», где записана вся информация о выдаваемых и возвращаемых книгах;

– просмотр статистики читателя (количество книг на руках, количество просроченных книг и т.д.);

– регистрация выдачи/приёма книг.

Для разработки программы будет использован язык Visual Basic.NET (среда Visual Studio 2008). Внешний вид окна программы представлен на рис. 3.1.

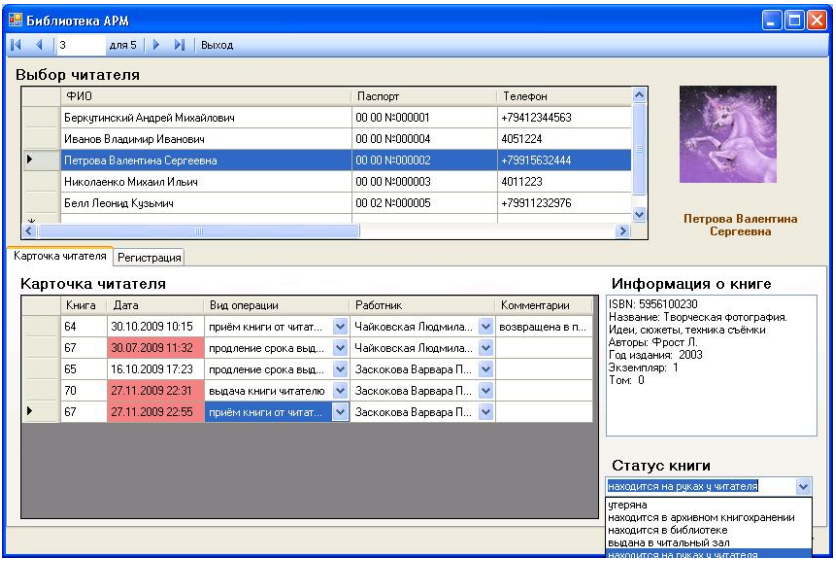


Рис. 3.1. Главное окно программы

Предварительно, для обеспечения возможности тестирования разрабатываемой программы, следует вручную (пользуясь штатными средствами *Access*) внести в таблицу *users* несколько записей о читателях библиотеки. Обратите внимание, что программа может выводить на экран фотографии пользователей (в представленном варианте – «аватары», сохранённые в соответствующих ячейках таблицы). Рекомендуется сохранять соответствующие изображения в виде «объект: точечный рисунок», ис-

пользуя выпадающее меню «добавить объект…» при редактировании столбца *photo* таблицы *users*. Также, для предварительного тестирования, следует вручную добавить несколько записей по приёму/выдаче книг в таблицу *actions*. Это связано с тем, что при разработке программы на Visual Studio сначала проще научиться читать данные из базы, нежели добавлять и изменять соответствующие записи.

Обычно для осуществления операции доступа к внешним источникам данных используются либо интерфейс ODBC, либо интерфейс ADO (OLE). Учитывая, что последний отличается более высокой скоростью и более широкими функциональными возможностями, будем использовать его. Для организации проекта воспользуемся стандартным окном Visual Studio (рис. 3.2).

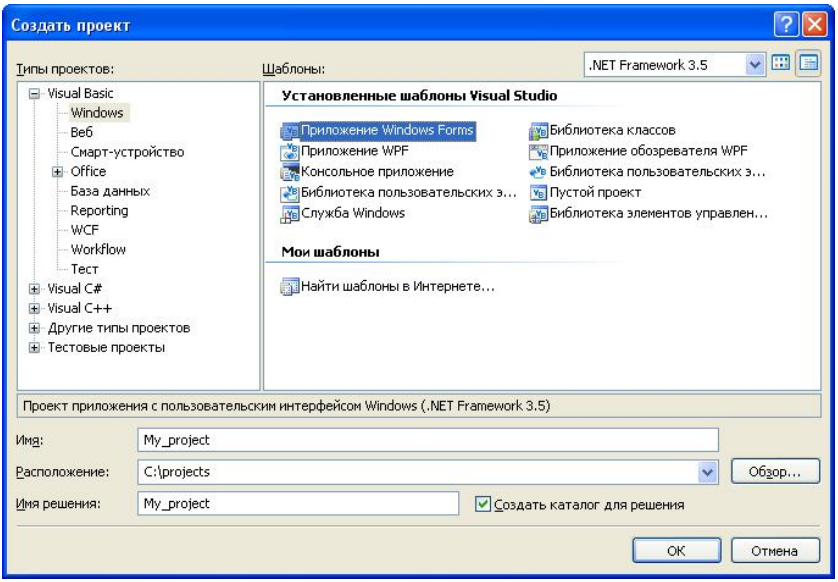


Рис. 3.2. Окно создания проекта

Представленный далее проект был назван «*Library*», в этой связи данное имя вошло в состав большинства имён, автоматически предлагаемых системой в ходе разработки проекта. В случае разработки вашего проекта с собственным именем это обстоя-

тельство следует учитывать при применении текстов запросов и программ данного пособия.

Для подключения файла базы данных Microsoft Access следует воспользоваться пунктом меню «*Данные / Добавить новый источник данных*» и пройти предлагаемые мастером шаги. Если была введена защита на доступ к данным, следует использовать имя *Admin* и известный вам пароль. Для облегчения работы с базой можно согласиться с введением конфиденциальных данных в строку подключения. При этом пароль будет в явном виде присутствовать в строке подключения, которая хранится в файле *app.config* в папке проекта.

Рекомендуется в файле конфигурации хранить реальный путь к базе данных (например, начиная с корня диска C: или из каталога проекта). В обозревателе серверов Visual Studio можно посмотреть структуру и содержание используемой базы данных (рис. 3.3).

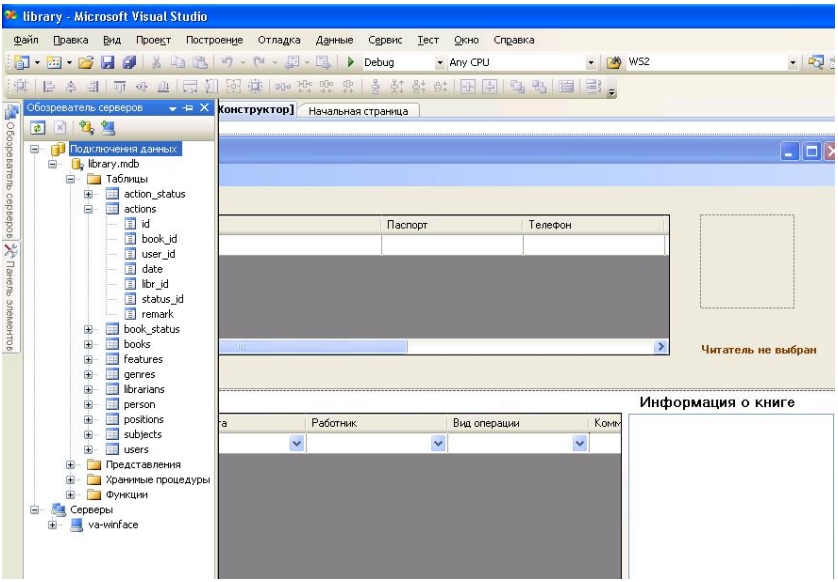


Рис. 3.3. Просмотр базы данных

3.3. Организация доступа к данным

Следующий шаг, необходимый для работы с данными, заключается в организации согласованного набора данных *DataSet*, используемого для промежуточной обработки данных. Фактически *DataSet* может играть роль суженного или расширенного (по сравнению с исходной базой данных) набора таблиц, необходимых для корректного функционирования разрабатываемых визуальных элементов, опосредуя связь с таблицами базы данных, в том числе с целью записи и изменения данных в базе. Структура *DataSet* схожа со структурой реляционной базы данных: она представляет собой иерархическую объектную модель таблиц, строк, столбцов, ограничений и связей. Можно работать с данными в наборе данных, даже если приложение временно отключается от базы данных.

Структуру *DataSet* можно формировать постепенно, по мере программирования новых визуальных компонентов в формах. Не рекомендуется сразу копировать весь набор таблиц БД MS Access в *DataSet*, в нашем случае это может привести к неожиданным ошибкам при работе со средой программирования.

Начнём с создания таблицы, необходимой для функционирования компонента *DataGridView*, содержащего данные о читателях. Для этого вытащим из панели инструментов в область формы каждый из перечисленных выше компонентов (рис. 3.4).

После создания пустого *LibraryDataSet* файл с соответствующим именем (и расширением *.xsd*) появится в обозревателе решений. Его можно открыть и начать формирование таблиц данных с помощью контекстного меню (рис. 3.5).

После запуска мастера следует согласиться с предложенными условиями доступа к базе данных и добраться до кнопки «Построитель запросов». С помощью интерфейса построителя запросов можно сформировать и протестировать запрос, например так, как показано на рис. 3.6. Предварительно задаются таблицы из базы данных, из которых делается выборка.

