

Куандыков Абу Абдикадырович

**СОЗДАНИЕ СИСТЕМЫ БАЗЫ ДАННЫХ
В СРЕДЕ DELPHI**

Алматы 2007

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ
КАЗАХСТАН**

**Казахский национальный технический университет
им. К.И. Сатпаева**

А.А. Куандыков

**СОЗДАНИЕ СИСТЕМЫ БАЗЫ ДАННЫХ
В СРЕДЕ DELPHI**

Рекомендовано Министерством образования и науки
Республики Казахстан в качестве учебного пособия

Алматы 2007

УДК 519.6 (075)
ББК 32.973–018.2 я 73
К 88

Куандыков А.А. Создание системы базы данных в среде Delphi: Учеб. пособие. – Алматы: КазНТУ, 2007. – 102 с., ИЛЛ

ISBN 9965–843–74–0

В настоящем пособии рассматриваются вопросы проектирования и разработки системы баз и банков данных в среде Delphi. Учебные материалы распределены по пяти разделам и приложениям. В теоретической части вкратце излагаются технология проектирования и вопросы планирования, процесс создания системы. Процесс создания программного обеспечения системы продемонстрирован в режиме быстрой разработки приложений на одном практическом примере.

Учебное пособие предназначено для студентов технических вузов, изучающих вопросы и методы проектирования и создания систем баз данных, аспирантам и программистам, специализирующихся в области разработки баз и банки данных.

Ил. 59. Библиогр.–1 назв.

БК 32.973–018.2 я 73

Рецензенты: *И.Т. Утепбергенов*, д-р техн. наук, проф., зав. каф. АТК им. М. Тынышпаева.
Е.Ж. Айтхожаева, канд. техн. наук, проф. каф. ВТ КазНТУ им. К.И. Сатпаева.
Б.Д. Хисаров, канд. техн. наук, проф., зав. каф. «Инженерная кибернетика» АИЭИС.

Печатается по плану издания Министерства науки и высшего образования Республики Казахстан на 2007 г.

ISBN 9965–843–74–0

© КазНТУ, 2007

ВВЕДЕНИЕ

Вычислительная техника (или машина) в самом начале своего развития использовалась как инструмент вычисления, т. е. как инструмент решения математических задач итерационными численными методами. Для этой цели писали программы, которые реализуют методы вычислений, позволяющие вычислить или рассчитать значения искомого параметра на основе исходных данных. На этом этапе данные особого значения не имели для построения вычислительной работы и ее программного обеспечения, и исполняют роль исходных значений параметров, вводимых в машину вместе с программой. В ходе выполнения программы, подставив значения исходных данных в формулу, получали результат, который также является значением расчетных параметров.

Таким образом, можно сказать, что вычислительная техника со своим программным обеспечением была использована для математических расчетов: инженерных, научных, научно-экспериментальных (например, исследование определенных явлений или процессов путем проведения имитационного эксперимента методом Монте-Карло).

Однако по мере использования вычислительной техники (теперь уже компьютеры) в процессе хозяйственной деятельности предприятия, в решении практических задач, причем при автоматизации процессов как отдельных производственных задач, так и автоматизации деятельности предприятия и организации в целом, доля данных, как по объему, так и по значимости для предприятия и пользователей результатов решения задач, повысилась.

В результате слияния, как вычислительных процессов, так и процессов обработки данных и информации, вычислительную технику со своим программным обеспечением, предназначенную для решения задач данного класса, стали полнофункциональными системами поддержки бизнес-процессов предприятия.

Кроме того, увеличение объемов, обрабатываемых и хранимых данных, требовало организации всех данных, необходимых для поддержки всех бизнес-процессов предприятия, определенным образом, и создание специальной программы, для работы с ней. В условиях корпоративной деятельности, в целях поддержки бизнес-процессов пред-

приятия, требовалась организация доступа к организованным данным множества пользователей. Решение этой задачи было достигнуто на основе создания сетевой связи и технологий между компьютерами пользователей и сервером данных.

Таким образом, была заложена основа для развития распределенной системы для различных сфер деятельности человека. При этом компьютерные распределенные системы назывались либо распределенными вычислительными системами, когда подчеркивались вычислительные аспекты, решаемых или выполняемых задач, либо распределенными информационными системами, когда подчеркивались информационные составляющие процесса решения задач.

При решении задач, как вычислительные, так и информационные процессы имеют одинаково важное значение. Поэтому выбор названия системы «вычислительная» или «информационная» является равнозначным, поэтому иногда данные классы систем называют «информационно-вычислительными системами».

Итак, возрастание значимости данных, изменил роль программы введенных в компьютер (или компьютерных систем). Программы теперь стали обслуживать данные, хранящиеся в априори и введенные в текущий момент с клавиатуры, датчиков или с других устройств в компьютер. Поэтому от качества хранения и представления данных во многом зависит успешность решения задачи, порученной компьютеру.

Организованное и структурированное объединение этих данных стало называться базой данных (БД), а программы, предназначенные для их обработки, стали называть программным приложением.

База данных совместно с программным приложением составляет систему базы данных (СБД). В свою очередь, системы базы данных составляют основу всех видов информационных систем, т. е. систем, которые предназначены для обработки, хранения и представления данных для пользователей. Другими словами, система базы данных является платформой для систем, обеспечивающих решения задачи информационной логистики в организациях, предприятиях и территориально распределенных структурных образованиях.

Отсюда вытекает, что эффективность функционирования системы базы данных имеет важное значение для эффективной автоматизации задач практики. В свою очередь, функциональные качества системы базы данных зависят от эффективных инструментариев и технологии, применяемых для создания системы базы данных, и имеют важные значения для решения проблем автоматизации практических задач.

В настоящее время одним из эффективных и широко распространенных средств разработки системы базы данных или, точнее, приложения к базе данных стал инструментарий, т. е. пакет программы Delphi с различными версиями. Причем, Delphi используется в различных вариантах, в частности:

- система базы данных (СБД) создается полностью в среде Delphi, т. е. как БД, так и система управления БД (СУБД) в локальной системе (т. е. на одном компьютере) или в распределенной системе. При этом имеется в виду, что БД создается на одном из локальных СУБД: FoxPRO, Visual Foxpro, Paradox, Visual Basic, Access и т. д., но с помощью средств Delphi, а точнее с помощью мастера базы данных *Database Desktop* (DBD);

- СУБД создается только на Delphi, а БД, независимо от инструментария Delphi, может быть создана другими средствами, в частности, на FoxPRO, Visual Foxpro, Paradox, Visual Basic, Access и т. д.;

- на Delphi создается только клиентская часть приложения, а БД может находиться на сервере, которая создана в Oracle или в другой среде.

Поэтому Delphi, как инструментарий и для нынешних, и для будущих проектировщиков приложений под Windows 95/98/2000/XP, имеет большое значение. Delphi, по существу, послужит связующим звеном между существующими технологиями и новыми, которые будут разработаны в будущем.

Таким образом, очевидна важность знания Delphi для специалистов разработчиков приложений. Но теперь встает вопрос, каким образом эффективно учить желающих освоить этот инструмент.

Давно известно, что лучше один раз увидеть, чем сто раз услышать. А при изучении программных средств еще лучше один раз попробовать. Поэтому давайте попрактикуемся в создании простейшего приложения, работающего с базами данных, пройдя при этом все этапы – от создания псевдонима базы данных до запуска разработанного приложения.

Если Вы никогда не проектировали приложения для работы с базами данных в среде Delphi, приобретенный опыт поможет Вам понять, что построение таких приложений в Delphi максимально упрощено. Есть и еще одно преимущество выполнения описываемых упражнений: читая в дальнейшем о средствах разработки приложений баз данных в Delphi, Вы будете встречать уже знакомые названия и понятия.

Автор легко продемонстрировал процесс создания простых информационных систем и вместе с тем старался ввести Вас в технологию создания больших информационных систем.

Данная цель достигается путем раскрытия и соблюдения основных этапов проектирования крупных систем.

Следует отметить, что данное учебное пособие является усовершенствованным переизданием ранее изданного учебного пособия [1].

ГЛАВА I. МЕТОДОЛОГИЯ И ОБЩИЕ ВОПРОСЫ СОЗДАНИЯ СИСТЕМЫ БАЗЫ ДАННЫХ

1.1. Общая технология создания информационных систем

Информационные системы как любое изделие имеет жизненный цикл (ЖЦ), состоящий из этапов проектирования, создания (изготовление), внедрения и эксплуатации. В отличие от других изделий, ИС, в зависимости от размера и уровня долевого участия в бизнес - процессе организации (предприятия), на каждом этапе ведет себя по-разному. В связи с тем, что внешняя среда функционирования ИС является динамической, для эффективного выполнения своих функций более масштабные ИС необходимо сопровождать и администрировать.

Отметим, что в настоящее время в связи с интенсивным развитием информационных технологий этапы проектирования и создания четкой границы не имеют. Применительно к процессам этих этапов жизненного цикла ИС используются такие термины как построение, создание, разработка, проектирование, конструирование, программирование, кодирование, которые в общем случае имеют различные смыслы.

В общем случае процесс проектирования и создания различны:

1) в конце этапа проектирования само изделие (т. е. ИС) еще отсутствует, мы получим только его проект на бумаге (здесь следует отметить, что если при проектировании используется CASE технология, то проект мы получим в электронном виде);

2) в конце этапа создания изделие (т. е. ИС) будет получено в физическом виде, поэтому этот этап соответствует этапу изготовления изделия.

Однако на практике при создании ИС отсутствует четкое разделение на этапы проектирования и создания. Процессы этих этапов смешаны (это отчасти связано с инструментарием и средой, в которой создается ИС).

Отметим следующие особенности создания ИС в настоящее время.

В целом создание ИС можно вести кустарным образом без использования опыта и известных приемов, методов, инструментов или средств, а можно использовать полученный ранее опыт и другие знания. Второй вариант соответствует технологии создания ИС. Использование определенной технологии обеспечивает эффективность процессов создания и достижения качественных результатов.

Поэтому большое значение имеет использование общепринятых или формирование собственных технологий создания ИС, которые затем выступают в качестве стандартов.

Итак, при создании ИС любого типа возникает необходимость определения или выбора:

- технологии создания ИС, с помощью которой или на базе которой должна создаваться ИС;
- способа организации процессов или технологии ведения бизнес - процессов создания ИС, с помощью которой должны вестись проектные и внедренческие работы.

Результаты исследования и накопленный опыт, полученные в области проектирования и разработки больших ИС, позволяют построить общую технологию создания ИС для различных целей с различными характеристиками.

Механизм возникновения и/или формирования технологий следующий.

Технология состоит из ряда составляющих, в частности, из общей и отдельных схем выполнения этапов проекта, а также методики, инструментария и стандартов. Соотношения между этими составляющими технологии следующие:

1. Общая схема и схема отдельных этапов отражает лишь логику последовательности выполнения стадий, этапов и операций, необходимых для проектирования и создания ИС.

2. Если на каждом этапе схемы показать инструменты выполнения этапов и операции (которые служат как, например, станки и инструменты в машиностроительной промышленности), то схема переходит в технологию создания ИС. Инструментальный состав технологии создания ИС включает инструменты для проведения анализа предметной области и задачи моделирования, проектирования, испытания и исследования качества спроектированной системы.

3. Стандарты нужны для того, чтобы унифицировать и повысить качество результатов выполнения этапов и всего процесса создания ИС. Они еще нужны для обеспечения совместимости, расширяемости и интероперабельности согласно стандартам взаимодействия открытых систем.

Таким образом, исходя из вышеизложенного, вытекает следующая общая схема технологии создания системы ИС:

1. Формулировка и анализ задачи автоматизации.

2. Проектирование СУБД.
 - 2.1. Проектирование на концептуальном уровне.
 - 2.2. Проектирование на физическом уровне.
3. Создание, т. е. производство системы.
4. Испытание и внедрение системы.
5. Сопровождение системы. Корректировка и развитие ИС.
6. Администрирование системы.

Следующим важным вопросом создания ИС является организация процессов создания. Эффективность проектирования больших ИС будет зависеть от того, как удачно организован процесс проектирования. Организация (технология организации или бизнес-процесса) и менеджмент на различных этапах жизненного цикла ИС, точнее процессы проектирования и процессы эксплуатации друг от друга отличаются.

Для проектирования больших систем организационная структура должна быть объемной. На одной (нижней) плоскости организационной структуры все специалисты, ведущие проектные работы (т. е. программисты) должны иметь прямые контакты между собой. Менеджер проекта должен находиться на верхнем уровне.

ИС после внедрения не должна находиться в статическом положении, а, наоборот, она должна развиваться по мере изменения условий функционирования ИС и по мере возникновения новых заданий и требований к системе. Здесь, все эти вопросы не будут рассматриваться, т. к. изучаются вопросы создания простой ИС или ее программного обеспечения, при проектировании которого требования к технологии и организации особую актуальность не имеют. Отметим только их важность для эффективного создания больших ИС.

1.2. Общая схема проектирования системы базы данных

Система базы данных является классом ИС, которая имеет БД и СУБД.

База данных является составной частью информационной системы (ИС), которая выполняет не только хранение данных, но и их обработку. Поэтому, проектированию данных всегда сопутствует (а чаще предшествует) проектирование алгоритмов их использования. Поэтому рассмотрим вопросы проектирования информационной системы, которые, в первую очередь, сводятся к процессам проектирования базы данных, состоящих от функционального моделирования предметной

области до построения структуры реляционной базы данных.

В связи с этим раскроем основную суть процесса создания базы данных.