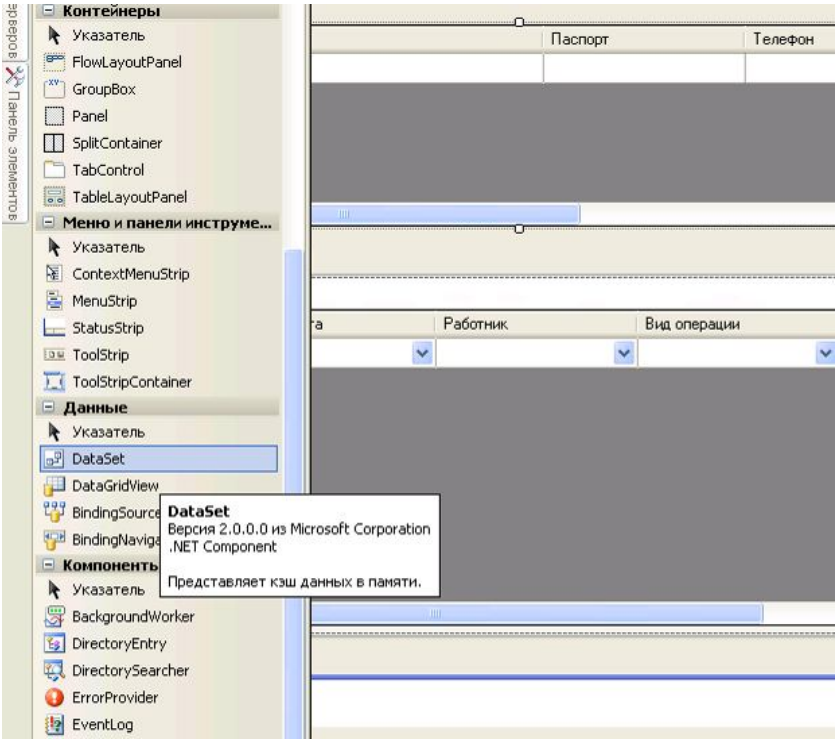


Рис. 3.4. Набор инструментов для работы с базами данных

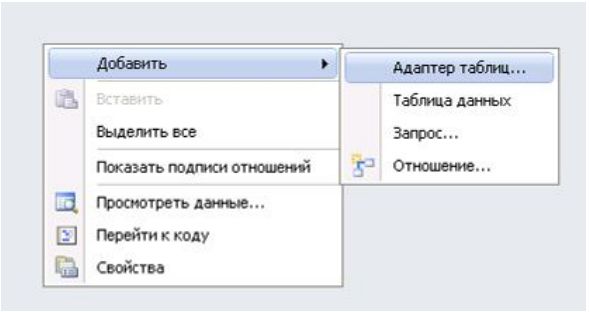


Рис. 3.5. Для формирования таблиц данных используется пункт меню «Адаптер таблиц»

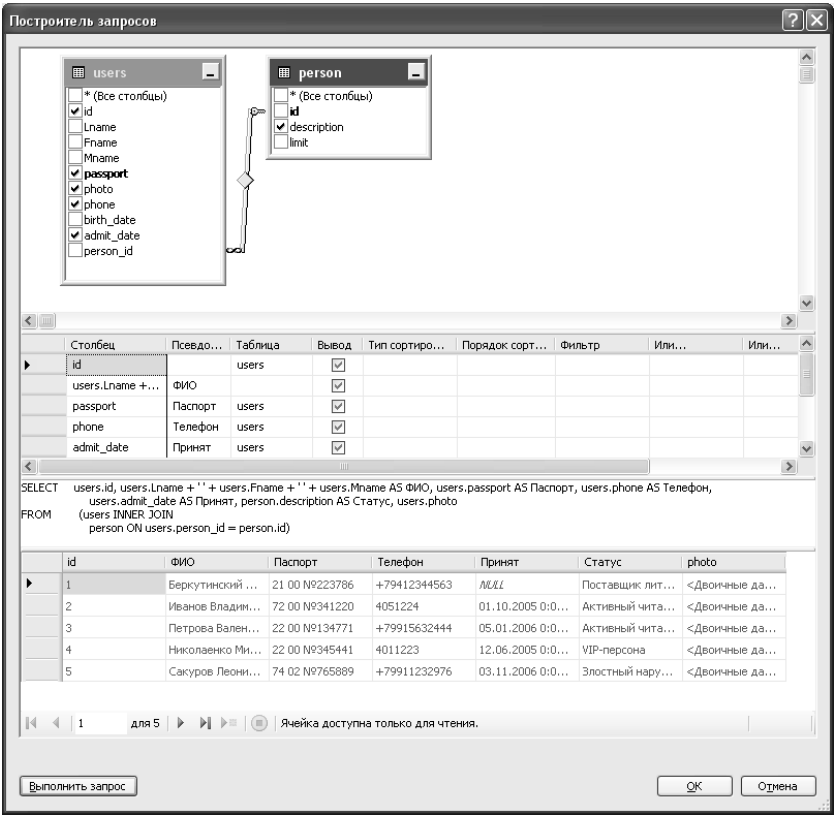


Рис. 3.6. Окно построителя запросов

Особенности работы с построителем запросов должны быть хорошо понятны из приведенного рисунка. Текст запроса пишется редактором автоматически, при необходимости корректируется вручную. Рекомендуется в случае серьёзных изменений текста запроса сохранять текст в буфере обмена, поскольку при проверке корректности запроса система может серьёзно изменить ваш текст, вплоть до полного уничтожения сделанных вами изменений.

В итоге в *LibraryDataSet* появится таблица, подобная представленной на рис. 3.7.

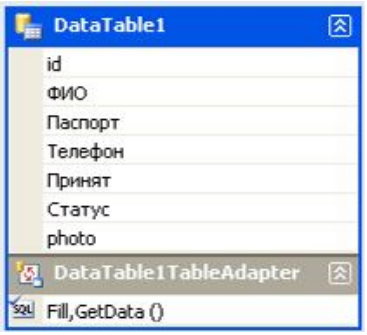


Рис. 3.7. Новая таблица в наборе данных

Компонент *DataGridView*, который вы разместите на форме, является одним из наиболее функционально богатых элементов управления. Он позволяет отображать данные в табличном виде в разных вариантах из различных типов источников данных, а также сортировать, изменять и добавлять данные. Например, можно программно задать собственные алгоритмы сортировки, а также создать собственные типы ячеек. Однако из широкого набора возможностей данного элемента нам необходимы только те, которые удобны для решения нашей задачи.

В первую очередь необходимо привязать экземпляр *DataGridView1* (здесь и далее используются имена, автоматически подставляемые системой) к данным из *LibraryDataSet*. Это можно сделать так, как показано на рис. 3.8. В качестве источника данных выбирается только что созданная нами таблица. После привязки соответствующая структура столбцов автоматически возникает внутри элемента, и после запуска приложения можно уже просматривать готовую таблицу пользователей, а также формально разрешить редактирование, удаление и добавление строк.

Фактически созданная нами таблица неудобна для редактирования, и механизмы удаления, добавления и изменения строк в нашем случае не будут задействованы автоматически (но их, при желании, можно запрограммировать вручную). При необходимости исправления числа и свойств столбцов (например, столбец *photo* в таблице не задействован) используйте пункт «Правка столбцов…» в боковом меню элемента (рис. 3.9).

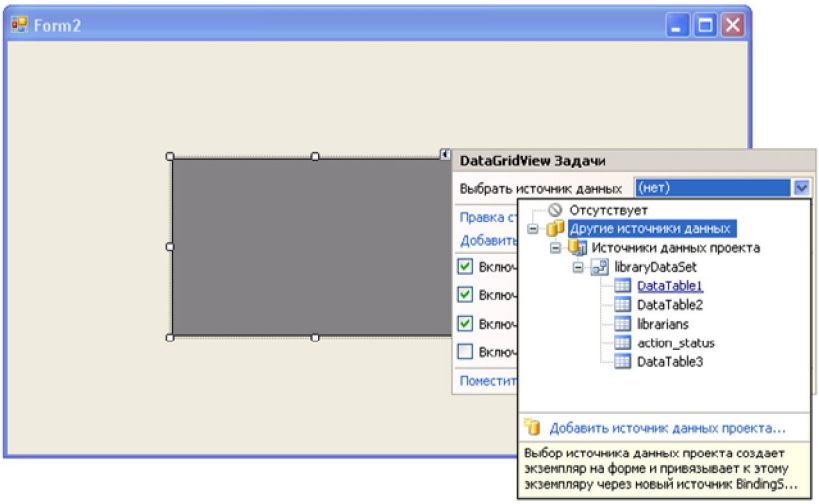


Рис. 3.8. Связь визуальных элементов с набором данных

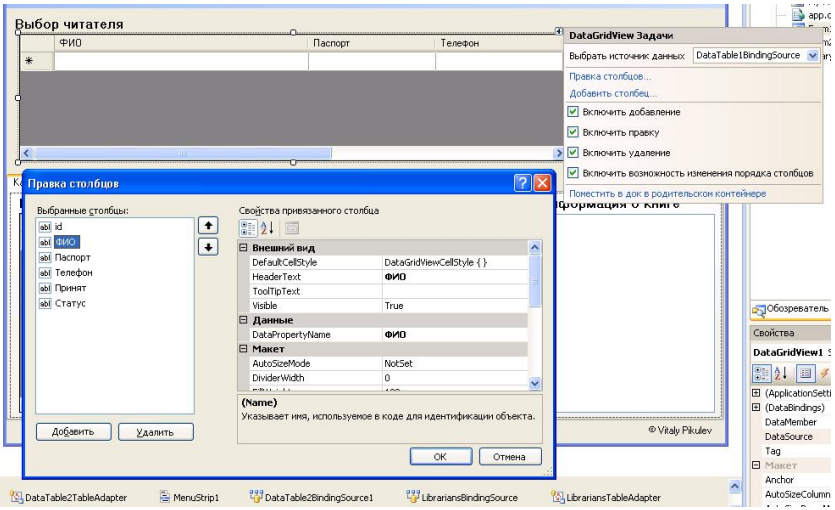


Рис. 3.9. Редактирование *DataGridView*

После запуска приложения компонент уже вполне готов для работы с данными, а именно с таблицей *DataTable1*. При этом в «подвале» конструктора форм Visual Basic автоматически появились компоненты *DataTable1BindingSource* и *DataTable1-TableAdapter*, с которыми можно работать и из других визуальных элементов формы, а в тексте программы, привязанной к соответствующей форме, начала автоматически заполняться процедура *Form1_Load*.

Для компонента *DataGridView1* в окошке свойств следует установить свойство *SelectionMode* в **FullRowSelect**, свойство *Multiselect* сделать **False**, а свойство *ReadOnly* сделать **True**.

Добавим несколько элементов дизайна в нашу форму. В первую очередь, хотелось бы увидеть фотографию выбранного в таблице пользователя. Для реализации этой функции воспользуемся компонентом *PictureBox*, перенеся его с панели инструментов на форму. Запрограммируем свойство *DataBinding* компонента так, как показано на рис. 3.10.

Этого будет достаточно, чтобы компонент начал корректно функционировать, когда мы выбираем пользователя в «таблице» *DataGridView1*. Возможные проблемы обычно бывают связаны только с неправильным форматом сохранения изображений в базе данных, т.е. изображение должно храниться в стандартном графическом формате (например, Microsoft bitmap), но не в виде OLE-объекта.

Точно таким же образом поступим с компонентом *Label*, который будет показывать ФИО читателя, выбранного в данный момент (рис. 3.11).

И так же поступим с компонентом *BindingNavigator* (полоска в верхней части формы), убрав из него ненужные кнопки и добавив полезные (например, кнопку «Выход»). На рис. 3.12 и 3.13 показаны особенности программирования данного компонента. Особой функциональной значимости этот компонент не имеет, но эстетики в программу, возможно, добавляет. Теперь при выборе читателя можно использовать либо таблицу *DataGridView1*, либо кнопки этого компонента.

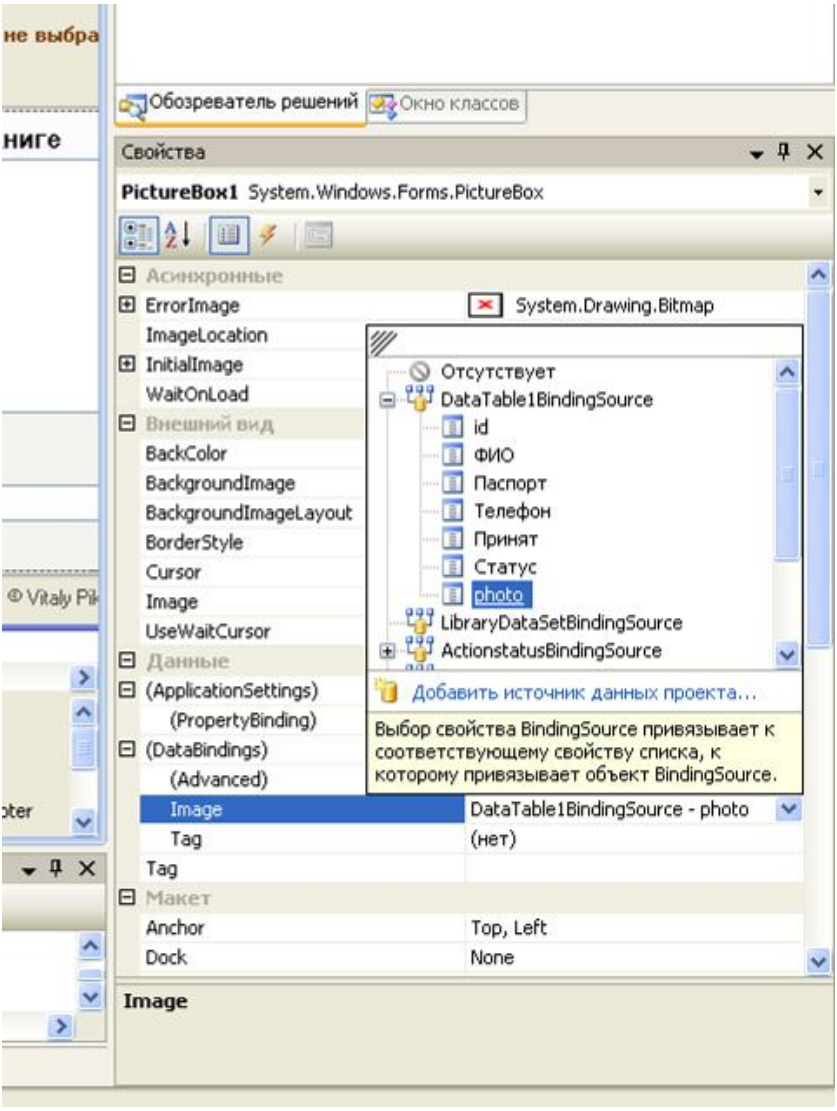


Рис. 3.10. Программирование вывода изображений

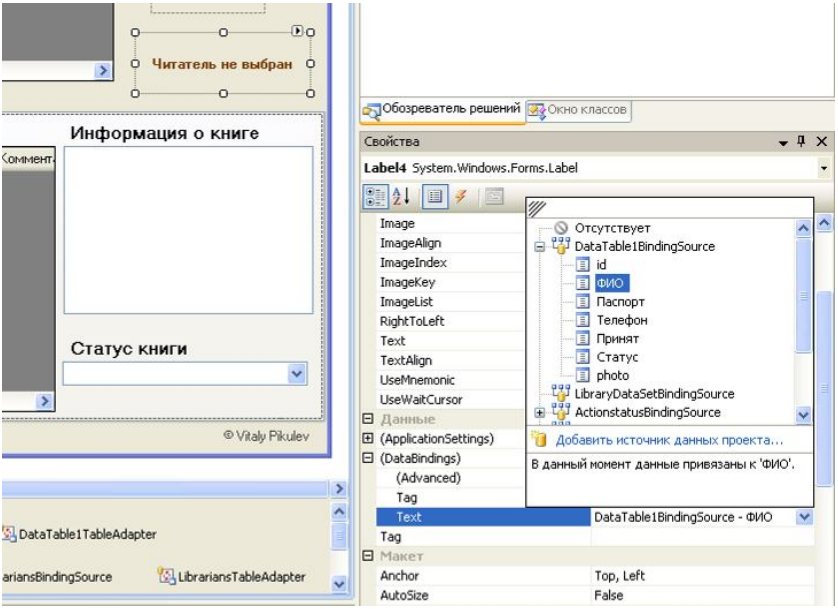


Рис. 3.11. Программирование вывода информации о читателе

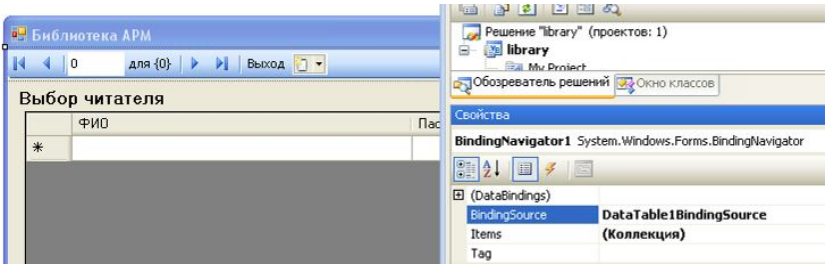


Рис. 3.12. Привязка компонента к данным

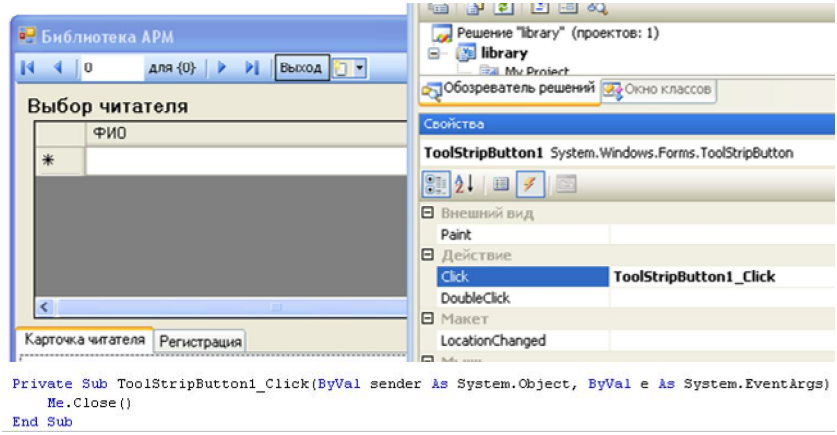


Рис. 3.13. Обработка события *Click* для кнопки «Выход»

3.4. Разработка «Карточки читателя»

Расположенная в нижней части формы (см. рис. 3.1) карточка читателя показывает статистику выдачи/возврата книг для выбранного читателя. В карточке, например, отмечаются строки с просроченными книгами. При выборе строки появляется полная информация о книге, взятой или возвращённой читателем. Ряд полей (или все поля – по желанию) можно сделать редактируемыми.

Для создания соответствующего интерфейса воспользуемся компонентом *TabControl*, экземпляр которого расположим в нижней половине разрабатываемой формы. Создадим первую вкладку с именем «Карточка читателя», на которой разместим экземпляр компонента *DataGridView2*, многострочный элемент для вывода текста *TextBox1* и *ComboBox1* для представления сведений о местонахождении книги.

Для компонента *DataGridView2* разработаем запрос, который, с одной стороны, будет привязан к таблице *DataGridView1* и позволит формировать данные для выбранного пользователя, с другой стороны, позволит редактировать таблицу *Actions* (т.е. будет использовать столбцы данных, точно соответствующие

данной таблице) и, кроме того, будет поставлять готовые данные для *TextBox1* и *ComboBox1*.

Ниже приведен вариант предлагаемого запроса:

```
SELECT    actions.id, actions.[date], actions.libr_id, actions.status_id,
actions.remark, actions.book_id, 'ISBN: ' + books.ISBN + chr(13) + chr(10)
+ 'Название: ' + books.title + chr(13) + chr(10) + 'Авторы: ' + books.authors + chr(13)
+ chr(10) + 'Год издания: ' + str(datepart('yyyy', books.pub_year))
+ chr(13) + chr(10) + 'Экземпляр: ' + str(books.specimen) + chr(13) + chr(10) + 'Том: '
+ str(books.volume) AS KeyInfo, books.status_id AS book_status
FROM      (actions INNER JOIN books ON actions.book_id = books.id)
WHERE    (actions.user_id = ?)
```

«Огромный» текст для формирования столбца *book_status* – это всего лишь полная информация о книге, которая представляется как набор символов для поля *TextBox1*. Более интригующе выглядит строка «*action.user_id=?*». Таким способом указывается на использование *параметров* в запросах ADO.NET. Сколько параметров необходимо использовать – столько знаков вопроса в запросе следует поставить. В нашем случае параметр один, ему автоматически присвоено имя *user_id* (рис. 3.14). Использовать этот параметр можно так, как показано ниже (параметру *user_id* присваивается значение *c_i*).

```
Private Sub DataGridView1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles DataGridView1.Click
    Dim c_i As Integer
    Dim Row As DataGridViewRow
    For Each Row In DataGridView1.SelectedRows
      c_i = Row.Cells(0).Value
    DataTable2TableAdapter.Fill(Me.LibraryDataSet.DataTable2, c_i)
    Next
End Sub
```

Тест данной процедуры работоспособен даже при разрешении (по умолчанию) множественного выбора ячеек в *DataGridView1*. Однако поскольку редактирование записей читателей у нас не запланировано, будет удобно разрешить выбор только одной строки (как представлено на рис. 3.15).