Суть процесса создания базы данных.

При создании эффективной ИС важным вопросом является изучение предметной области. Поэтому остановимся на особенностях предметной области автоматизации.

Предметная область – часть реального мира, подлежащая изучению с целью организации управления и, в конечном счете, автоматизации. Предметная область представляется множеством фрагментов, например, предприятие – цехами, дирекцией, бухгалтерией и т. д. Каждый фрагмент предметной области характеризуется множеством объектов и процессов (использующих объекты), а также множеством пользователей, характеризующихся различными взглядами на предметную область.

В теории проектирования информационных систем предметную область (или, если угодно, весь реальный мир в целом) принято рассматривать в виде трех представлений:

1) представление предметной области в том виде, как она реально существует;

- как ее воспринимает человек (имеется в виду проектировщик базы данных);

2) как она может быть описана с помощью символов.

Таким образом, при проектировании ИС проектировщики и разработчики после обследования и анализа предметной области работают с ее моделью, т. е. с описанием (представлением) реальности и с данными о предметной области, которые отражают это представление.

Данные, используемые для описания предметной области, представляются в виде трехуровневой схемы, т. е. в виде модели ANSI/SPARC как на рис. 1.1:



Рис.1.1. Трехуровневая схема модели ANSI/SPARC предметной области

Внешнее представление (внешняя схема) данных является совокупностью требований к данным со стороны некоторой конкретной функции, выполняемой пользователем. Концептуальная схема является полной совокупностью всех требований к данным, полученным из пользовательских представлений о реальном мире. Внутренняя схема – это сама база данных.

Отсюда вытекают основные этапы, на которые разбивается процесс проектирования базы данных информационной системы:

1. **Концептуальное проектирование** – сбор, анализ и редактирование требований к данным. Для этого осуществляются следующие мероприятия:

- обследование предметной области, изучение ее информационной структуры;
- выявление всех фрагментов, каждый из которых характеризуется пользовательским представлением, информационными объектами и связями между ними, процессами над информационными объектами;
- моделирование и интеграция всех представлений.

По окончании данного этапа получаем концептуальную модель, инвариантную к структуре базы данных. Часто она представляется в виде модели "сущность-связь".

2. **Логическое проектирование** – преобразование требований к данным в структуры данных. На выходе получаем СУБД – ориентированную структуру базы данных и спецификации прикладных программ. На этом этапе часто моделируют базы данных применительно к различным СУБД и проводят сравнительный анализ моделей.

3. **Физическое проектирование** – определение особенностей хранения данных, методов доступа и т. д.

Различие уровней представления данных на каждом этапе проектирования представлено табл. 1.1.

Таблица 1.1.

Уровни представления данных

Концептуальный уровень • сущности • атрибуты • связи	Представление аналитика
Логический уровень • записи • элементы данных • связи между записями	Представление программиста

Физический уровень ● группирование данных ● индексы ● методы доступа	Представление администратора
---	------------------------------

На основе приведенного представления сути процесса создания базы данных позволяет определить общую технологию создания системы базы данных.

Общую схему технологии создания ИС применительно к системам базы данных можно представить таким образом:

1. Формулировка и анализ задачи автоматизации.
2. Проектирование СБД.
 - 2.1. Проектирование базы данных системы.
 - 2.1.1. Концептуальное или даталогическое проектирование.
 - 2.1.2. Физическое проектирование. Определение типа архитектуры и места хранения БД: локальная, сетевая клиент-серверная, распределенная.
 - 2.2. Проектирование СУБД.
 - 2.2.1. Концептуальное проектирование.
 - 2.2.2. Физическое проектирование. Определение архитектуры СУБД. Определение режима программирования.
3. Создание, т. е. производство системы.
4. Испытание и внедрение системы.
5. Сопровождение системы. Корректировка и развитие БД и СУБД.
6. Администрирование системы.

Процесс проектирования БД на логическом уровне сводится к следующему.

Данные, необходимые для решения поставленной задачи с помощью СБД состоит из множества группы данных. Каждая группа из множества представляется в виде таблицы. Поэтому БД СБД состоит из ряда таблиц.

Отсюда процесс проектирования БД на логическом уровне сводится, путем анализа проблемной среды, к выявлению групп данных, которые можно представить в виде таблиц. Определяются структуры выделенных таблиц.

Тем самым спроектируем отдельные таблицы, которые являются основными компонентами БД.

Далее из этих таблиц следует сформулировать единую модель данных путем установления связей между ними.

1.3. Общая схема современных архитектур системы базы данных

В настоящее время в области информационных технологий применяются термины: платформа, архитектура, инфраструктура, технология, методология, инструментарии и стандарт. Не вдаваясь в смысл этих понятий, приведем обзор архитектуры информационных систем, ныне существующих. В общем случае архитектура – это структура технического обеспечения системы и плюс еще общая технология (или общая идея) обработки данных в системе на базе заданной структуры технического обеспечения.

В настоящее время система базы данных может строиться на основе одной из семи архитектур. Перечислим их.

1. Система базы данных с локальной архитектуры. Данный вид архитектуры системы базы данных может состоять из одного компьютера или из мэйнфрейма.

Мэйнфрейм состоит (имеет архитектуру) из одного центрального мощного компьютера-сервера и множества, связанных с ним компьютеров-терминалов.

2. Система базы данных файл-серверной архитектуры, состоящая из двух уровней.

Верхний уровень – сервер данных, нижний – клиентские системы. Количество клиентских систем соответствует числу пользователей.

Технология работы файл-серверной системы осуществляется полной передачей БД из сервера в систему работающего клиента (пользователя).

3. Система базы данных с двухуровневой клиент-серверной архитектурой, которая состоит из:

- верхнего уровня, составляющая сервер базы данных;
- нижнего уровня, состоящего из совокупности клиентских систем или станций.

Количество клиентских систем также соответствует числу пользователей.

Структура данной архитектуры совпадает со структурой предыдущей архитектуры. В данной архитектуре технология работ клиент-серверной системы сводится к выборочной передаче данных из сервера БД только тех, которые являются ответом на запрос клиента (пользователя), сформулированного в клиентской системе. В данном случае двухуровневая клиент-серверная система становится OLTP-системой (система оперативной обработки транзакций), т. е. система работает на базе OLTP-технологии.

4. Система базы данных с трехуровневой клиент-серверной архитектурой. Данная архитектура системы базы данных имеет:

- верхний уровень, который составляет сервер базы данных;
- средний уровень, на котором располагается сервер процедур обработки данных (сервер приложения);
- нижний уровень составляет совокупность клиентских систем или станции. Количество клиентских систем столько же, сколько и число пользователей.

5. Система базы данных с распределенной архитектурой. Данная архитектура состоит из двух уровней.

Верхний уровень архитектуры – одноранговая сеть, состоящая из серверов. Уровень серверов может состоять из серверов данных и процедур (приложений).

Нижний уровень – клиентские системы и станции пользователей.

6. Система базы данных со смешанными с традиционной и Internet/Web архитектурами. Система базы данных с одной из вышеперечисленных архитектур, входящая в сеть Internet. Внутренняя технология обработки данных совместима с Internet-технологией.

7. Система базы данных с полной Internet архитектурой. Система базы данных построена полностью на базе архитектуры и технологии сети Internet.

В данном учебнике описывается технология создания только системы локальной базы данных на базе одного компьютера.

Теперь несколько слов об особенностях создания систем в среде операционной системы Windows. В настоящее время интенсивно развивается технология создания программных и информационных систем в Windows, которая имеет свои особенности. Эта особенность в первую очередь влияет на статус прикладных программных систем.

В среде операционной системы Windows главную управляющую функцию выполняет сама операционная система, а разработанная программа выполняет подчиненную функцию. Поэтому в случае работы систем под ОС Windows прикладную программную систему называют приложением. Однако мы все же будем использовать термин программная система или система.

Во-вторых, технология создания программных систем под Windows является объектно-ориентированной, поэтому искомая прикладная программная система komponуется из компонентов, которые определяют технологию разработки программных систем.

На других особенностях не будем останавливаться, т. к. это является отдельной темой.

Еще следует дать терминологические определения и уточнения.

Важным является понятие системы базы данных. Система, которая работает с базой данных, состоит из двух компонентов:

- базы данных;
- системы управления базой данных (СУБД).

По отдельности и база данных, и СУБД самостоятельно не имеют функциональную законченность. Функциональную законченность имеет только совместное их использование. Практическая система должна иметь обе эти компоненты, поэтому полную систему назовем системой базы данных. Тем самым подчеркиваем, что данная система работает с БД.

ГЛАВА II. СОВРЕМЕННЫЕ ИНСТРУМЕНТАРИИ СОЗДАНИЯ ПРОГРАММНЫХ СРЕДСТВ И DELPHI

Среди существующих инструментов создания приложений Delphi является самым эффективным. Поэтому для создания нашей системы в качестве инструмента берем Delphi. В наше время уже имеется 7-ая версия этого популярного продукта, но, начиная с 3-ей, технология создания примерно идентичны. В данном учебнике создание системы ведется в среде Delphi-3 и выше.

Для начала работы предполагается, что на вашем компьютере Delphi уже установлен. Поэтому покажем только схему запуска Delphi.

Основная схема запуска Delphi.

Пусть на вашем компьютере (т. е. компьютер включен, рабочий стол Windows запущен) загружен рабочий стол одной из версии Windows, т. е. Windows 95/98/2000/XP. Будем считать, что читатель умеет запускать компьютер и знаком с операционной системой Windows. Тогда запуск Delphi с рабочего стола Windows происходит по схеме:

Пуск/Программы/Borland Delphi 3.

После подачи этой команды через некоторое время полностью запускается Delphi и на экране компьютера появляется интерфейс Delphi или точнее интерфейс интегрированной среды разработки (IDE - Integrated Development Environment). Структура IDE Delphi приведена на рис.1.

Отметим, что Delphi можно запустить с ярлыка, если на рабочем столе Windows создан его ярлык.

Вкратце опишем элементы IDE Delphi.

2.1. Назначение IDE Delphi

Интегрированная среда разработки Delphi предназначена для простого, логичного представления на экране объектов, используемых при разработке приложения, в нашем примере системы базы данных. Через ряд окон интерфейса можно, в частности, выполнять следующие процедуры разработки (в режиме RAD): добавлять, изменять и удалять объекты в приложениях и компонентах.

Поэтому уместно напомнить, что при работе с Delphi можно по-

лучить достаточно полную информацию об активном в данный момент средстве программирования, нажав клавишу F1.

2.2. Структура и состав IDE Delphi

После запуска Delphi на экране открываются четыре окна интегрированной среды разработки приложений: главное окно, окна проектировщика форм, редактора кода и инспектора объектов. На рис. 2.1 показан вид экрана после запуска.

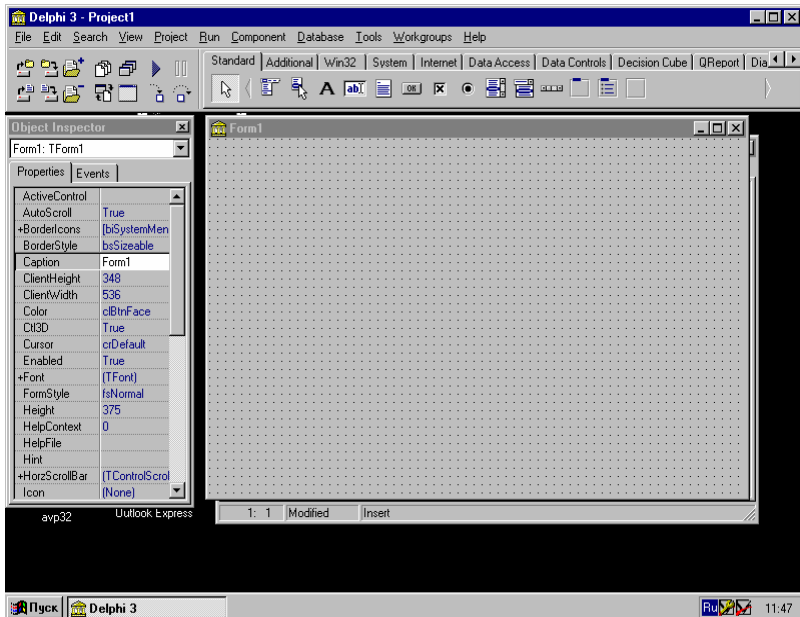


Рис.2.1. Интегрированная среда разработки Delphi

2.3. Главное окно

В верхней части экрана расположено главное окно, в котором находятся строка главного меню, панели инструментов и палитра компонентов. Главное окно программы остается открытым все время, пока остается загруженной Delphi. Закрывая главное окно, вы тем самым выгружаете из оперативной памяти Delphi.

В строке заголовка главного окна отображается имя открытого в данный момент проекта: Project1. Проект может быть открыт в IDE как в режиме проектирования, так и в режиме выполнения. По заголовку главного окна можно определить, в каком режиме открыт проект. В левой части заголовка главного окна, рядом с символом Delphi во время выполнения проекта следует пометка [Running]. Таким образом, всегда можно узнать, в каком режиме находится проект.

Строка меню содержит команды, необходимые для разработки и тестирования приложений, а также управления ими. Команды меню подробно рассмотрены в разделе "Меню".