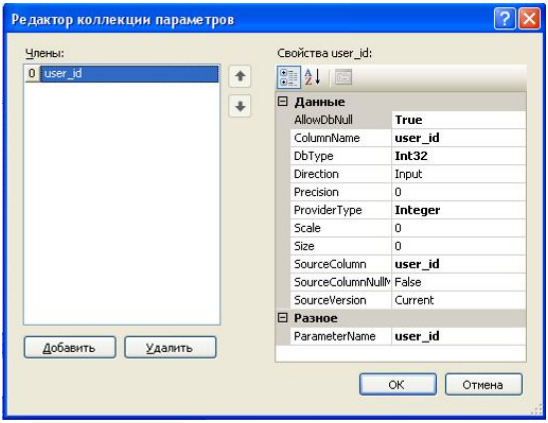


Рис. 3.14. Редактор параметров при необходимости может быть вызван из панели «Свойства» для страницы LibraryDataSet.xsd

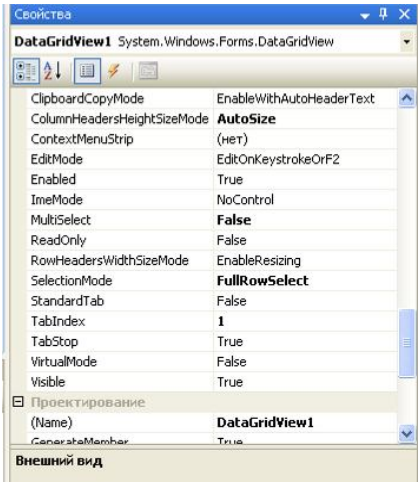


Рис. 3.15. Изменение параметров отображения данных

В свою очередь, свойства столбцов компонента *DataGridView2* программируются так, как показано на рис. 3.16. Так, первый и последний столбцы сделаны невидимыми, второй столбец запрещён для редактирования, а четвёртый и пятый столбцы сделаны в виде выпадающих списков, привязанных соответственно к таблицам *action_status* и *librarians* (рис. 3.17).

Конечно, для того чтобы привязать выпадающие списки к этим таблицам, необходимо иметь образы указанных таблиц в *LibraryDataSet*. Их следует создать, вновь воспользовавшись адаптером таблиц (рис. 3.18), используя запросы, сходные с представленными ранее в п. 3.3. Как только к выбранному объекту *LibraryDataSet* подключается какой-либо из созданных вами компонентов, автоматически генерируется соответствующий *BindingSource* и помещается в «подвал» конструктора форм. Функционирование созданных выпадающих списков показано на рис. 3.19.

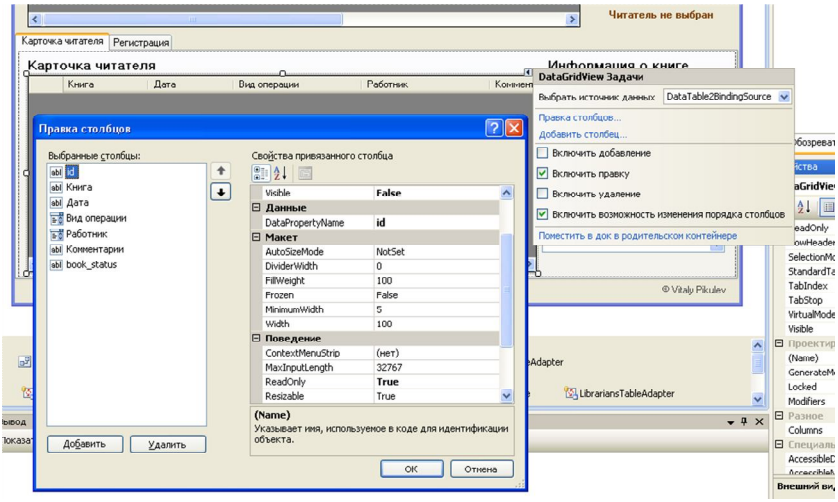


Рис. 3.16. Программирование *DataGridView2*

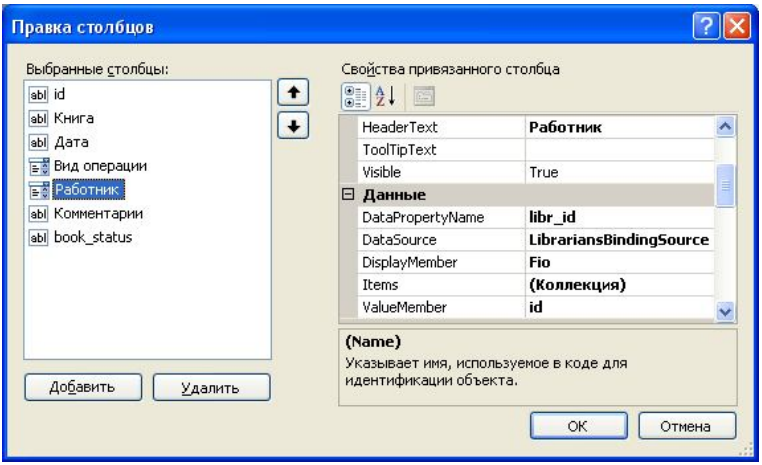


Рис. 3.17. Программирование свойств столбцов

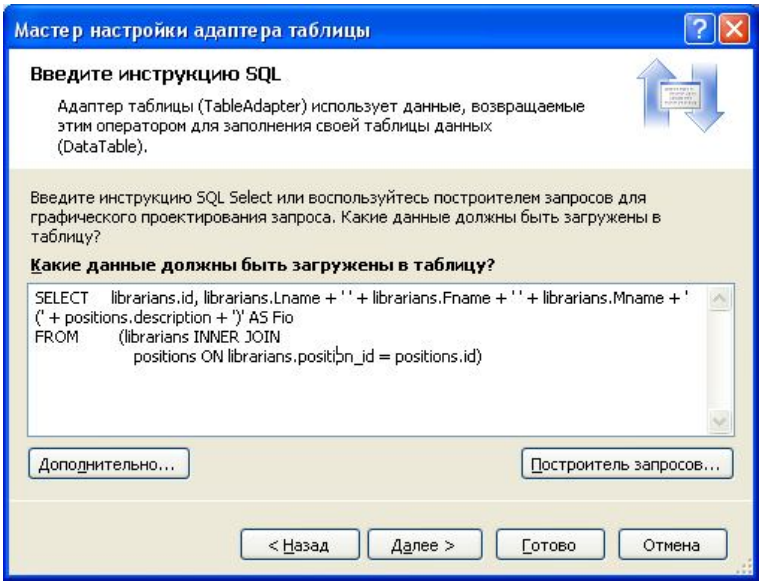


Рис. 3.18. Создание таблиц в *DataSet*

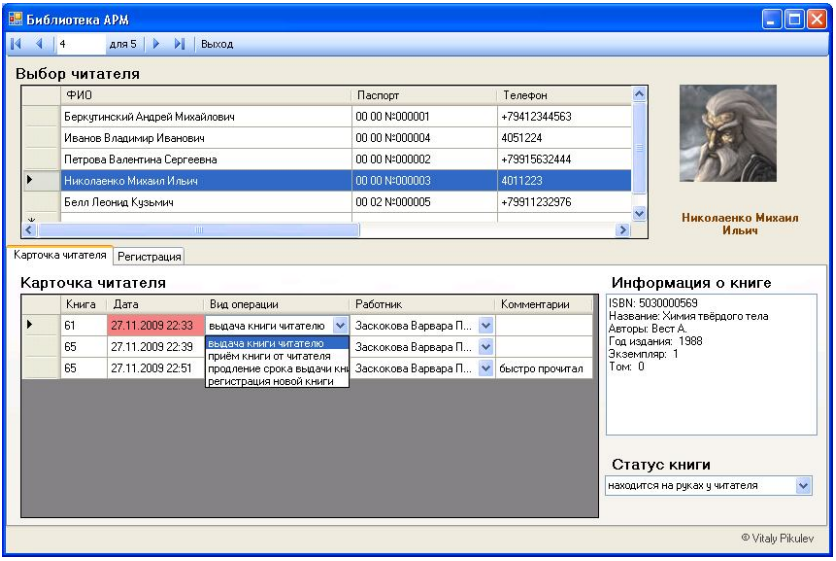


Рис. 3.19. Формирование выпадающих списков в *DataGridView2*

Задание 4 для самостоятельной работы:

На данном этапе должен быть совершенно очевиден механизм подключения элементов «Информация о книге» и «Статус книги», которые планировалось разместить в форме. Сделайте это самостоятельно. Проверьте работоспособность всех созданных вами компонентов.

Теперь включим возможность отправления отредактированных данных в таблицы Microsoft Access. Для этого в *LibraryDataSet* в свойствах таблицы *DataTable2TableAdapter* для свойства *UpdateCommand* укажите «(Новый)» и перейдите к редактированию запроса на изменение данных, как показано на рис. 3.20. Автоматически будет создана коллекция из пяти параметров, которые будут привязаны к одноимённым столбцам. Но выполняться обновление данных в базовых таблицах автоматически не будет.

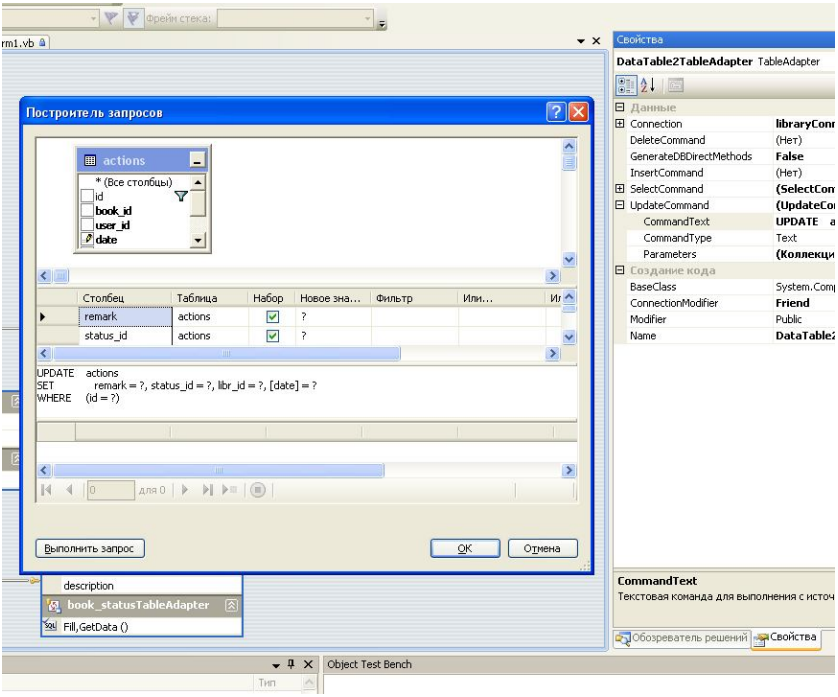


Рис. 3.20. Создание запроса на изменение данных

Для запуска обновления следует обработать события, связанные с редактированием данных, например так, как показано ниже:

```
  Private Sub DataGridView2_CellEndEdit(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.DataGridViewCellEventArgs) Handles
DataGridView2.CellEndEdit
      DataTable2BindingSource.EndEdit()
      DataTable2TableAdapter.Update(LibraryDataSet.DataTable2)
  End Sub

  Private Sub ComboBox1_SelectionChangeCommitted(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ComboBox1.SelectionChangeCommitted
      BooksTableAdapter.Update(ComboBox1.SelectedValue,
DataGridView2.CurrentRow.Cells(1).Value)
  End Sub
```

Следует обратить внимание, что предложенный запрос *Update* изменяет только поля в таблице *action*. А как же быть с полем *status_id* таблицы *book*, которое также хотелось бы уметь изменять в этом же окне с помощью элемента *ComboBox1* («Статус книги»)?

Для этого в *LibraryDataSet* можно создать аналог таблицы *books* (SELECT * FROM books) и для появившегося *booksTableAdapter* сформировать, как было сделано выше, запрос на изменение данных (UPDATE books SET status_id = ? WHERE id = ?). Извлечь необходимую пару параметров из компонентов формы можно с помощью приёма, показанного в вышеприведенном программном коде.

И последний «штрих» данного этапа. Если поставить задачу закрасить в разный цвет ячейки дат в таблице «Карточка читателя», в зависимости от того, сильно ли просрочен возврат литературы в библиотеку у выбранного пользователя, то это уже не должно вызвать каких-либо проблем с программированием. Представленный ниже код иллюстрирует лишь один из возможных вариантов, как это можно сделать.

```
Private Sub DataGridView2_CellFormatting(ByVal sender As System.Object, ByVal e
As System.Windows.Forms.DataGridViewCellFormattingEventArgs) Handles
DataGridView2.CellFormatting
    If e.ColumnIndex <> 2 Then Return
    If DataGridView2.Rows(e.RowIndex).Cells(6).Value > 3 Then
      If DateDiff(DateInterval.Day, CDate(e.Value), Now) > 30 Then
        e.CellStyle.BackColor = Color.LemonChiffon
      End If
      If DateDiff(DateInterval.Day, CDate(e.Value), Now) > 60 Then
        e.CellStyle.BackColor = Color.LightCoral
      End If
    End If
  End Sub
```

3.5. Разработка вкладки регистрации приёма-выдачи книг

В соответствии с целью разработки данного приложения функция регистрации приёма-выдачи книг является главной це-

лью данной программы. Для её реализации потребуются полный список книг с возможностью их быстрого выбора (фильтрации и сортировки) и две кнопки (рис. 3.21), изменяющие статус нахождения книги в библиотеке.

Вкладка регистрации состоит из элемента *DataGridView3*, связанного с новым адаптером *Datatable3* (только чтение данных) в *LibraryDataSet*, текстового поля для содержания фильтра *var_find*, поля с выпадающим списком *var_field*, в котором перечислены наименования столбцов, по которым можно вести фильтрацию, агрегирующую рамку *Panel1*, выполняющую в основном декоративные функции. Внутри рамки находятся кнопки *Button1* («Выдать книгу») и *Button2* («Принять книгу»), поле с выпадающим списком «Библиотекарь» (*var_libr*), а также метка *Label7*, в которой отображается текущая дата, также имеющая пока чисто декоративное назначение.

Начнём формирование функциональности у элементов с уже достаточно знакомой процедуры формирования новой таблицы в *LibraryDataSet*. Пример запроса приведен на рис. 3.22.

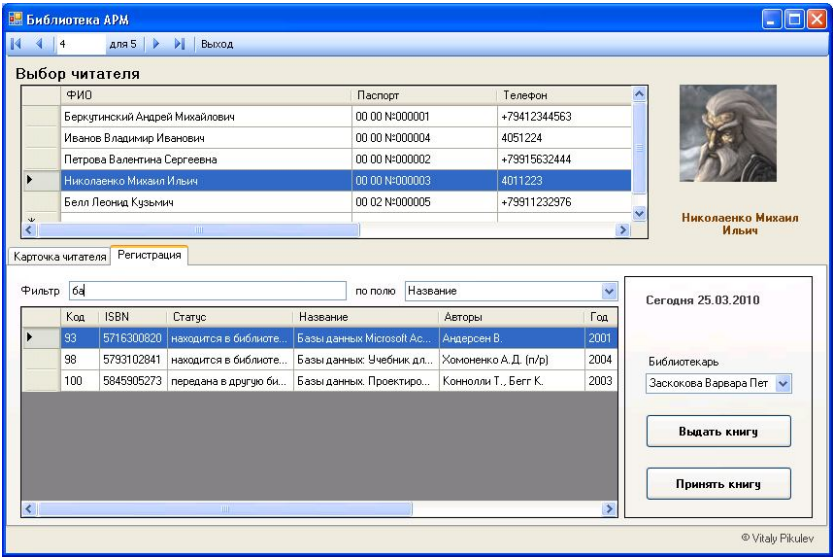


Рис. 3.21. Вкладка регистрации книг в рабочем состоянии

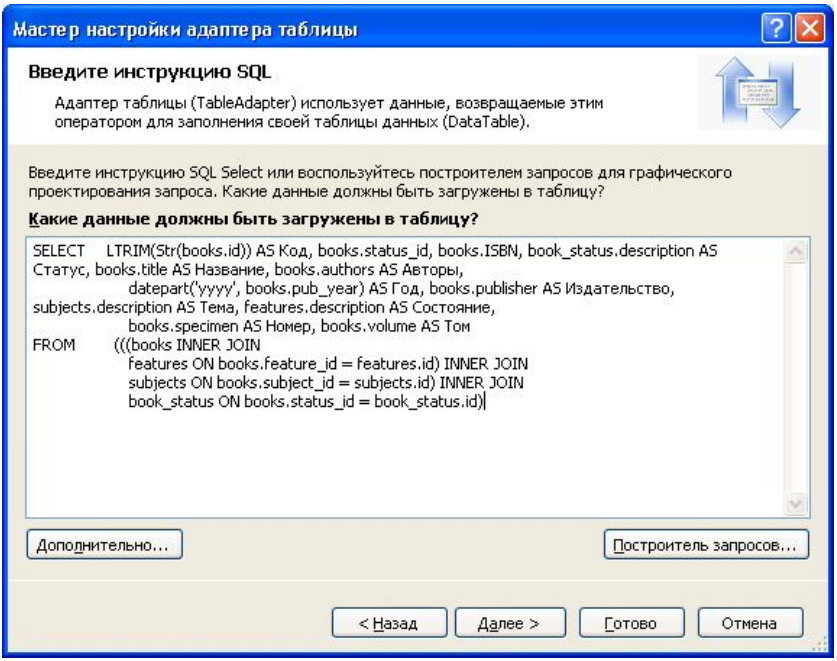


Рис. 3.22. Запрос на выборку для списка книг

Это ещё один (и не последний!) вариант интегральной выборки по списку литературы, который отличается от всех предыдущих тем, что всегда выводит полный список книг со значениями атрибутов из связанных таблиц (если не включен фильтр), причём названия столбцов специально даны на русском (чтобы упростить структуру выпадающего списка фильтра). Единственный код, который потребуется в явном виде – это поле *status_id*, поскольку именно его значение изменяется при приёме/выдаче книг. Более никаких особенностей в этом запросе нет. Используйте созданную таблицу в качестве источника данных для *DataGridView3*.

В свойствах столбцов для элемента *DataGridView3* можно подправить только значения ширины столбцов, а столбец *status_id* сделать невидимым. Писать какой-либо программный код

для этого визуального элемента не нужно, включать правку, удаление и добавление – тем более. Обязательно следует установить свойство *SelectionMode* в **FullRowSelect**, свойство *Multiselect* сделать **False**, а свойство *ReadOnly* сделать **True**.

Сразу же сделаем систему фильтрации, которая очень напоминает поиск, ранее самостоятельно реализованный в Access-форме (см. рис. 2.36). Она нужна для того, чтобы как можно быстрее найти нужную книгу (по коду, названию или другому признаку) из списка всех книг в библиотеке. Однако здесь мы обойдёмся без стимулирующей кнопки, фильтр будет отрабатывать всякий раз, как только мы изменим значение в поле *var_find.* Для этого запрограммируем событие *Key_Up* для элемента *var_find*; назначим ему следующий код:

```
Private Sub var_find_KeyUp(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles var_find.KeyUp
    If Len(var_find.Text) > 0 Then
        DataTable3BindingSource.Filter = var_field.Text & " LIKE '" & var_find.Text &
"%'"
    Else
        DataTable3BindingSource.Filter = ""
    End If
End Sub
```

Естественно, при этом следует не забыть назначить значения и для поля с выпадающим списком *var_field.* Можно ограничиться простым списком, созданным вручную (рис. 3.23), в котором перечислены названия необходимых полей в таблице. После заполнения списка фильтр должен сразу начать работать. Обратите внимание, что свойство *Filter* принадлежит не *DataGridView3* и даже не *DataTable3*, а экземпляру класса, привязывающему источник данных к элементу управления *DataTable3BindingSource*.