2.4. Панель инструментов

Панель инструментов содержит кнопки, которые соответствуют определенным командам меню, например, командам меню File, View и др. Щелчок на какой-либо из кнопок приводит к тому же результату, что и выбор соответствующей команды в главном меню. Например, для открытия файла необходимо выбрать команду Open меню File или выполнить щелчок на кнопке Open панели инструментов Standard. Кнопки панели инструментов снабжены всплывающими подсказками (hints или tooltips). Если подвести курсор мыши к какой-либо кнопке, на экране будет отображено имя команды, вызываемой данной кнопкой.

Вы можете изменить конфигурацию любой панели инструментов, можете отображать или скрывать на экране панели инструментов, а также добавлять, удалять кнопки панелей инструментов и изменять их расположение.

2.5. Палитра компонентов

В палитре компонентов (Palette) отображаются компоненты, с помощью которых Вы создаете свои приложения. Хотя, возможно, в данный момент это вам мало о чем говорит, в дальнейшем вы будете очень часто использовать палитру компонентов при создании приложений. Компоненты являются основными элементами каждого Delphi-приложения и, одновременно, основой библиотеки визуальных компонентов Delphi — Visual Component Library (VCL). Они позволяют создавать пользовательский интерфейс ваших прикладных программ.

Пиктограммы стандартных компонентов Delphi, в соответствии с выполняемыми ими функциями, разделены на группы; каждая из этих

групп пиктограмм расположена на отдельной странице палитры компонентов. После запуска Delphi активной является страница Standard палитры компонентов. Выбрать другую страницу можно щелчком на вкладке с соответствующим названием. Например, при создании приложений предназначенных для работы с базами данных, как правило, бывают необходимы компоненты страниц Data Access и Data Controls.

При первом запуске Delphi все компоненты представлены на соответствующих страницах палитры компонентов. Для того чтобы новый компонент можно было использовать при разработке приложения, он должен быть представлен в палитре компонентов. Компоненты можно переименовывать, удалять, добавлять новые и менять их расположение на страницах. Кроме того, сам порядок страниц может быть изменен. Все эти изменения производятся при помощи диалогового окна Palette.

Диалоговое окно Palette может быть открыто двумя различными способами. Для его открытия необходимо:

- Выполнить щелчок правой кнопкой мыши в любом месте палитры компонентов и выбрать в контекстном меню команду Properties.
- Выбрать команду Configure Palette в меню Component.

2.6. Проектировщик форм

Как вы уже, наверное, знаете, каждое Windows-приложение выполняется в собственном окне — главном окне соответствующего приложения. Delphi назначает главное окно для каждого приложения автоматически. Разработка приложения всегда начинается с создания нового проекта. Для каждого нового проекта в IDE автоматически отображается окно проектировщика форм, которое представляет собой главное окно вашего будущего приложения и по умолчанию имеет имя Form1.

Главное окно — это первое, что вы видите после запуска приложения. Если пользователь закрывает это окно, тем самым он закрывает приложение. Проект может содержать несколько форм. Установить, какая из них будет главной, можно в диалоговом окне Project /Options, вызываемом командой Options меню Project.

Форма (Form1) является Windows-окном, поэтому она может быть увеличена, уменьшена, увеличена до размеров экрана или свернута в пиктограмму. При визуальном проектировании приложений в Delphi, вы можете изменять размер формы и размещать в ней некото-

рые компоненты. Для того чтобы поместить компоненты в форме, необходимо сначала выполнить щелчок на пиктограмме нужного компонента, а затем – в том месте проектировщика формы, где должен быть расположен компонент.

При выполнении приложения форма будет по умолчанию отображаться в том же виде, как и при проектировании, т. е., будет иметь те же размеры и содержать те же компоненты. Вы можете изменить размер формы, ее расположение на экране и т. д. в соответствии со своими требованиями.

Проектировщик форм содержит контекстное меню, вызываемое щелчком правой кнопкой мыши в любом месте формы, посредством которого можно выполнять определенные команды редактирования компонентов.

2.7. Окно редактора кода

Окно редактора кода имеет заголовок Unit1.pas и находится позади окна проектировщика форм.

Редактор кода и проектировщик форм тесно связаны между собой, но об этом будет сказано ниже.

В редакторе кода могут быть открыты несколько файлов. Каждый открытый файл размещается на отдельной странице, а его название отображается в верхней части окна на отдельной вкладке.

2.8. Окно свойств (инспектор объектов)

Инспектор Объектов позволяет определять свойства и поведение объектов, помещенных на форму, т. е. он используется для изменения характеристик любого объекта на форме. Кроме того, и для изменения свойств самой формы.

Инспектор Объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница – это список свойств, вторая – список событий. Если нужно изменить что-нибудь, связанное с определенным компонентом, то вы обычно делаете это в Инспекторе Объектов. К примеру, вы можете изменить имя и размер компонента TLabel, изменяя свойства Caption, Left, Top, Height, и Width.

Вы можете использовать закладки внизу Инспектора Объектов для переключения между страницами свойств и событий.

ГЛАВА III. СОЗДАНИЕ СИСТЕМЫ БАЗЫ ДАННЫХ В DELPHI

3.1. Выбор задачи автоматизации и их формулировка

В данном пункте приводится формулировка, т. е. описание задачи, которая решается с помощью системы базы данных.

Пусть предприятие намерено автоматизировать учет складского хозяйства. Тогда из всевозможных задач выберем задачу складского учета. Причем из всего процесса учета – учет поступления материалов.

Характер процесса складского учета следующий. В складском хозяйстве учет материалов начинается с начала месяца и состоит из следующих операций и процессов:

- перенос остатка с предыдущего месяца на текущий;
- учет и регистрация поступления материалов в течение месяца;
- учет и регистрация расхода материалов в течение месяца;
- определение остатка в конце месяца.

Для учебной цели из всего набора процессов учета материалов на складе выберем процесс учета поступлений.

Описание задачи. На склад ежемесячно поступают материалы, перечень материалов может меняться. Поступление этих материалов происходит поэтапно.

Требуется автоматизировать учет поступающих на склад материалов и хранение данных о них. Основным требованием к решению данной задачи является:

- минимальность размера системы;
- максимальность быстродействия;
- максимальность удобства для работы оператора.

Для автоматизации данной задачи необходимо создать систему. Данная система будет работать с данными о материалах, поступающих на склад. Поэтому тип системы является системой базы данных (СБД). Она состоит из главных компонентов БД и СУБД, которая будет работать с БД.

Следует еще отметить, что для цели облегчить понимание процесса построения БД и тем более СБД полный вариант технологий или методологии создания СБД не приводится. Изложение такого усеченного варианта создания системы является дидактическим приемом, позволяющим легко освоить неопытным программистам или студентам основной смысл теории практики базы данных.

Поэтому теория и методика проектирования и создания БД и СБД, изложенные в предыдущем разделе, являются теоретической предпосылкой для развития, полученных здесь знаний в будущем.

Таким образом, материалы данного раздела могут быть изучены независимо от предыдущего раздела.

3.2. План работы по созданию СБД

Исходя из технологии создания СБД, вытекает следующий порядок:

1. Определить архитектуру СБД, а также среду и технологию ее создания;
2. Создать БД;
3. Создать СУБД.

Согласно данному плану начнем создание БД.

Однако в связи с тем, что и БД и СУБД состоят из ряда компонентов, прежде чем программировать систему необходимо предварительно определить основные компоненты будущей системы и их назначения, а затем планировать последовательность создаваемых компонентов по каждому разделу, т. е. БД и СУБД.

Таким образом, для БД следует определить необходимое количество баз данных и их состав, которые необходимы для эффективности функционирования СБД.

Для СУБД определяем, сколько потребуется экранных форм и отчетов, которые необходимы для удобной работы пользователя.

Наиболее правильный путь создания системы базы данных является построение БД, а затем программирование форм просмотра базы.

Итак, проектирование (т. е. изготовление) системы начнем с создания БД.

3.3. Построение базы данных

3.3.1. Проектирование базы данных

3.3.1.1. Проектирование концептуальной модели базы данных

Изучение процессов приема, регистрации и хранения материалов на складе показывает, что для работы удобны данные о материалах и процессах, представить в виде двух взаимосвязанных таблиц.

В одной таблице представим данные о материалах, с которыми

склад работает. Это список материалов (ассортимент материалов), которые на текущий момент либо имеются на складе, либо поступят на склад. Каждый материал в этой таблице задается названием, единицей измерения и ценой. Данную таблицу назовем таблицей материалов – “Материалы”, ее структура представлена на рис. 3.1.

В другой таблице будут фиксироваться данные о тех материалах, которые поступили на склад. Каждый материал в этой таблице задается также названием, номером приходного ордера, датой поступления и количеством поступления по данному приходному ордеру.

Таблица 3.1.

Материалы

Название материала	Единица измерения материала	Цена материала

Рис. 3.1. Структура таблицы базы данных (Первая таблица или мастер-таблица)

Цена и единица измерения материалов в данной таблице не задается, т. к. эти данные зафиксированы в первой таблице. Эту таблицу назовем таблицей поступления материалов – “Поступление материалов” и структура приведена на рис. 3.2.

Таблица 3.2.

Поступление материалов

Номер приходного материала	Название материала	Дата поступления	Количество поступившего материала

Рис. 3.2. Структура таблицы базы данных (Вторая таблица или деталь-таблица)

В данном варианте формализации процесса поступления и учета материалов на склад таблица “Материалы” выполняет первичную функцию (ведущая), а таблица “Поступление материалов” – вторичную функцию (подчиненная).

В состав данных о материалах и об их поступлениях в соответст-

вующие таблицы можно включить информацию о других особенностях материалов. Например, в состав второй таблицы можно включить стоимость поступивших материалов.

Отметим, что эти недостающие данные могут быть вычислены позже из имеющихся в таблицах данных.

Таким образом, для учебной цели состав данных достаточен.

Один и тот же материал может поступать по частям несколько раз в течение месяца или года. Поэтому одной строке первой таблицы может соответствовать несколько строк во второй таблице. В связи с этим при проектировании физической модели необходимо показать тип отношения между этими таблицами.

3.3.1.2. Проектирование физической модели (представления) базы данных

Теперь рассмотрим вариант физического представления таблиц БД или физическую модель БД.

При переводе таблицы с реального на физическое представление структура остается инвариантной, т. е. структура таблицы остается, неизменной. Однако параметры реальной таблицы приобретают конкретные значения, т. е. интуитивно понятные свойства и характеристики таблицы на физическом уровне представления получают однозначное значение (рис. 3.3).

Таблица 3.3.

Материалы

Назначение поля таблицы	Конструкция поля таблицы		
	Название	Тип	Длина
Заносится название материалов, которые на склад поступили или поступят	Material	Строка	20
Единица измерения материала	Edizm	Строка	10
Цена единицы материала	Price	Целочисленный	

Таблица 3.4.

Поступление материалов

Назначение поля таблицы	Конструкция поля таблицы		
	Название	Тип	Длина
Номер прихода, материала подтупленного на склад	N_Prih	Автоинкремент	20
Название материала	Material	Строка	20

Дата поступления материала	DatPrih	Дата	
Количество материала	Kolvo	Целочисленный	

Рис.3.3. Структура физической модели таблиц, разрабатываемой базы данных

Соответствие между реальной и физической структурой таблиц БД является однозначной.

Структуру и параметры таблицы БД продемонстрируем на примере вышеприведенных таблиц.

Связь или отношения между таблицами "один – ко многим", т. е. одному материалу в таблице "Материалы" может соответствовать более одной записи в таблице "Поступление материалов".

3.3.2. Создание базы данных

3.3.2.1. План создания базы данных

Для того, чтобы начать процесс создания базы данных необходимо разработать план ее создания, который является частью общего плана создания СБД.

Пусть исходное задание для создания БД формулируется таким образом:

необходимо построить БД, которая состоит из двух таблиц и одна из них является мастер-таблицей, а другая – деталь-таблицей. Между ними отношение "один – ко многим".

Отсюда вытекает следующий план создания БД.

Шаг 1. Выбор технологии создания.

Программную систему, в т. ч. СБД (БД и СУБД) в Delphi можно построить в двух вариантах или режимах:

- в консольном режиме, в стиле традиционного процедурного программирования;
- в режиме RAD (Rapid Application Development – быстрая разработка приложений).

Последний режим еще называют интерактивным режимом разработки программных систем. Кроме того, следует отметить, что при более широком толковании эти режимы соответствуют технологиям создания систем в Delphi.

Для построения БД мы выбираем режим RAD, в качестве инструментов реализации выступает интегрированная среда Delphi.

Шаг 2. Определение и регистрация места хранения БД (т. е. ус-

тановление архитектуры БД), для этого следует выполнить:

- 1) определение места расположения БД: на своем компьютере, на другом компьютере (или сервере);
- 2) определение типа БД: для Delphi стандартный (т. е. DBASE, PARADOX, FoxPro);
- 3) определение псевдонима (или алиаса) БД;
- 4) регистрация алиаса и адрес местоположения БД.

Создание или конструирование БД. Для этого следует выполнить:

Шаг 3. Конструирование структуры таблицы БД.

1. Конструирование мастер-таблицы БД:

- 1) Определение типа таблицы.
- 2) Определение (спроектировать структуры и параметры) физической модели таблицы. Установление признаков и параметров первичной (ведущей) таблицы: установка первичного индекса сортировки.
- 3) Сохранить по выбранному адресу созданную таблицу.

2. Создание структуры деталь-таблицы БД.

Создание структуры второй таблицы БД происходит аналогично первой. Отличается лишь параметрами установки вторичного индекса сортировки.

Шаг 4. Конструирование (формирование) БД из этих таблиц путем установления связи целостности между ними.

Конструирование БД из созданных таблиц сводится к следующим действиям:

- 1) выполнение вторичной индексации;
- 2) установление отношения целостности путем установления отношения типа: "один – ко многим" между созданными таблицами (отношение соподчиненности).

3.3.2.2. Регистрация базы данных в Delphi

Определение БД. БД – это совокупность файлов: таблиц, индексов и триггеров, отражающие связи между таблицами.

БД состоит из:

- таблиц БД;
- индексов;
- примечаний (мемо-полей) и т. д.

Для хранения одной таблицы создается отдельный файл. Такие же отдельные файлы создаются для хранения индексов таблицы и мемо-полей. Все эти файлы должны находиться и храниться вместе, ина-

че СУБД может работать некорректно. Поэтому все компоненты БД следует хранить вместе.

Отсюда, создание БД в Delphi (или для работы в Delphi) начинается с задания места расположения и хранения БД. Таким образом, первым шагом процесса создания БД является определение местоположения будущей, создаваемой БД.

Задание местоположения БД. В общем случае БД может находиться на определенном компьютере, носителе и в директории.

В качестве которого может быть автономный компьютер самого пользователя, сервер сети или любой другой компьютер сети.

Если БД находится на диске того же компьютера, на котором СУБД, то такую БД называют локальной.

Это можно сделать путем выделения директории (или путем создания папки) для компонентов БД.

Директорию можно создать на своем компьютере, на сервере. Кроме того, директория может быть в корневом каталоге или в подкаталоге, например

- 1) C:\Primer;
- 2) C:\SYSTEMA\STUDENT\.....\Primer.

Обычно в режиме локальных баз данных работают БД (или таблицы БД), созданные в формате СУБД Paradox, dBase и FoxPro.

Присвоение БД псевдонима. Как было отмечено, что БД состоит из совокупности компонентов. Поэтому они должны иметь общее название. В технологии СУБД принято в качестве названия БД присвоить ей *псевдоним* (или еще называют: алиас, alias) и тогда обращение из утилит и программы СУБД к БД будет осуществляться через ее *псевдоним*.

В связи с отмеченным, в утилите *BDE Administrator* Delphi должен быть:

- зарегистрирован псевдоним БД;
- указан адрес (путь) к БД.

Создание БД в Delphi или для работы в Delphi начинается с задания места расположения и хранения.

Таким образом, рассмотрели первый шаг процесса создания БД.

Для нашего примера выполним первого шага процесса создания БД будет выглядеть таким образом.

Пусть наша база данных будет находиться в директории в корне-

вом каталоге диска "C:" на нашем компьютере (или самого пользователя). Название директории (или папки) – **"PRIMER"**. Тогда полное название созданной директории, вместе с путем папки будет называться **"C:\PRIMER"**. Если БД была бы создана на диске D:, то она имела бы название **"D:\PRIMER"**.

Базе данных, размещенной в **"C:\PRIMER"**, для удобства можно присвоить любое рабочее название, например, **"BD"**, но мы назовем (присвоим псевдоним создаваемой БД) создаваемую БД тем же именем, т. е. **"C:\PRIMER"**.

Таким образом, псевдоним (или алиас) БД **"PRIMER"**.

Регистрация БД. Регистрация БД в Delphi осуществляется с помощью его утилиты (или модуля) *BDE Administrator*. Поэтому для регистрации нашей БД **"C:\PRIMER"** необходимо запустить *BDE Administrator*.

Запуск утилиты *BDE Administrator* осуществляется в одном из двух вариантах таким образом:

1. Непосредственно с рабочего стола Windows:

(*Пуск/Программы/Borland Delphi 3/BDE Administrator*).

В данном случае приведена последовательность открытия меню и подменю (или подача команд из меню).

2. Сначала запускается главный модуль Delphi, который на экран отображает IDE. Структура IDE запущенного Delphi приведена на рис.2.1.

В самом верхнем ряду IDE имеется главное меню Delphi. В нем открываем меню *DateBase* и там же выберем команду *Exproler*. По этой команде запускается утилита *BDE Administrator*.

После запуска утилиты *BDE Administrator*, в нем выполняются следующие действия.

Выберем в главном меню элемент *Object|New*. В появившемся окне (рис. 3.4) оставим тип создаваемой БД без изменений (**STANDARD**) и нажмем **OK**.

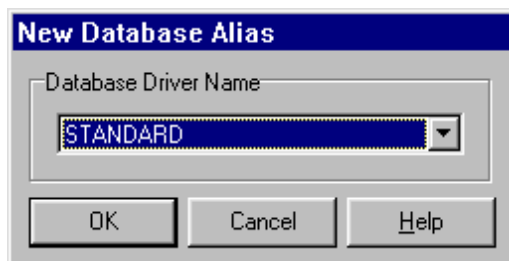


Рис. 3.4. Окно выбора типа драйвера базы данных

В левом окне появившегося окна администратора БД мы увидим строку с именем STANDARD1. Изменим это имя на **"PRIMER"**. Для этого щелкнем мышью на названии БД и введем новый текст.

В правом окне приведены параметры БД. Оставим их без изменения, изменив лишь PATH. Этот параметр указывает путь к каталогу, в котором будет расположена БД. Можно ввести путь вручную, но лучше воспользоваться средствами администратора: для этого нужно щелкнуть по полю PATH, и нажав на появившуюся в правом углу поля кнопку , следует выбрать каталог C:\PRIMER и нажать кнопку Ok (рис. 3.5).

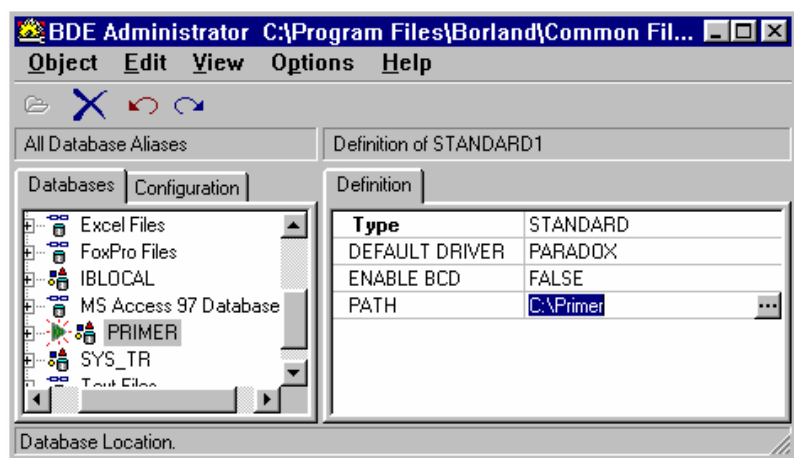


Рис. 3.5. Окно параметров псевдонима базы данных

Теперь необходимо запомнить определение псевдонима. Для это-

го в левом окне администратора БД необходимо щелкнуть по имени псевдонима правой кнопкой мыши и во всплывающем меню выбрать элемент *Apply*. В появившемся диалоговом окне, в котором спрашивается, собираемся ли мы запоминать изменения для псевдонима, необходимо нажать кнопку *Ok*. Затем следует выйти из утилиты *BDE Administrator*. Теперь создание псевдонима завершено и к нему можно обращаться из других утилит и приложений. Однако каталог, на который ссылается псевдоним БД, еще пуст. Необходимо создать таблицы базы данных.

3.3.2.3. Создание таблиц базы данных

Создание БД представим таким образом.

Согласно концептуальному проекту база данных состоит из двух взаимосвязанных таблиц. Причем первая таблица является главной. Поэтому БД создадим в следующей последовательности:

- 1) формирование структуры таблицы 3.1, которая является конструированием мастер-таблицы базы данных;
- 2) формирование структуры таблицы 3.2, которая является конструированием деталь-таблицы базы данных;
- 3) установление связи между таблицами, которое соответствует созданию базы данных из сформулированных двух таблиц.

Рассмотрим каждый этап последовательности создания базы данных более подробно.

Конструирование мастер-таблицы. Для создания таблиц базы данных необходимо запустить утилиту *Database Desktop (DBD)* (*Пуск/Программы/Borland Delphi 3/Database Desktop*). После запуска утилиты установим рабочий псевдоним утилиты. Это псевдоним, с которым утилита работает по умолчанию. Если рабочий псевдоним не установлен, придется при работе с DBD всякий раз указывать псевдоним, что отнимает время.

Для установки рабочего псевдонима нужно выбрать элемент главного меню *File|Working Directory* и в выпадающем списке *Aliases* выбрать имя псевдонима *PRIMER*, после чего нажать кнопку *Ok*.

Конструирование структуры таблицы. Для создания таблицы БД нужно выбрать элемент главного меню *File|New|Table*. В появившемся окне *Create Table* оставляем без изменения тип создаваемой таблицы (*Paradox 7*) и нажимаем кнопку *Ok*. После этого появится окно определения структуры таблицы БД (рис. 3.6).

Каждая строка таблицы соответствует полю. Назначения столбцов

- *Fields Name* – имя поля;
- *Type* – тип поля;
- *Size* – размер поля (для строковых полей, поскольку иные поля подразумевают размер, определяемый типом поля);
- *Key* – содержит звездочку «*», если поле входит в состав первичного ключа.

Если в первичный ключ входит несколько полей, они должны определяться в той последовательности, в которой они присутствуют в первичном ключе. Кроме того, все поля, входящие в состав первичного индекса, должны определяться перед иными полями, т. е. быть в списке полей наверху.

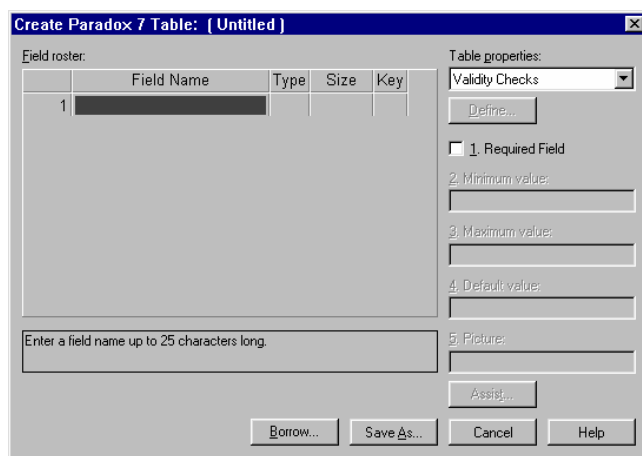


Рис. 3.6. Утилита Database Desktop: окно определения структуры таблицы БД

Определим поля, входящие в таблицу "Материалы". Введем Material в столбец *Field Name*. Для того, чтобы определить тип поля, щелкните по столбцу *Type* и нажмите клавишу пробела. В ответ на это будет выдан список типов полей, из которых необходимо произвести выбор нужного типа (рис. 3.7).

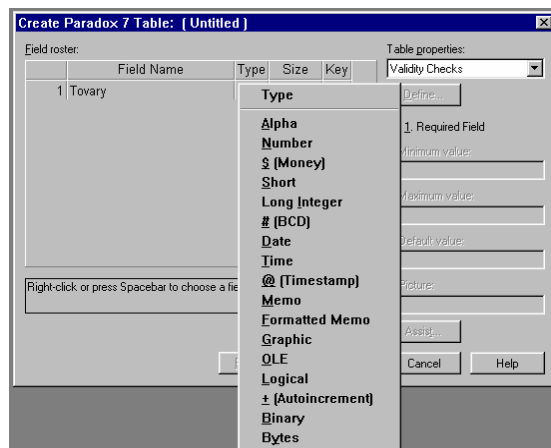


Рис. 3.7. Выбор типа поля

Всевозможные типы полей СУБД Paradox приведены в таблице 3.5.

Таблица 3.5.

Значение типов полей БД

Тип поля	Обозначение	Хранимые значения
Alpha	A	Символьные значения длиной до 255 символов.
Number	N	Числовые значения с плавающей точкой в диапазоне $-10^{307} \dots +10^{308}$. Точность до 15 значащих цифр.
Money	\$	Аналогичен типу Number, но предназначен для хранения денежных сумм. Число знаков после запятой по умолчанию – при показе значения выводится знак денежной единицы.
Short	S	Целочисленные значения в диапазоне $-32\,767 \dots 32\,767$.
LongInteger	I	Целочисленные значения в диапазоне $-2\,147\,483\,648 \dots 2\,147\,483\,647$.
BCD	#	Числовые значения, в т. Ч. И дробные, в двоично-десятичном формате. Обеспечивает исключительную точность при работе с большим числом знаков в дробной части. Применяется в вычислениях, где важна точность (финансовые, научные приложения). Для проведения вычислений требует больше времени, чем для числовых полей иных типов.
Date	D	Значения даты (в диапазоне от 01.01.9999 до н.э. до 31.12.9999).
Time	T	Значения времени.
Timestamp	@	Значения даты и времени.

Memo	M	Строковые значения длиной более 255 символов. Максимальная длина не ограничена. От 1 до 240 символов могут храниться вместе с таблицей БД; остальные хранятся в виде Мемо-файла (расширение .MB)
Formatted Memo	F	Аналогично мемо-полно, но может хранить форматированные тексты, в которых фрагменты текста представлены разным шрифтом, цветом и стилями.
Graphic Fields	G	Графические изображения в формате файлов BMP, .PCX, .TIF, .GIF, .EPS, которые при хранении преобразуются к формату. BMP. Хранятся отдельно от основной таблицы БД.
OLE	O	Информация в форматах, поддерживаемых технологией OLE (Object Linking and Embedding) фирмы Microsoft.
Logical	L	Логические значения («True», «False»). Высота букв не имеет значения.
Autoincrement	±	Автоинкрементное поле. Значения доступны только для чтения. Обычно – ключевое поле в составе первичного ключа. При добавлении новой записи значение поля вычисляется автоматически таким образом, чтобы в одной и той же таблице не было одинаковых значений. Значения поля из удаленных записей повторно не используются.