А вот для заполнения выпадающего списка *var_libr* можно использовать одну из уже созданных в *LibraryDataSet* таблиц-адаптеров данных, так, чтобы первый (невидимый) столбец элемента *var_libr* содержал код библиотекаря, а второй – его ФИО и статус. Метке *Label7* информация присваивается, как только происходит активация вкладки «Регистрация», т.е. обрабатывается свойство Selected объекта *TabControl1* так, как показано ниже:

```
Private Sub TabControl1_Selected(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.TabControlEventArgs) Handles TabControl1.Selected
    Label7.Text = "Сегодня " & Now.Date
End Sub
```

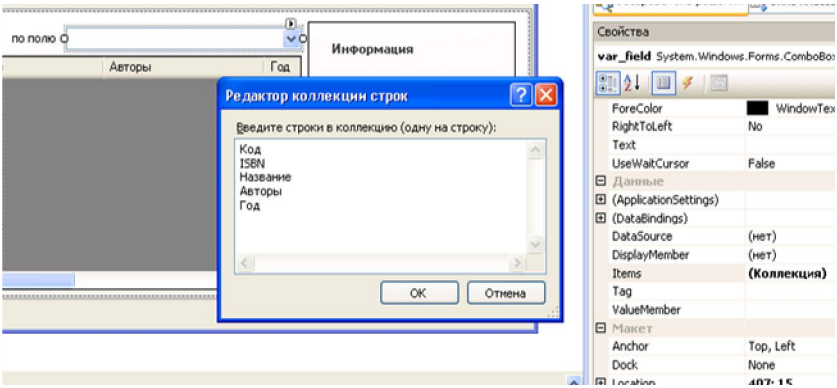


Рис. 3.23. Поле с выпадающим списком можно формировать либо вручную, либо использовать названия полей

И, наконец, финальная стадия работы над программой. Кнопки предполагается запрограммировать следующим образом:

1) При нажатии на кнопку «Выдать книгу» предварительно должна произойти проверка – выбрана ли книга из списка; и если *да*, то находится ли книга в библиотеке. Если это так, происходит смена статуса выбранной книги в таблице *books*, вносится запись о совершённой операции выдачи книги в таблицу *actions* с указанием кода читателя и работника библиотеки, а кнопка «Выдать книгу» перестаёт быть активной.

2) При нажатии на кнопку «Принять книгу» предварительно следует проверить – выбрана ли книга; и если *да*, то находится ли она на руках у читателя. Если так, статус книги изменяется в таблице *books*, вносится запись о возвращении книги в библиотеку в таблицу *actions*, кнопка «Принять книгу» переходит в пассивное состояние и не может быть нажа-

та повторно, пока указатель мышки не выйдет за пределы ограничительной рамки.

Чтобы запрограммировать эти действия, нужно создать по два запроса на изменение и ввод новых данных, однако предварительно следует решить, следует ли взаимодействовать с таблицами *LibraryDataSet* или можно обратиться непосредственно к таблицам исходной базы данных. В учебных целях рассмотрим второй вариант, который, возможно, покажется более интересным и для практического применения. Приведём текст программы, обеспечивающей функциональность кнопки *Button1*.

```
1   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button1.Click
2   'процедура выдача книги читателю
3   Dim c_i As Integer = -1
4   Dim d_i As Integer = -1
5   Dim u_i As Integer = -1
6   Dim Row As DataGridViewRow
7   'проверки истинности операций
8   For Each Row In DataGridView1.SelectedRows
9           u_i = Row.Cells(0).Value
10  Next
11  If u_i = -1 Then
12          MsgBox("Не выбран читатель!", MsgBoxStyle.Exclamation)
13  Exit Sub
14  End If
15  For Each Row In DataGridView3.SelectedRows
16          c_i = Row.Cells(0).Value
17          d_i = Row.Cells(1).Value
18  Next
19  If c_i = -1 Then
20          MsgBox("Не выбрана книга из списка!", MsgBoxStyle.Exclamation)
21          Exit Sub
22  End If
23  'проводка действия по регистрации выдачи книги
24  Select Case d_i
25          Case 1, 2
26          Conn.ConnectionString =
    System.Configuration.ConfigurationManager.ConnectionStrings
    ("library.My.MySettings.libraryConnectionString").ToString
27              Conn.Open()
```

```
28              SQL = "INSERT INTO actions (book_id, user_id, [date], libr_id,
    status_id) VALUES (" & c_i.ToString & ", " & u_i.ToString & ", now(), " &
    var_libr.SelectedValue.ToString & ", 1)"
29              Conn.Execute(SQL)
30              SQL = "UPDATE books SET status_id=4 WHERE id=" & c_i.ToString
31              Conn.Execute(SQL)
32              Conn.Close()
33              Button1.Text = "Выполнено"
34              Button1.Enabled = False
35              DataTable2TableAdapter.Fill(Me.LibraryDataSet.DataTable2,
    DataGridView1.SelectedRows(0).Cells(0).Value)
36              Me.DataTable3TableAdapter.Fill(Me.LibraryDataSet.DataTable3)
37          Case 0, 4, 5
38              MsgBox("В наличии требуемой книги нет!",
    MsgBoxStyle.Exclamation)
39          Case Else
40              MsgBox("Книга не может быть сейчас выдана на руки!",
    MsgBoxStyle.Exclamation)
41      End Select
42  End Sub
```

Рассмотрим представленную процедуру. Строки с 1 по 22 достаточно очевидны и реализуют условия, о которых говорилось выше. Переменная *u_i* содержит *id* читателя, переменная *c_i* есть *id* книги, переменная *d_i* есть код статуса книги.

Строка 26 представляет собой ссылку на ту самую строку атрибутов соединения с базой данных, которую использовали все *DataAdapter*s в *LibraryDataSet*. В качестве строковой константы здесь вполне можно было бы использовать явное указание пути к файлу базы данных Access, однако всегда считалось плохим тоном оставлять в коде прямые указания на путь к файлам, фактически дублируя уже существующие константы. Однако для того, чтобы Visual Basic корректно понимал представленную строку и всё, что находится за ней, требуется провести ряд дополнительных манипуляций:

1) В строках 1 и 2 вашего программного модуля следует указать:

```
Imports System.Configuration
Imports ADODB
```

2) В меню, как показано на рис. 3.24, выбрать пункт *Проект/Добавить ссылку…*, подождать появления окошка с компонентами и выбрать из списка сначала «**adodb**», затем «**system.configuration**».

3) В тексте модуля после указанных выше строк должна быть строка *Public Class* <имя формы>, следом за которой следует описать следующие переменные:

```
Dim Conn As New ADODB.Connection
Dim RS As New ADODB.Recordset
Dim SQL As String
```

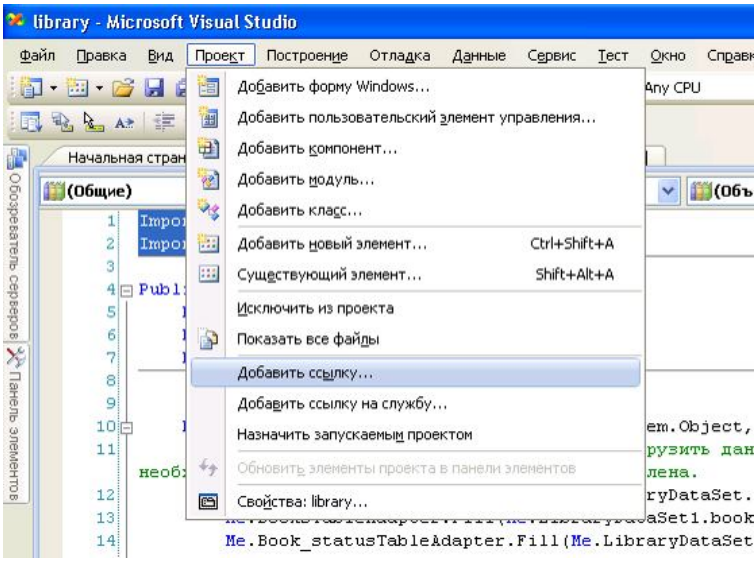


Рис. 3.24. Изменение структуры проекта

4) Проверить, чтобы в файле *app.config*, откуда будет подставляться путь к файлу базы данных (секция <*connectionStrings*>), вид пути содержал реальную иерархическую структуру подкаталогов. При необходимости (или при переносе базы данных в другое место) строку следует откорректировать.

После этого все запросы (29 и 31 строки) будут идти, используя непосредственно базу данных Microsoft Access. Однако после этого возникает проблема: как *LibraryDataSet* узнает, что в материнской базе произошли изменения? Для решения этой проблемы можно использовать перезагрузку *TableAdapter*s (строки 35 и 36), при этом *DataGridViews* перерисуются автоматически.

Теперь в качестве дополнительного элемента дизайна осталось внести код, необходимый для разблокирования кнопки, когда курсор мышки выйдет за границы рамки, ограничивающей кнопки. Несложно придумать и более удобные способы реактивации кнопок. Фрагмент текста программы, обслуживающего событие кнопки *Leave*:

```
Private Sub Panel1_Leave(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Panel1.Leave
    Button1.Text = "Выдать книгу"
    Button1.Enabled = True
    Button2.Text = "Принять книгу"
    Button2.Enabled = True
End Sub
```

Готово! Остаётся проверить работу кнопки.

Задание 5 для самостоятельной работы:

Запрограммируйте сходным образом вторую кнопку и проверьте работоспособность формы в целом, а также откорректируйте её графический интерфейс (читабельность полей, удобство расположения и размеры кнопок и других видимых элементов, подберите фон).

Глава 4. ВОЗВРАЩЕНИЕ К MICROSOFT ACCESS

4.1. Перспективы использования Microsoft Access

Несмотря на созданную нами удобную и полезную (если смотреть в перспективу) программу на Visual Basic, мы вновь вернёмся к программированию в СУБД Microsoft Access и расширим возможности нашего исходного проекта. Этот шаг будет сделан не зря, поскольку логика использования «однопользовательской» СУБД вовсе не означает, что лишь один пользователь на единственном компьютере может иметь к ней доступ. Более интересна следующая ситуация: к файлу базы данных, который расположен на сетевом ресурсе, могут быть одновременно подключены несколько пользователей – в нашем случае библиотекарь, выдающий литературу, сотрудник библиотеки, вводящий в базу новые книги, и директор библиотеки, пытающийся оценить статистику использования библиотечного фонда.