Продолжение табл. 3.5

Тип поля	Обозначение	Хранимые значения
Binary	B	Произвольные двоичные значения. Должны интерпретироваться приложениями пользователя. DBD не интерпретирует значения этих полей. Хранятся в отдельных от основной таблицы .MB -файлах. Длина не определена.
Bytes	Y	Произвольные двоичные значения, интерпретируемые приложениями пользователя, длиной от 1 до 240 байт. Хранятся вместе с таблицей БД.

Для того чтобы определить тип поля Material, выберем *Alpha* и затем в столбце *Size* укажем значение 20. В столбце *Key* поместим звездочку, означающую, что данное поле входит в состав первичного ключа. Для этого нажмем любой символ на клавиатуре. Повторное нажатие любого символа снимает отметку звездочкой в столбце *Key*.

Мастер таблицы по первичному ключу индексируется.

Введем определения и других столбцов таблицы Materialy (рис. 3.8).

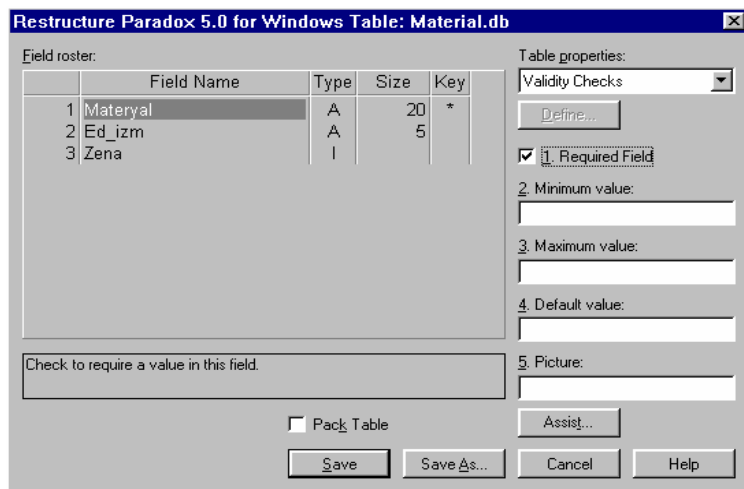


Рис. 3.8. Определение структуры таблицы Material

Согласно условиям задачи все поля данной таблицы должны заполняться. Поэтому для каждого поля определим требование обязательного заполнения поля значением. Для этого, переходя от поля к полю, включим переключатели *Required Field*. Другие поля служат для наложения ограничений на значение поля

- *Minimum value* – определяет минимальное значение поля
- *Maximum value* – определяет максимальное значение поля
- *Default value* – определяет значение поля по умолчанию
- *Picture* – определяет шаблон изображения поля.

Для формирования шаблона следует нажать кнопку *Assist*.

Отсутствие значения в одном из полей означает отсутствие ограничений на значение поля.

Заполнение и запоминание таблицы. После конструирования структуры таблицы в *Database Desktop* можно заполнить таблицу. Отметим, что заполнение таблицы в русском алфавите может не удасться, тогда следует использовать латинский алфавит.

Чтобы запомнить сохраненную таблицу на диске, следует нажать кнопку *Save As*. Затем, в появившемся окне, следует указать имя таблицы (рис. 3.9). При желании можно указать каталог или псевдоним, отличные от принятых по умолчанию. Напомним, что по умолчанию принимается рабочий каталог или каталог, определяемый рабочим

псевдонимом.

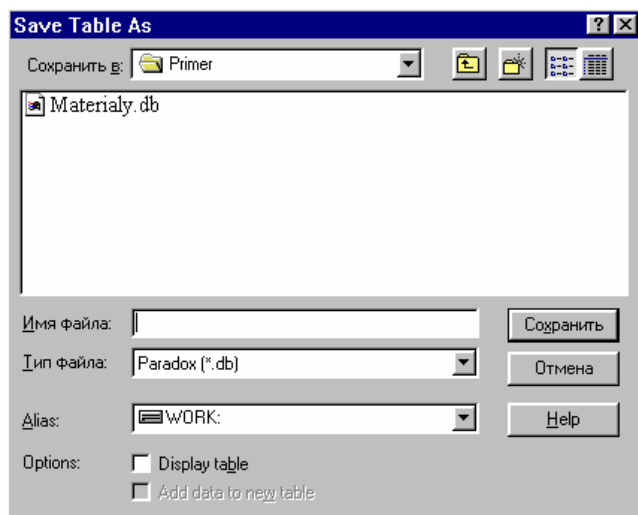


Рис. 3.9. Окно сохранения таблицы

После того как мы определим имя создаваемой таблицы (Materialy) в каталоге C:\PRIMER (он назначен псевдониму PRIMER, используемому нами в качестве рабочего псевдонима), будет создан файл Materialy.DB и файлы индексов с соответствующими расширениями.

Реструктурирование созданной таблицы. Если в структуру существующей таблицы БД необходимо внести изменения, следует выбрать элемент меню *File | Open | Table*, в появившемся диалоговом окне выбрать имя таблицы и нажать кнопку Ok. Будет показано содержимое таблицы (на рис. 3.10, в таблице Materialy отсутствуют записи, что не удивительно поскольку мы их еще не вводили).

В том случае, когда мы хотим ввести новые записи в таблицу прямо в DBD (а также изменить значения в некоторых записях или произвести удаление), следует нажать кнопку F9. После этого таблица будет переведена в режим внесения изменений.

Чтобы изменить структуру таблицы, выберите элемент меню *Table | Restructure*. Вслед за этим будет показано диалоговое окно для определения структуры таблицы.



Рис. 3.10. Содержимое таблицы Materialy – записи еще не введены

Конструирование деталь-таблицы. Определим структуру второй таблицы "Поступление материалов" (рис. 3.11), которая является деталь-таблицей. Способ конструирования структуры этой таблицы почти аналогичен мастер – таблице, кроме следующих отличий.

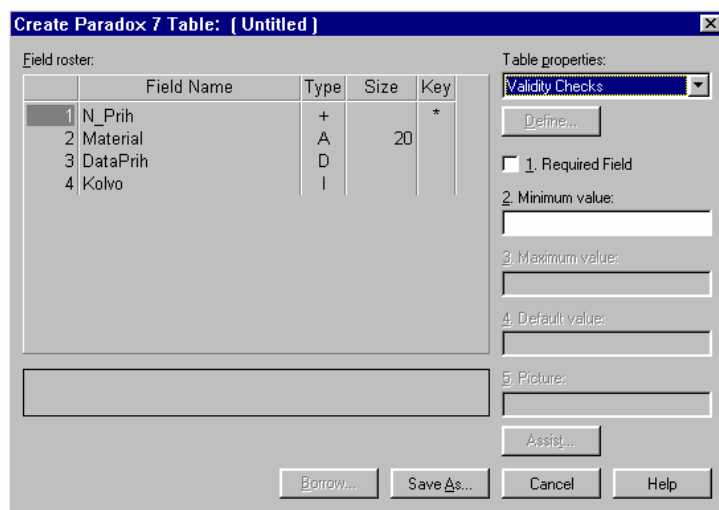


Рис. 3.11. Структура таблицы Prihod

Всем полям назначим атрибут *Required* (требование обязательного существования значения у поля на момент его запоминания в БД), кроме поля N_Prih, поскольку это поле автоинкрементальное, заполнение его значением производится автоматически при запоминании новой записи.

Создадим индекс по полям “Дата прихода”, “Материал”. Для этого в комбинированном списке Table Properties (в правом верхнем углу окна) выберем элемент Secondary Indexes. После этого диалоговое окно приобретет вид, показанный на рис. 3.12.

Чтобы определить новый индекс, нажмем кнопку *Define*. В появившемся диалоговом окне в поле *Fields* содержится список полей определяемой нами таблицы. Поле *Index Fields* предназначено для хранения полей, входящих в создаваемый индекс. Чтобы скопировать конкретное поле из списка *Fields* в список *Index Fields*, нужно нажать кнопку с изображением правой стрелки. Последовательность добавления полей в список важна, она определяет порядок чередования полей в списке. После того как мы поместили нужные поля в список *Index Fields* (рис. 3.13), нажмем кнопку *Ok*.

В появившемся окне запрашивается имя индекса (рис. 3.14). Следует ввести имя и нажать *Ok*.

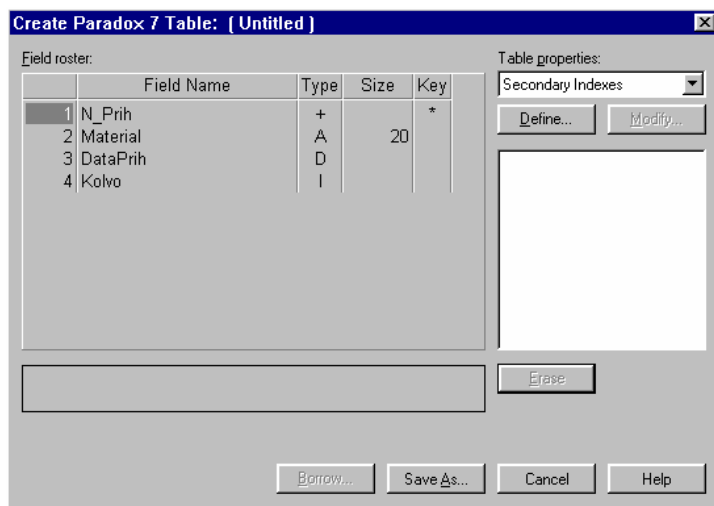


Рис. 3.12. В правой части окна появился элемент Secondary Indexes

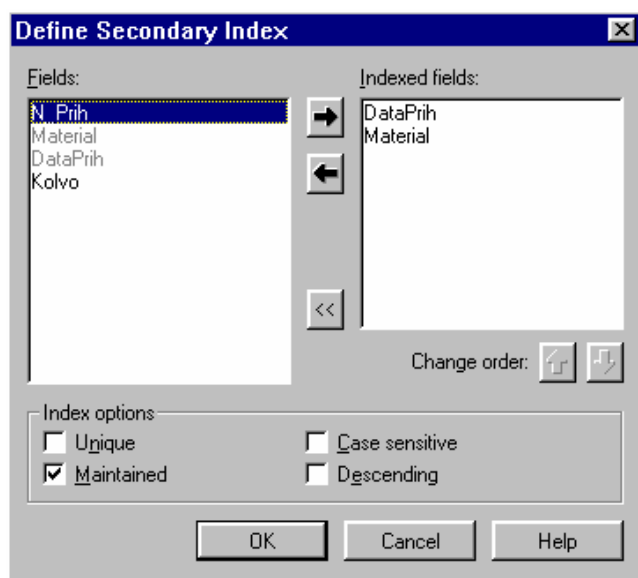


Рис. 3.13. Определение полей, входящих в состав индекса

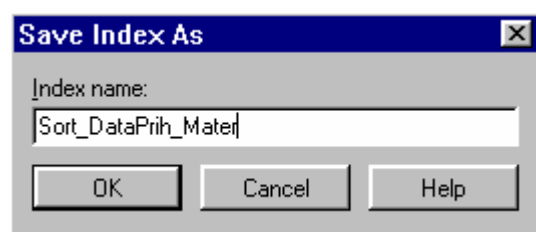


Рис. 3.14. Окно сохранения индекса

Не рекомендуется составлять название индекса только из имен полей, поскольку такой способ именования индексов используется автоматически при создании ссылочной целостности между таблицами (см. следующий раздел).

Как видно на рис. 3.15 после добавления нового индекса его имя появилось в списке индексов.

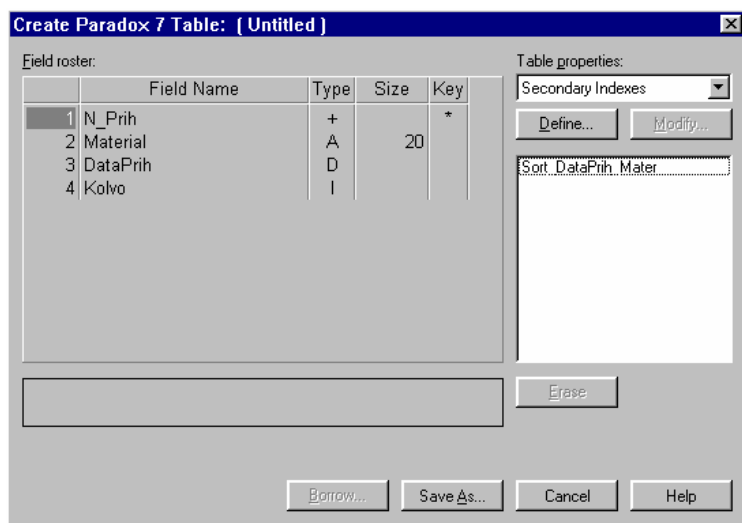


Рис. 3.15. Список индексов, определенных для таблицы Prihod

Впоследствии, щелкнув по имени индекса, мы можем его удалить (кнопка *Erase*) или изменить (кнопка *Modify*).

Сохраним созданную нами таблицу под именем Prihod.

Конструирование базы данных

Теперь конструируем БД в целом из таблиц, структуры которых созданы, из конструированных таблиц.