Хорошо известно, что такой способ организации данных имеет очень много недостатков и применяется главным образом на стадии тестирования разрабатываемых приложений. Основные недостатки – практически полное отсутствие защиты файла данных и трудно разрешаемые конфликты пользователей при одновременном доступе к одним и тем же таблицам. Однако имеются и определённые достоинства: например, базу данных можно легко скопировать на другой компьютер и спокойно просматривать её на индивидуальном рабочем месте (хотя нельзя однозначно сказать, можно ли считать этот факт достоинством).

Самым правильным решением в этой ситуации является переход на клиент-серверную сетевую СУБД с возможностью разработки различных внешних интерфейсов, включая Интернет-ориентированный интерфейс удалённого доступа. Не следует забывать, что СУБД MS Access, помимо выполнения своих основных функций однопользовательской СУБД, с успехом может быть использована также в роли клиента сетевой реляционной базы данных, в качестве которой традиционно понимается сетевая СУБД MS SQL Server. Таким образом, разработанная нами структура данных легко экспортируется в MS SQL Server, а созданные формы с небольшими переделками начинают работать

уже с «удалёнными» данными в корректном многопользовательском режиме.

Тем не менее, для случая «малой библиотеки» не противопоказано использовать вариант *файлового сервера*, одновременно подключив несколько СУБД MS Access и VB-приложений к *нескольким* файлам базы данных. Развитие представленной концепции может привести к достаточно интересной модели «распределённого файлового сервера», в которой каждый из сотрудников имеет собственный файл базы данных с необходимыми ему инструментами и таблицами под управлением собственной СУБД или специализированной программы, однако с возможностью полного (или ограниченного) доступа к базам данных других сотрудников.

Файлы в этом случае должны находиться в едином сетевом пространстве – например, в общих папках сети Microsoft Windows. При этом для формирования выборок и обработки данных необходимо будет использовать *распределённые* SQL-запросы, которые будут взаимодействовать одновременно с несколькими базами данных в общем сетевом пространстве и, при необходимости, менять информацию в таблицах во всех связанных между собой БД. Однако реализацию перечисленных вариантов развития архитектуры системы автор предлагает оставить читателям на будущее, ещё раз предлагая предварительно взвесить все «за» и «против» упомянутых путей совершенствования созданной системы.

4.2. Разработка набора отчётов для проекта

Продолжим работу над проектом в направлении создания отчётов, демонстрирующих статистику использования ресурсов библиотеки.

Отчёты можно создавать либо с помощью конструктора, либо (в простых случаях) с помощью мастера отчётов. В режиме конструктора в отчёте будем использовать следующие разделы: «Заголовок отчёта» для заголовка, «Верхний колонтитул» или «Заголовок группы» для наименования столбцов таблицы, «Область данных» собственно для вывода таблицы результатов, «Примечание для группы» для вывода промежуточных итогов, «Нижний

колонтитул» для номеров страниц и справочной информации, «Примечание отчёта» для общего итога. Если изначально какие-либо разделы не видны, их можно ввести, выбрав соответствующий пункт из выпадающего меню, щёлкнув правой кнопкой мышки по области отчёта.

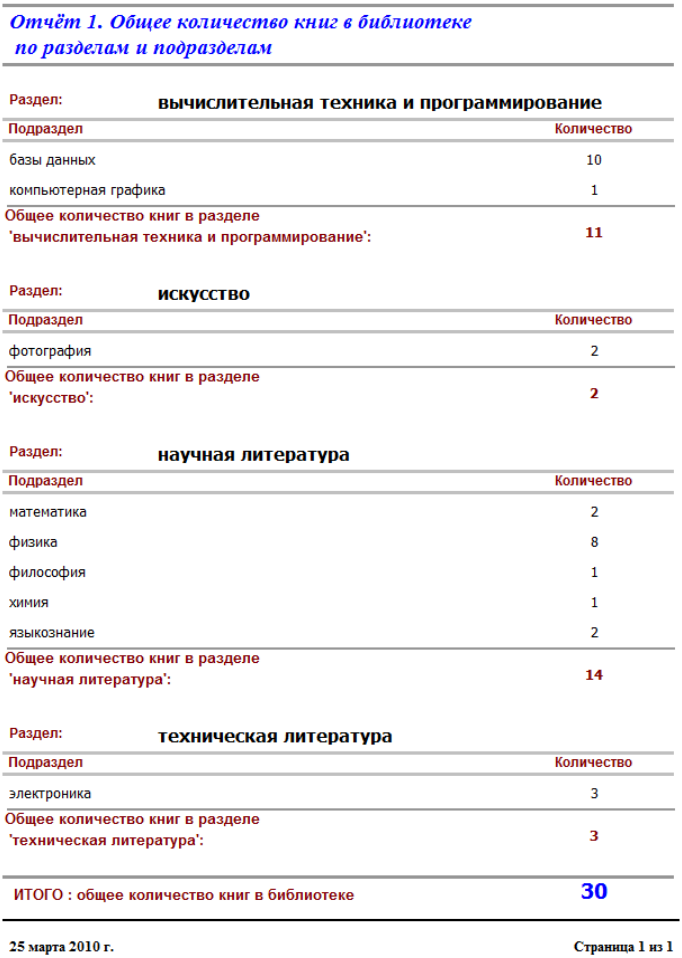
При создании отчётов мы будем отталкиваться от стандартных возможностей мастера, благо их вполне достаточно для разработки простых документов. Однако при работе с мастером, как это ни парадоксально, приходится уделять достаточно много времени «тюнингу» дизайна страницы, созданного этим мастером.

Создадим для примера отчёт «Количество книг по разделам и подразделам». Внешний вид отчёта *books_by_genres* представлен на рис. 4.1. В качестве источника данных для отчёта в первую очередь необходимо разработать и проверить запрос, поставляющий для него все необходимые данные.

Пример запроса с именем *books_by_genres*:

```
SELECT g.description AS genre, s.description AS subject, Count(b.id) AS amount
FROM (genres AS g INNER JOIN subjects AS s ON g.id = s.genre_id) INNER JOIN books
AS b ON s.id = b.subject_id
GROUP BY g.description, s.description;
```

Далее выберем в окне менеджера базы данных пункт «Отчёты», запустим мастер формирования отчёта и укажем в качестве источника данных только что созданный запрос. Выберем все поля, которые предоставляет запрос, и на следующем шаге работы мастера не забудем указать наиболее удобный уровень группировки информации на поле отчёта. В представленном на рис. 4.1 случае было выбрано одно поле *genre*. На следующем шаге можно указать необходимый способ сортировки внутри группы, а кнопкой «Итоги» инициировать расчёт общего количества книг в группах и общего итога (рис. 4.2). На последующих шагах работы мастера следует выбрать наиболее удобный, на ваш взгляд, макет для представления данных и стиль оформления.



Отчёт 1. Общее количество книг в библиотеке по разделам и подразделам

Раздел: **вычислительная техника и программирование**

Подраздел	Количество
базы данных	10
компьютерная графика	1
Общее количество книг в разделе 'вычислительная техника и программирование':	11

Раздел: **искусство**

Подраздел	Количество
фотография	2
Общее количество книг в разделе 'искусство':	2

Раздел: **научная литература**

Подраздел	Количество
математика	2
физика	8
философия	1
химия	1
языкознание	2
Общее количество книг в разделе 'научная литература':	14

Раздел: **техническая литература**

Подраздел	Количество
электроника	3
Общее количество книг в разделе 'техническая литература':	3

ИТОГО : общее количество книг в библиотеке	30

25 марта 2010 г. Страница 1 из 1

Рис. 4.1. Результат формирования отчёта в Microsoft Access

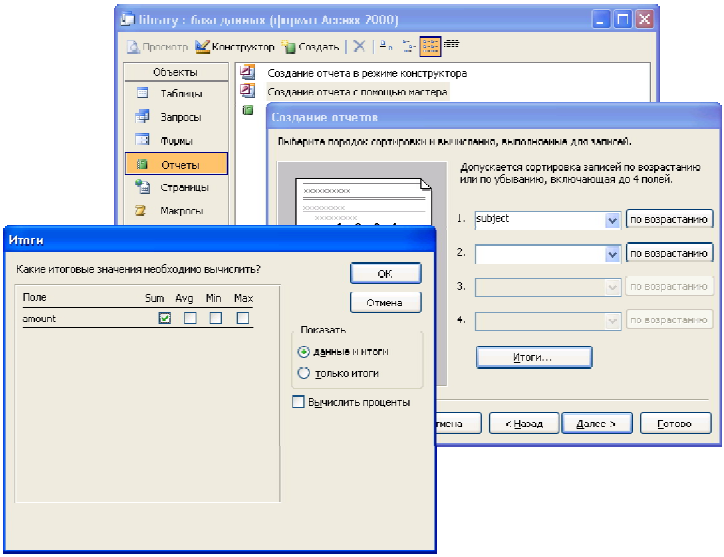


Рис. 4.2. Работа с мастером формирования отчётов

Теперь, когда отчёт, созданный мастером, готов, нетрудно заметить, что для придания ему презентабельного вида требуется поработать с макетом в режиме конструктора, изменив вид и наименование заголовков, а также способ отображения информации в примечаниях.

Заметим, что вычисляемые значения в полях, равно как и в случае создания форм, начинаются с символа «=». В выражениях могут использоваться имена полей запроса (в квадратных скобках), функции агрегирования, функции замены по условию (*iif*) и др. Поля отчёта могут ссылаться в том числе и на поля форм, существующих в вашем проекте. Для этого потребуется указать полный путь к соответствующему полю указанной формы, и, естественно, форма на момент генерации отчёта должна содержать адекватное значение в этом поле. Однако для создаваемого отчёта потребуется, напротив, несколько упростить выражения, подготовленные мастером.

Задание 6 для самостоятельной работы:

По аналогии с представленным документом сформируйте ещё несколько отчётов по приведенным темам:

1) Ротация книжного фонда (сколько новых книг принято в течение года, сколько книг утеряно, находится на руках, находится в архиве и т.д.).
2) Состояние книжного фонда (сколько книг находится в повреждённом состоянии, относятся к редким или ценным книгам).

Рассмотрим ещё один способ аналитической обработки информации из таблицы *actions*. Тема отчёта «График приёма-выдачи литературы библиотекой по месяцам». Заметим, что таблица *actions* как раз и создавалась для проведения анализа такого рода.