Как известно из постановки задачи, таблицы "Материалы" и "Приход материалов" находятся в отношении "один – ко многим", т. е. с одной записью в таблице Materialy может быть связано несколько записей по приходу того же материала в таблице Prihod. В качестве поля связи выступает поле Material, присутствующее в обеих таблицах.

Определим ссылочную целостность между данными таблицами. Ссылочная целостность в Paradox определяет, во-первых, связь между таблицами, а во-вторых, вид каскадных воздействий.

Откроем таблицу Prihod (элемент меню *File | Table | Open*) и затем войдем в режим изменения структуры таблицы (*Table | Restructure*). В выпадающем списке *Table Properties* выберем элемент *Refrential Integrity* и нажмем кнопку *Define*. В появившемся диалого-

вом окне (рис. 3.16) в списке *Fields* показаны поля таблицы Prihod, а в списке *Tables* – таблицы базы данных PRIMER.

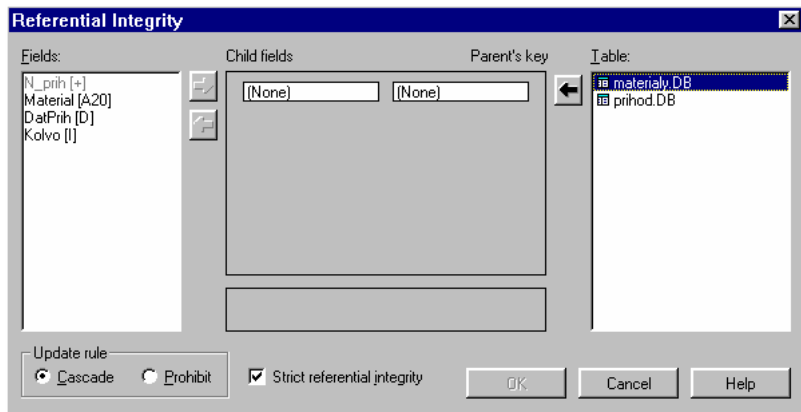


Рис. 3.16. Окно для создания ссылочной целостности

Выберем в списке *Fields* поле *Material* и нажмем кнопку с изображением стрелки вправо. Название *Material* будет записано в поле *Child Fields* (поле внешнего ключа дочерней таблицы).

Выберем в списке *Tables* таблицу *Materialy* и нажмем кнопку с изображением стрелки влево. В поле *Parents Key* (ключ родительской таблицы) будут показаны поля из первичного ключа таблицы *Materialy*. В данном случае это поле *Material*.

Переключатели *Update rules* определяют вид каскадных воздействий на таблицу *Prihod* при изменении значения поля связи в таблице *Materialy* или при удалении записи в таблице *Materialy*

- *Cascade* – каскадные изменения и удаления подчиненных записей в таблице *Prihod*;
- *Prohibit* – запрет на изменение поля связи или удаление записи в таблице *Materialy*, если для данной записи есть связанные записи в таблице *Prihod*. Выберем *Cascade* (рис. 3.17) и нажмем кнопку *Ok*.

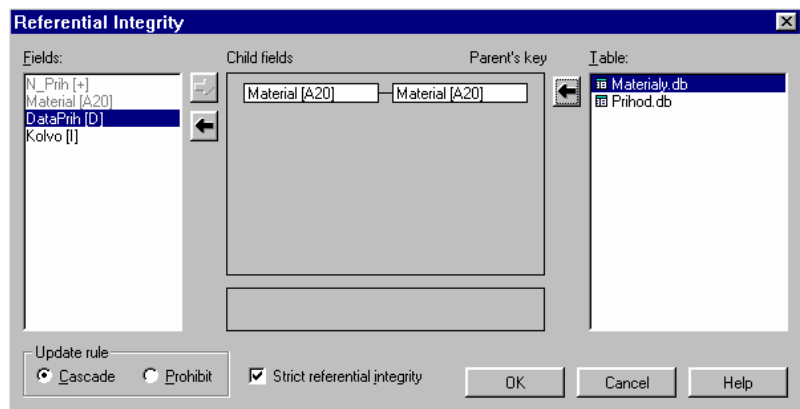


Рис. 3.17. Определение условий ссылочной целостности

Будет запрошено имя – в Paradox ссылочные целостности имену-
ются. Введем имя, например, *Materialy_Prihod_Integrity* и нажмем
кнопку *Ok*. Теперь имя созданной ссылочной целостности будет поме-
щено в список.

Запомним изменения в таблице *Prihod* (кнопка *Save*) и заново
войдем в режим реструктуризации таблицы *Prihod* (*Table|Restructure*).
В выпадающем списке *Table properties* выберем элемент *Secondary
Indexes* (индексы таблицы, кроме индекса, построенного по определе-
нию первичного ключа). В списке индексов увидим, что появился но-
вый индекс с именем *Material* (по полю *Material*). Этот индекс постро-
ен автоматически по неявному определению внешнего ключа при соз-
дании ссылочной целостности (рис. 3.18).

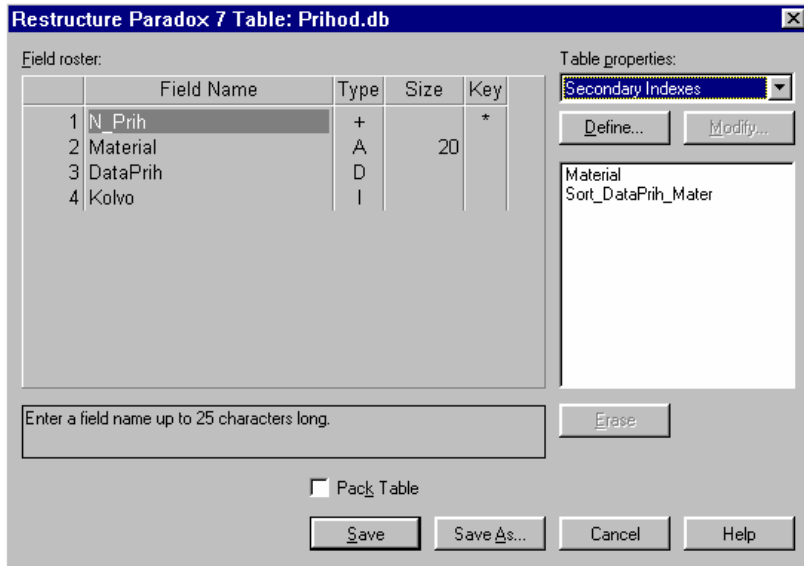


Рис. 3.18. Список индексов таблицы Prihod после создания ссылочной целостности

На этом процесс построения БД закончится, поэтому можно выйти из режима реструктуризации и покинем **DBD**. После этого перейдем к разработке простейшего приложения для работы с созданными таблицами.

Заключение по разделу

Структура материала по созданию БД следующая:

1. Формирование структуры первой таблицы. Ее характеристики таковы – она является мастер-таблицей и для неё создан первичный ключ, первичная индексация по первичному ключу.
2. Формирование структуры второй таблицы. Ее характеристики таковы – она является деталь-таблицей, создан вторичный ключ, вторичная индексация.
3. Между таблицами установлено отношение “один – ко многим”.

ГЛАВА IV. СОЗДАНИЕ СУБД В DELPHI

4.1. Определение состава СУБД

СУБД состоит из совокупности экранных и отчетных форм и процедур обработки данных вводимых в БД и выводимых из БД. При программировании СУБД в среде Windows СУБД называют приложением.

Чтобы определить перечень экранных форм, рассмотрим порядок работы пользователя, т. е. задаем поведение пользователя нашей системы.

Пусть порядок работы пользователя задается следующей схемой.

Схема работы пользователя.

Интерфейс пользователя происходит через главное меню. Поэтому система должна иметь главное меню.

Для ввода информации в базу необходима отдельная экранная форма. Базу следует просматривать на отдельной экранной форме.

Кроме того, пользователю необходимо получить данные из базы в печатной форме, причем различные данные в различных формах. Все эти выходные данные называются отчетами или отчетными формами. Поэтому пользователю необходимо получать отчетные формы.

Таким образом, для получения отчетов необходимо предусмотреть несколько отчетных форм, требующих для полноценной работы пользователя.

Порядок работы пользователя на основе предусмотренных форм происходит следующим образом.

При запуске системы на экране должно отображаться главное меню системы. В меню отражен перечень команд-действий, которые может выполнить система. Если пользователь планирует ввод данных о поступивших материалах, то он из меню вызывает форму для ввода информации.

Для того чтобы просмотреть базу он может из меню вызвать форму просмотра базы. Для вывода данных в отчет для пользователя можно предусмотреть различные варианты. Пользователь из главного меню может вызвать форму просмотра базы и оттуда он может выбирать данные и форму отчета. Далее сгенерированный вариант отчета выдается на печатное устройство. Для того чтобы достичь гибкости можно предусмотреть варианты формирования отчета в режиме (с формы) ввода информации и в режиме (с формы) просмотра базы.

Таким образом, из описанной выше схемы работы системы

“пользователь – система” видно, что для работы системы необходимы следующие формы:

- форма для меню;
- форма для ввода данных в базу;
- форма для просмотра БД;
- форма для отчета №1;
- форма для отчета №2;
- форма для отчета №3.

Выше была указана внешняя форма проявления СБД. Для того чтобы обеспечить работоспособность СБД необходимо обеспечить связь форм с БД: обработку данных БД из экранной формы. Что и будет сделано в следующих пунктах.

4.2. Проект и план создания компонентов СУБД

Кроме внешних форм проявления, СУБД еще должна иметь процедуры обработки и средства обеспечения доступа к БД. Вся эта совокупность компонентов, составляющая СУБД, должна быть создана и сохранена.

Следует отметить, что основой для проектирования СУБД является форма. Являясь основой и носителем всех остальных компонентов, форма служит фундаментом для процессов обработки данных. Теоретической основой такой схемы является объектно-ориентированная технология проектирования и программирования.

Однако на теоретических вопросах объектно-ориентированной технологии проектирования и программирования не будем специально останавливаться.

Теперь составим проект СУБД, т. е. установим состав, назначение и связь между компонентами СУБД. При этом будем исходить из указанной выше схемы работы пользователя.

Будем считать, что для работы пользователя достаточно трех отчетных форм. Поэтому для каждой экранной и отчетной формы должна быть выделена отдельная форма.

В Delphi информационная связь форм СУБД с БД осуществляется через определенные невизуальные компоненты. Поэтому для обеспечения связи экранных форм с БД в проекте используется специальный невизуальный компонент (контейнер) TDataModule, который также должен создаваться отдельно.

Таким образом, исходя из описанных эвристических предпосы-

лок, установим перечень необходимых для нашей задачи форм:

- форма для меню;
- форма для ввода данных в базу;
- форма для просмотра БД;
- форма для отчета № 1.
- форма для отчета № 2.
- форма для отчета № 3.
- компонента TDataModule.

Опыт подсказывает, что одним из наиболее удобных для разработчика порядок формирования форм и их компонентов является следующий.

- 1) Компонента TDataModule;
- 2) Форма для меню;
- 3) Форма для просмотра БД;
- 4) Форма для ввода данных в базу;
- 5) Форма для отчета № 1.
- 6) Форма для отчета № 2.
- 7) Форма для отчета № 3.

Формы, необходимые для создания СУБД, можно задать двумя способами:

1. Если заранее известно их количество, то на экране монитора и соответственно в хранилище (repository) проекта IDE Delphi заранее можно вывести нужное количество визуальных и не визуальных форм, т. е. в проект можно добавить столько форм, сколько требуется. С помощью команды *File|New Form* добавляются визуальные формы, а командой *File|New Data Module* – не визуальная. Отметим, что для нашей системы требуется всего шесть визуальных форм и одна компонента TDataModule.

2. Если заранее количество нужных форм для проекта не известно, то на экране монитора и соответственно в хранилище (repository) проекта IDE Delphi добавляется та форма, которая нужна для проекта на текущем этапе создания СУБД. Установление форм на экране осуществляется через команды *File|New Form* или *File|New Data Module*. Другими словами каждая форма в проект добавляется по мере необхо-

димости.

В данном случае добавление новых форм в хранилище проекта происходит с помощью команды добавления (*File|Add to Project*).

Таким образом, исходя из проекта, представим план работы по созданию СУБД для поставленной выше задачи автоматизации из экранных и отчетных форм и процедур обработки данных.

Сначала будем считать, что на экране выведено столько форм, сколько требуется для создания всех форм СУБД. Если форм не хватает, то доводим их до необходимого количества. Это делается путем повторения действия (*File|New Form*) столько раз, сколько требует количество форм и один контейнер командой *File|New Data Module*.

Теперь определим последовательность проектирования (конструирования) форм. Последовательность создания форм в общем случае произвольная, но мы все-таки выбираем вышеизложенную последовательность.

Данная последовательность оформления и создания форм упрощает процесс создания СУБД.

Создание СУБД начнем с построения контейнера TDataModule для связи с БД. Проектирование форм СУБД начнем с экранной формы, на которой будет отображено содержимое БД.

Проектирование СУБД происходит по каждой форме по отдельности путем установления связи с БД (т. е. с компонентом TdataModule) и созданием процедур обработки данных, вводимые с экранной формы или получаемые из БД.

Кроме того, требуется установление связи между компонентами различных форм. В одной из экранных форм следует предусмотреть команду завершения работы системы.

После определения набора компонентов и их размещения на форме, устанавливается их доступ к БД и создается процедура обработки данных для каждого компонента формы.

Из вышеприведенного определения видно, что СУБД также как БД состоит из множества компонентов. Все эти компоненты СУБД требуют формирования и программирования и их файлы также как и файлы БД следует хранить в определенном месте. Для этой цели можно использовать тот же каталог, что и для БД, т. е. C:\PRIMER, либо создадим в каталоге C:\PRIMER подкаталог APP и в нем храним разработанные приложения.

В общем случае приложение можно хранить в любом месте, однако, хранение его вместе с БД в одной папке, но все же в отдельной

поддиректории удобно для работы разработчика. Так и будем делать.

4.3. Создание СУБД

4.3.1. Способы организации доступа экранных форм и отчетных форм к базе данных

В данном пункте описываются вопросы организации доступа экранных форм (ЭФ) и отчетных форм (ОФ) к базе данных путем создания контейнера *TDataModule*.

Связь экранной формы с БД обеспечивается специальными компонентами, которые называются невидимыми компонентами. Эти элементы видны только в режиме конструирования, а в режиме работы системы эти элементы будут на экране невидимы.

Если СУБД состоит из одной экранной формы, то не нужно специального хранилища в виде контейнера для хранения невидимых компонентов. В этом случае невидимые компоненты можно расположить на той же экранной форме, которая является единственной.

В случае когда СУБД имеет несколько экранных форм, то их доступ следует осуществить двумя способами:

1. Доступ каждой формы (и ее компонентов) к БД можно осуществить автономно, независимо от остальных форм.
2. Второй способ предполагает не повторять для каждой формы организацию доступа к БД, а для всех экранных и отчетных форм организовать единый доступ через специальный компонент типа *TDataModule*. Компонент типа *TDataModule* представляет собой контейнер, в который можно помещать все невидимые компоненты *TTable*, *TQuery*, *DataSource*, которые необходимы для доступа к БД и т.д. Тогда все те формы, для которых потребуется доступ к БД, доступ получат через компонент *TdataModule*. Для этого для каждой экранной формы необходимо установить связь с данным компонентом, т. е. с *TdataModule*.

Таким образом, в случае множества ЭФ и ОФ для работы СУБД удобно создать компонент типа *TDataModule*, который является контейнером для невидимых компонентов, обеспечивающих доступ к базам данных.

Создание экземпляра *TDataModule* в среде Delphi осуществляется через пункт меню *File|New Data Module*. После чего, в палитре компо-

нентов Delphi на странице *Data Access* выберем мышью невизуальный компонент TTable (рис. 4.1), щелкнем на нем мышью и затем щелкнем мышью в контейнере. После этого, изображение компоненты останется в контейнере.

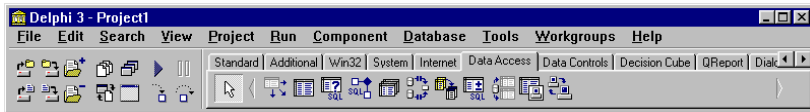


Рис. 4.1. Палитра компонентов Delphi

Невизуальным компонент TTable (как и другие компоненты, например, TQuery, TDataSource) называется потому, что он применяется для хранения и доступа к данным, а не для их визуализации – для этой цели применяются визуальные компоненты (TDBGrid, TEdit и др.).

После того, как мы разместили компонент TTable, установим его свойства. Для этой цели воспользуемся *инспектором объектов* (Object Inspector), который обычно помещается слева от формы. Если он не видим, его можно вызвать, нажав кнопку F11. *Инспектор объектов* позволяет устанавливать свойства того компонента, который выделен при помощи мыши. Выделим мышью компонент TTable.

Установим значение свойства *DatabaseName* в PRIMER (это является псевдонимом БД) при помощи выпадающего списка или введя его вручную. Установим значение свойства *TableName* (имя таблицы БД) в Materialy.DB при помощи выпадающего списка. После этого установим значение свойства *Active* в True. После этого произойдет реальное связывание компонента TTable (он по умолчанию имеет имя Table1) с реально существующей таблицей Materialy.DB.

Компонент TTable и компонент TQuery служат для хранения набора данных. Понятие набора данных несколько шире, чем понятие таблицы БД, поскольку набор данных может содержать:

- подмножество записей или полей таблицы БД (компоненты TTable, TQuery);
- записи, сформированные из нескольких таблиц БД (компонент TQuery).

Расположим в контейнере DataModule компонент TDataSource. Он служит в качестве связующего звена между невизуальными компонентами (в данном случае Table1) и визуальными компонентами, которые мы добавим в форму № 3 позднее. Поэтому компоненты

TDataSource часто называют *источниками данных*. Установим свойство *DataSet* (имя набора данных) компонента TDataSource в значение Table1 путем выбора из выпадающего списка.

TDataModule нужно сохранить под каким-либо именем (по умолчанию Unit5.pas). Имя модуля Unit, в котором описан TDataModule, добавляется в текст модулей unit всех иных форм приложения, которые будут использовать БД и TDataSource, расположенные в этом TDataModule. Это производится в главном меню среды Delphi, в элементе меню *File|Use Unit*.

В дальнейшем визуальные компоненты, работающие с данными набора данных (НД), должны в своем свойстве DataSource содержать имя соответствующего компонента TDataSource из TDataModule. При этом имя является составным: сначала идет имя компонента TDataModule и затем через точку – имя компонента TDataSource, например, в нашем случае – DataModule5.DataSource1.

4.3.2. Создание простых экранных форм

В данном пункте описываются вопросы создания простых вариантов экранных форм и установление их доступа к БД, а также организация процессов обработки данных.

Создание экранной формы для меню СУБД

Перейдем в форму № 1 (элемент меню *View|Forms*), выбрав из диалогового окна *Form1*, нажмем на кнопку *Ok*. Расположим в форме компонент MainMenu из страницы *Standard* палитры компонентов. Затем вызовем средства разработки меню (это можно сделать двумя способами):

- двойное нажатие на компонент MainMenu внутри формы № 1;
- переходим в окно Object Inspector при выделенном компоненте на форме. Щелкнем на свойстве Items и нажмем кнопку .

Находясь в Конструкторе меню Form1.MainMenu1 (это имя, присвоенное Delphi, по умолчанию созданному нами, перед этим компоненту TMainMenu), дадим названия пунктам меню (свойства *Caption* в *Object Inspector*) “Файл/Ввод/Просмотр/Отчет/Выход”. Для перехода на следующий уровень нажимаем на *<Enter>*. Для визуального разделения пунктов меню на отдельные группы после *Отчет* используем

несколько символов минус (–), назначать имя разделителю необязательно. Для пункта меню “Ввод” создадим подменю “Название материала/Поступление материала”. Для этого выделим его и щелкнем правой кнопкой мыши, в появившемся окне выберем пункт *Create Submenu*.

Теперь, когда меню создано, можно связать с каждым пунктом меню код, который будет выполняться при выборе данного пункта пользователем. Для создания процедуры обработки этого события – *OnClick*, находясь в Конструкторе меню, дважды щелкнем мышью на пункте меню. В появившемся окне Редактора кода процедуры обработки события *OnClick* напомним код, который будет выполняться при выборе этого пункта меню. Такого же результата можно достичь при помощи двойного щелчка мышью на имени процедуры обработки, которое находится на странице *Events* окна *Object Inspector*.

Например, в пункт Просмотр введем следующий код

```
Form1.Visible := False;  
Form3.Visible := True;
```

элемент меню Поступление материала

```
Form1.Visible := False;  
Form2.Visible := True;
```

элемент меню Выход: *Close*.

Выберем элемент меню *File|Save Project As* и сохраним проект. Сначала запрашивается имя формы проекта (у нас пять форм, с именами *Form1 ... Form4, DataModule5*). Сохраним формы под именами 'Unit*.pas'. Затем запрашивается имя проекта. Сохраним проект под именем 'appl.dpr'.

Добавим имена модулей Unit2 и Unit3 в текст модулей unit формы № 1.

После этого выполним приложение. Чтобы выполнить приложение, не выходя из среды Delphi, достаточно нажать кнопку F9. Чтобы создать приложение и запустить его вне среды Delphi, следует нажать комбинацию кнопок Ctrl+F9 и затем запустить созданный файл с расширением .exe и именем, совпадающим с именем проекта. В нашем случае следует запускать файл 'appl.exe'.

Остановить работающее приложение можно командой

Run|Program Reset.

Создание экранной формы для представления содержания БД

1. Представление содержания одной таблицы БД на экранной форме

Сначала построим простую экранную форму, которая имеет доступ только к одной таблице БД.

Перейдем в форму № 3 путем выбора элемента (или команды) меню *View|Forms* нажатием кнопки *Ok*, выбираем из диалогового окна *Form3*. Расположим в форме компонент *TDBGrid*, взяв его из палитры компонентов (страница *Data Controls*). Установим свойство *DataSource* компонента *TDBGrid* в значение *DataSource1* (это имя созданного нами выше компонента *TDataSource*, присвоенное Delphi по умолчанию) предварительно добавив имя модуля *Unit5*, в котором описан *TDataModule*, в текст модулей *unit* формы № 3. Компонент *TDBGrid* служит для отображения записей набора данных в табличной форме.

Добавим в форму компонент кнопки *TButton* (страница *Standard* палитры компонентов), дадим имя этому компоненту (свойство *Name*), используя инспектор объектов *CancelButton*. Изменим заголовок кнопки (свойство *Caption*) на *Отмена*.

Вид разрабатываемой формы представлен на рис. 4.2.

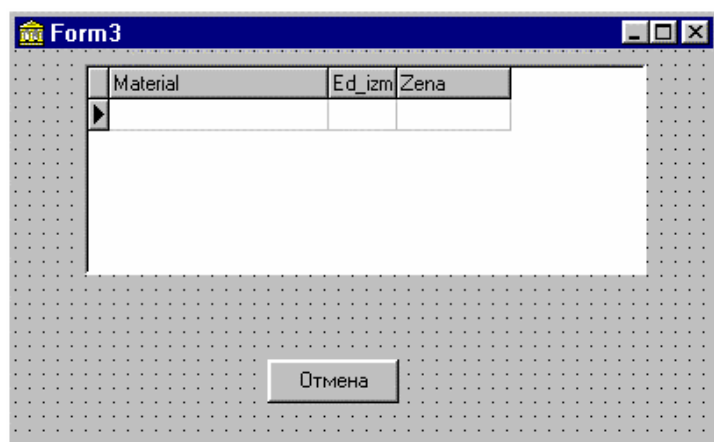


Рис. 4.2. Вид формы на этапе разработки

Свяжем с кнопкой код, который будет выполняться при нажатии на кнопку пользователем. Для создания процедуры обработки этого события – *OnClick*, находясь в форме, дважды щелчком мышь на кнопку Отмена. В появившемся окне Редактора кода процедуре обработки события *OnClick* напишем код

```
Close;  
Form1.Visible := True.
```

Запустим приложение на выполнение из Delphi, выбрав команду *Run* из меню *Run* (или F9). Из меню *Файл/Ввод* приложения выберем команду *Просмотр*.

Вид работающего приложения показан на рис. 4.3.

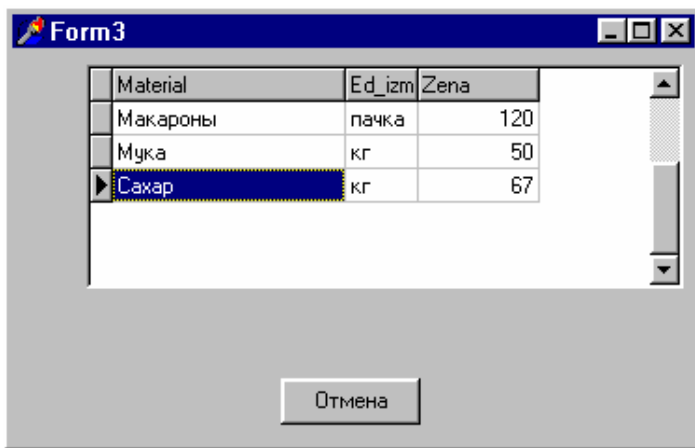


Рис. 4.3. Вид формы № 3 во время выполнения

Добавлять записи в набор данных (и, следовательно, в таблицу *Materialy.DB*) можно прямо из компонента *TDBGrid*.

Для добавления записи нужно нажать на клавиатуре кнопку *Insert* или, находясь на последней записи набора данных, кнопку "стрелка вниз". Набор данных автоматически перейдет в режим добавления новой записи. После ввода значений в поля записи, запомнить запись в наборе данных можно, перейдя на другую запись при помощи клавиш управления курсором. Отказаться от запоминания записи можно, на-

жав кнопку *Esc*.

Для изменения записи следует переместить указатель текущей записи в нужное место и изменить значения там, где это необходимо. Набор данных автоматически перейдет в режим редактирования.

Для удаления записи следует установить на нее указатель текущей записи и нажать комбинацию кнопок *Ctrl+Del*.

2. Представление содержания двух таблиц БД

Теперь работу усложним. Представим содержание двух таблиц БД. Причем в двух вариантах:

- а) без установления связи между таблицами на экранной форме;
- б) отобразим данные таблицы с учетом связи между таблицами.

2.1. Отображение данных таблиц БД на экранной форме без установления связи между ними

Покажем, как в одной форме можно связать два набора данных (родительский и подчиненный) так, чтобы в подчиненном наборе данных всегда показывались записи, соответствующие текущей записи в родительском НД.