В данном случае речь идёт о так называемом перекрёстном запросе (или сводной таблице), который формирует трёхмерный срез по совокупности анализируемых данных. В нашем случае необходимо рассчитать количество актов приёма, выдачи и приобретения новой литературы по месяцам. В виде графика это будет выглядеть, например, так, как показано на рис. 4.3.

Поскольку необходимые нам компоненты MS Access, такие как *Pivot Chart (сводная таблица)* и *Pivot Table (сводная диаграмма)*, обладают весьма серьёзной встроенной функциональностью, от нас при формировании запроса не потребуется составлять конструкций вида TRANSFORM… PIVOT, достаточно будет написать запрос следующего простого содержания:

```
SELECT actions.book_id, actions.Date, action_status.description
FROM action_status INNER JOIN actions ON action_status.id = actions.status_id
```

Сам по себе такой запрос не решит поставленной задачи, однако он послужит источником данных для новой формы *action_analysis*. Создайте эту форму в режиме конструктора, подключите к ней только что созданный запрос и выведите на форму три текстовых поля, связанных с тройкой значений, выдаваемых запросом (рис. 4.4).

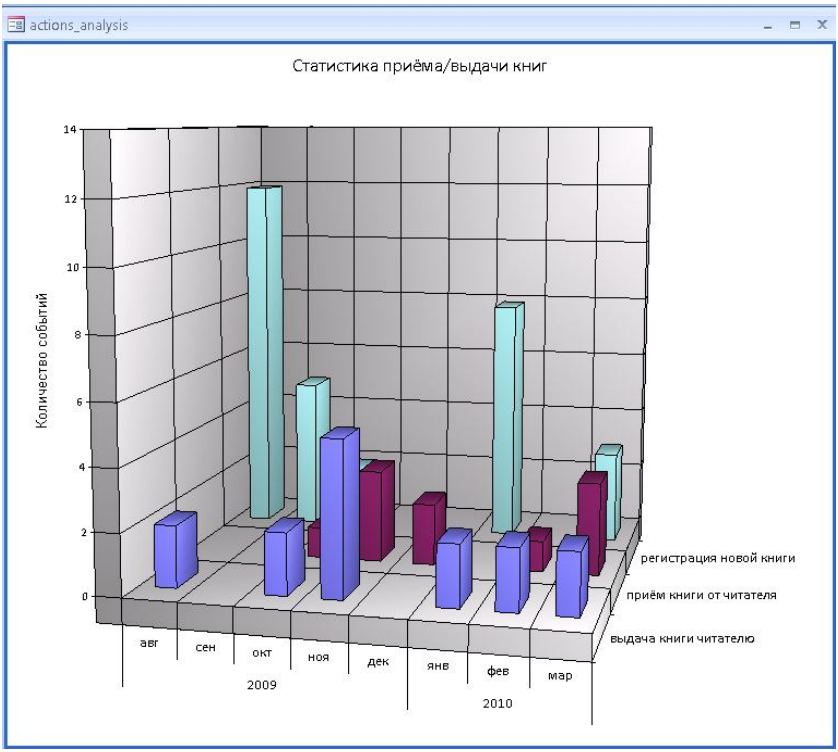


Рис. 4.3. Вид формы для представления графиков

Переключите созданную форму в представление «Сводная диаграмма» и сделайте его в свойствах формы представлением по умолчанию. В появившемся шаблоне (рис. 4.5) следует последовательно перетащить из «списка полей» на нижнюю активную позицию листа диаграммы элементы «годы» и «месяцы», на среднюю правую – поле *description*, на ближнюю к полю графика верхнюю – поле *book_id*.

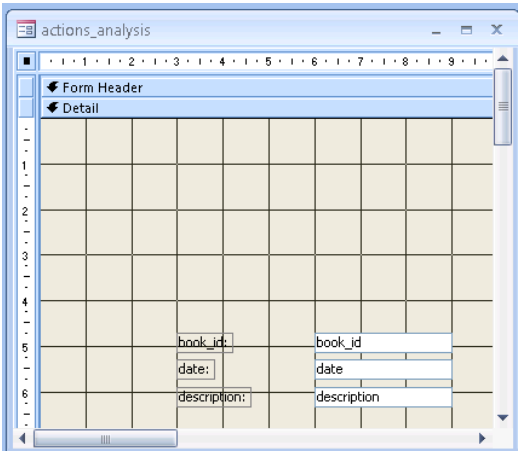


Рис. 4.4. Начало разработки формы для представления графиков

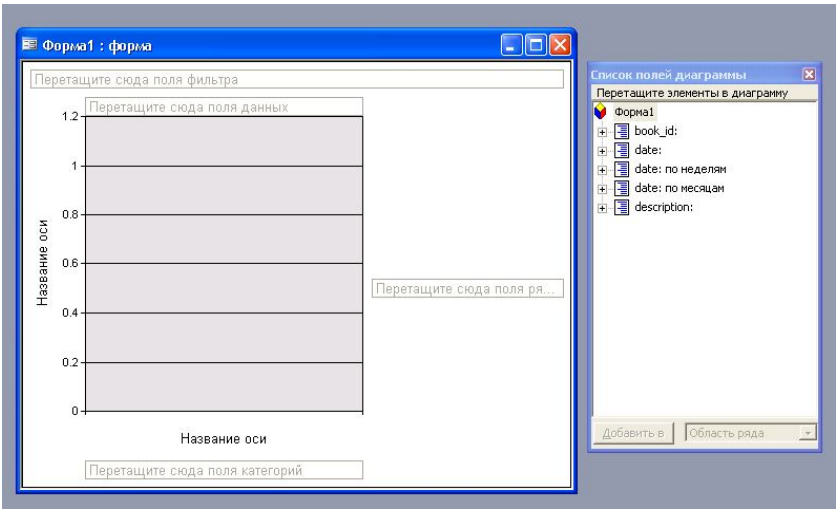
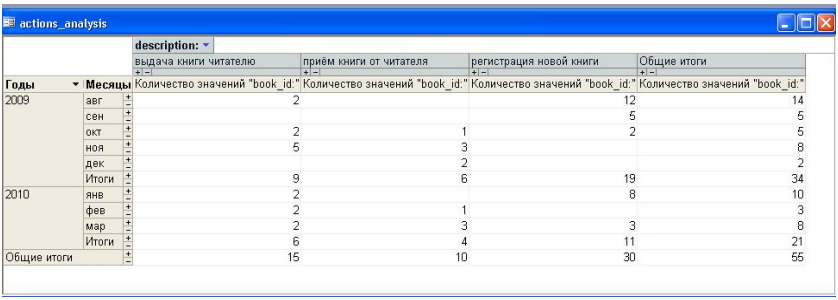


Рис. 4.5. Создание «перекрёстного» графика

Значение *book_id* автоматически примет функцию агрегирования *Sum()*, что неверно. Правым щелчком мышки на данном поле вызовите пункт *Автовычисления* и замените функцию на *Число* (*Count()*).

График в черновом варианте готов. Для более качественного вывода используйте панель «Свойства» и панель «Тип диаграммы», которые можно вызвать из контекстного меню графика. Переключение формы в представление «Сводная таблица» позволяет, при условии правильно выполненных предыдущих шагов, сразу же получить результат, показанный на рис. 4.6.


actions_analysis

		description:			
		выдача книги читателю	приём книги от читателя	регистрация новой книги	Общие итоги
Годы	Месяцы	Количество значений "book_id:"	Количество значений "book_id:"	Количество значений "book_id:"	Количество значений "book_id:"
2009	авг	2		12	14
	сен			5	5
	окт	2	1	2	5
	ноя	5	3		8
	дек		2		2
	Итоги	9	6	19	34
2010	янв	2		8	10
	фев	2	1		3
	мар	2	3	3	8
	Итоги	6	4	11	21
Общие итоги		15	10	30	55

Рис. 4.6. Вид сводной таблицы

Как же быть, если в этом разделе речь шла об отчётах, а представленный пример был направлен на изготовление формы? Ситуация разрешается просто: создаётся пустой отчёт в режиме конструктора и выбирается инструмент «подчинённая форма/отчёт». В качестве подчинённой формы отчёта указывается только что разработанная *action_analysis*, и отчёт снабжается необходимыми заголовками. После этого документ готов для вывода на печать.

Задание 7 для самостоятельной работы:

Поскольку все цели проекта выполнены, улучшение работы программ будет связано с конкретизацией практического использования созданного программного продукта. В частности, можно реализовать форму для регистрации читателей библиотеки, которую удобнее добавить в проект VB.NET.

СПИСОК ЛИТЕРАТУРЫ

1. *Конноли Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Конноли, К. Бегг. – М.: Вильямс, 2003. – 1440 с.
2. *Дейт К.* Введение в системы баз данных / К. Д. Дейт. – М.: Вильямс, 2001. – 1072 с.
3. *Хомоненко А.* Базы данных: учебник для высших учебных заведений / под ред. проф. А. Д. Хомоненко. – СПб.: Корона, 2004. – 736 с.
4. *Кузнецов С.* Основы баз данных: учебное пособие / С. Д. Кузнецов. – М.: Бином, 2007. – 484 с.
5. *Райордан Р.* Основы реляционных баз данных / Р. Райордан. – М.: Русская Редакция, 2001. – 384 с.
6. *Шкарина Л.* Язык SQL: учебный курс / Л. Шкарина. – СПб.: Питер, 2001. – 592 с.
7. *Грофф Дж.* SQL: Полное руководство / Дж. Грофф, П. Вайнберг. – Киев: BHV, 2001. – 816 с.
8. *Бекаревич Ю.* MS Access 2000 за 30 занятий / Ю. Б. Бекаревич, Н. В. Пушкина. – СПб.: БХВ-Петербург, 2000. – 512 с.
9. *Вилльариал Б.* Программирование Access 2002 в примерах / Б. Вилльариал. – М.: КУДИЦ-образ, 2003. – 496 с.
10. *Постолит А.* Visual Studio .NET: разработка приложений баз данных / А. В. Постолит. – СПб.: БХВ-Петербург, 2003. – 544 с.
11. *Патрик Т.* Visual Basic 2005. Рецепты программирования / Т. Патрик, К. Крейг. – СПб.: БХВ-Петербург, 2008. –752 с.
12. *Сеппа Д.* Microsoft ADO .NET / Д. Сеппа. – М.: Русская Редакция, 2003. – 640 с.