Добавим в контейнер *DataModule5* компонент *TTable* (с именем *Table2*) для работы с таблицей *Prihod* базы данных *PRIMER* (значения свойств такие же, как у компонента *Table1*, но свойство *TableName* ссылается на имя таблицы *Prihod.DB*). Добавим в форму компонент *TDataSource* (имя по умолчанию *DataSource2*). Установим свойство *DataSet* этого компонента в значение *Table2*. Разместим в форме № 3 компонент *TDBGrid* (имя по умолчанию *DBGrid2*) и установим его свойство *DataSowce* в значение *DataSource2* (рис. 4.5). Установим свойство *Tabl2.Active* в *True*. После этого произойдет реальное связывание компонентов *TTable* с реально существующими таблицами *Materialy.DB* и *Prihod.DB*.

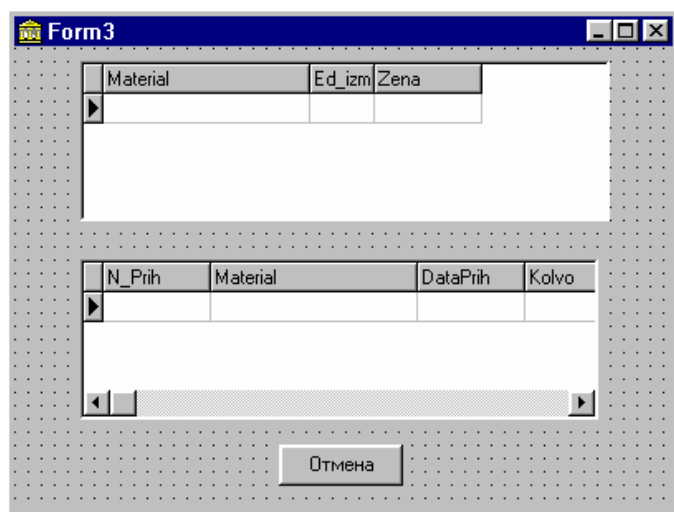


Рис. 4.5. Форма с родительским и дочерними наборами данных во время разработки

Чтобы при просмотре нельзя было изменять данные о материалах (таблица *Materialy.DB*), установим свойства *ReadOnly* компонента *DBGrid1* в положение *True*.

Запустим приложение на выполнение. Добавим несколько записей в таблицу *Prihod.DB* из компонента *TDBGrid2*. Обратите внимание, значение поля *N_Prih* формируется автоматически (рис. 4.6).

2.2. Отображение данных таблиц БД на экранной форме с учетом связи между ними

Нам известно, что таблицы базы данных *Materialy.DB* и *Prihod.DB* находятся в отношении "один – ко многим". Поскольку мы определили ссылочную целостность между этими таблицами, можно сделать так, чтобы при установке указателя на запись в наборе данных *Table1* (ассоциированном с *Materialy.DB*) в наборе данных *Table2* (ассоциированном с *Prihod.DB*) показывались только записи прихода текущего материала в *Table1*. Это реализуется через механизм связи наборов данных *Master-Detail*.

Material	Ed_izm	Zena
Макароны	пачка	120
Мука	кг	50
Сахар	кг	67

N_Prih	Material	DataPrih	Kolvo
1	Макароны	01.01.01	
2	Сахар	02.01.01	
3	Мука	12.11.00	

Отмена

Рис. 4.6. В подчиненном наборе данных показываются записи, связанные с текущей записью в родительском НД

В инспекторе объектов для компонента Table2 установим значение свойства *Master Source* в DataSource1. Переместимся на значение свойства *Master Fields* и нажмем кнопку . В появившемся окне *Field Link Designer* установим параметры связи. В поле *Available Indexes* выберем в качестве текущего индекса по полю 'Material'. В списке *Detail Fields* выберем поле Material, в списке *Master Fields* выберем поле Material и нажмем кнопку *Add*. В поле *Joined Fields* будет сформировано выражение 'Material → Material' (рис. 4.7). Нажмем кнопку *Ok*.

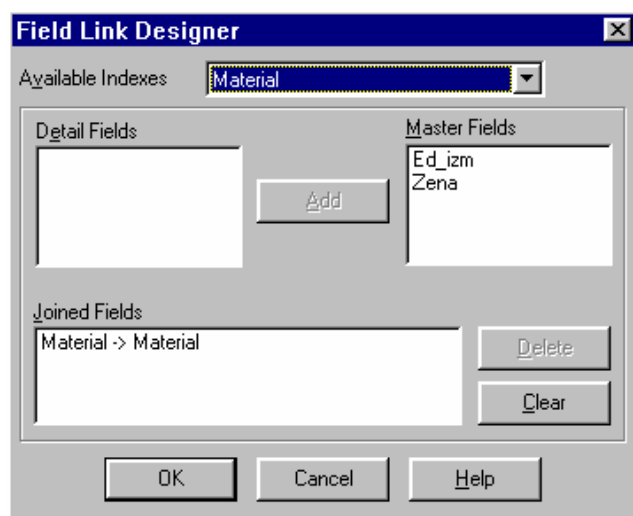


Рис. 4.7. Окно установки значения свойства MasterFields

Как можно заметить, в компоненте Table2 текущий индекс (свойство *FieldIndexNames*) заменен индекс, построенный по полю 'Material'.

Теперь в наборе данных Table2 показываются только записи по приходу материала, текущего в наборе данных Table1 (рис. 4.8).

The screenshot shows a Delphi form titled "Form3" with a dotted grid background. It contains two tables and a button. The top table has three columns: "Material", "Ed_izm", and "Zena". It contains three rows: "Макароны" (120), "Мука" (50), and "Сахар" (67). The bottom table has four columns: "Material", "N_Prih", "DataPrih", and "Kolvo". It contains one row: "Макароны" (3, 07.09.00, 200). A button labeled "Отмена" is located at the bottom center of the form.

Material	Ed_izm	Zena
Макароны	пачка	120
Мука	кг	50
Сахар	кг	67

Material	N_Prih	DataPrih	Kolvo
Макароны	3	07.09.00	200

Отмена

Рис. 4.8. Связь Master-Detail – в дочернем наборе данных показываются только подчиненные записи

Создание экранной формы для ввода данных в базе данных

Для дальнейшей работы подготовим контейнер *DataModule5*. Из палитры компонентов Delphi на странице *Data Access* добавим невидимый компонент *TTable*, установим значения свойств *DatabaseName* (псевдоним БД) в *PRIMER* при помощи выпадающего списка или введя его вручную. В *Table3* установим значение свойства *TableName* (имя таблицы БД) в *PRIHOD.DB*. После этого установим значение свойства *Active* в *True*. Расположим компонент *TDataSource*, установим свойство *DataSet* (имя набора данных) компонента *DataSource3* в значение *Table3*. Добавление невидимых компонентов для связи с таблицей *PRIHOD.DB* в контейнер обязательно, т. к. аналогичные компоненты в контейнере *DataModule* взаимно связаны и в дальнейшем может произойти циклическое обращение.

Добавим в форму № 2 компонент *TDBGrid* из палитры компонентов *Data Controls*. Установим свойство *DataSource* компонента *TDBGrid* в значение *DataSource3*, предварительно добавив имя модуля *Unit5* в текст модулей *unit* формы № 2.

Теперь сделаем так, чтобы к полям записи в наборе данных

Table3 можно было обращаться не только из сетки компонента DBGrid1, а из отдельных визуальных компонентов, позволяющих осуществлять доступ к отдельным полям записи набора данных.

Добавим в форму № 2 два компонента TDBEdit (палитра компонентов *Data Controls*). Определим поле, к которому можно иметь доступ через компонент DBEdit1. Для этого установим значение его свойств - *DataSource* в DataSource3 и *DataField* в DatPrih. Определим поле, к которому можно иметь доступ через компонент DBEdit2. Для этого установим значение его свойств *DataSource* в DataSource3 и *DataField* в Kolvo.

Для доступа к полю Material нам нужен более сложный компонент, который позволял бы вводить в поле Material таблицы Prihod.DB значения полей Material из таблицы Materialy.DB, и никакие другие значения. Для этой цели поместим компонент TDBLookupComboBox с именем по умолчанию DBLookupComboBox1. Установим свойства этого компонента

- *DataSource* – в значение DataModule5.DataSource3;
- *DataField* – в значение Material;
- *List Source* – в значение DataModule5.DataSource1
- *ListField* – в значение Material;
- *KeyField* – в значение Material.

Добавление кнопок управления данными базы данных на экранной форме

Далее на экранной форме можно создать набор кнопок, которые позволят проводить навигацию по таблицам БД и управлять ее состоянием, т. е. ее записями.

Для этого в приложение добавим пять компонентов кнопок TButton (страница *Standard* палитры компонентов). Изменим имена этих компонентов (свойство *Name*), используя *инспектор объектов*, соответственно на InsertButton, EditButton, DeleteButton, PostButton, CancelButton. Изменим заголовки этих кнопок (свойство *Caption*), используя *инспектор объектов*, соответственно на "Добавить", "Изменить", "Удалить", "Запомнить", "Закрыть" (рис. 4.9).

Выберем при помощи мыши кнопку InsertButton и два раза щелкнем на ней. После этого мы перейдем в редактор кода и определим для кнопки InsertButton обработчик события нажатия кнопки, *OnClick*

```
procedure TForm2.InsertButtonClick(Sender: TObject);  
begin
```

```
DataModule5.Table3.Insert;
End.
```

Метод *Insert* переводит набор данных Table1 в состояние добавления записи *dsInsert*. Ввод значений полей осуществляется в компонентах DBEdit1, DBLookupComboBox1, DBEdit2. Для этого необходимо, чтобы набор данных находился в режиме просмотра *dsBrowse*.

Определим обработчик нажатия кнопки EditButton

```
procedure TForm2.EditButtonClick(Sender: TObject);
begin
DataModule5.Table3.Edit;
end.
```

N_Prih	Material	DataPrih	Kolvo
1	Мука	12.01.01	12300
2	Мука	20.05.00	1000
3	Макароны	07.09.00	200
4	Сахар	09.07.00	3500

Сахар 09.07.00 3500

Добавить Изменить Удалить Запомнить Закреть

Рис. 4.9. Форма с визуальными компонентами для работы с полями текущей записи набора данных

Метод *Edit* переводит набор данных Table1 в состояние добавления записи *dsEdit*. Редактирование значений полей осуществляется в компонентах DBEdit1, DBLookupComboBox1, DBEdit2. Для этого необходимо, чтобы набор данных находился в режиме просмотра *dsBrowse*.

Определим обработчик нажатия кнопки DeleteButton

```
procedure TForm2.DeleteButtonClick(Sender: TObject);  
begin  
  IF MessageDlg('Подтвердите удаление записи',  
    mtConfirmation,[mbYes, mbNo],0) = mrYes THEN  
    DataModule5.Table3.Delete;  
end.
```

Если набор данных Table1 находится в режиме просмотра записей *dsBrowse*, вызывается окно диалога (при выполнении функции *MessageDlg*); если пользователь нажимает кнопку Yes, происходит удаление текущей записи в наборе данных Table1.

```
Определим обработчик нажатия кнопки PostButton  
procedure TForm2.PostButtonClick(Sender: TObject);  
begin  
  DataModule5.Table3.Post;  
end.
```

Если набор данных находится в режиме добавления новой записи или редактирования, происходит выполнение метода набора данных *Post*, который запоминает текущее состояние записи в таблице БД. После запоминания набор данных переводится в режим просмотра *dsBrowse*.

Определим обработчик нажатия кнопки CancelButton

```
procedure TForm2.CancelButtonClick(Sender: TObject);  
begin  
  close;  
  Form1.Visible:=true;  
end.
```

Если набор данных находится в режиме добавления новой записи или редактирования, происходит выполнение метода набора данных *Cancel*, который отменяет запоминание записи в таблице БД и переводит набор данных в режим просмотра и переходит на форму меню.

Для того чтобы набор данных нельзя было переводить в состояние добавления и изменения данных, а также удалять записи непосредственно из компонента DBGrid1, установим свойство *DBGrid1.ReadOnly* в значение True. После этого запустим приложение на выполнение. При добавлении новой записи или при корректировке существующей, в поля можно вносить значения, используя ввод в компоненты DBEdit1, DBEdit2 и путем выбора из списка значений в компо-

ненте DBLookupComboBox1 (рис. 4.10). То же происходит при изменении записи.



Рис. 4.10. Добавление новой записи

При удалении записи выдается окно диалога (рис. 4.11).

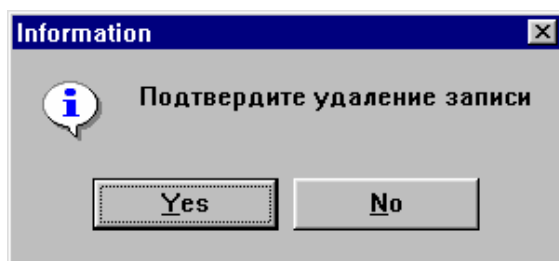


Рис. 4.11. Окно подтверждения удаления записи

4.3.3. Создание сложных экранных форм

В данном пункте описываются вопросы создания более сложных вариантов экранных форм и установление их доступа к БД, а также организация процессов обработки данных.

Варианты модификации TDBGrid на экранной форме

Как можно заметить, значение поля N_Prih необходимо для обеспечения уникальности в таблице Prihod и не несет никакой иной нагрузки. Поэтому данное поле лучше не показывать в составе столбцов DBGrid2. Для этой цели сформируем список полей таблицы Prihod. В Delphi имеются две возможности указать, какие из полей таблицы БД следует использовать в приложении для набора данных.

Первый способ состоит в использовании по умолчанию *всех* полей из таблицы БД, с которой ассоциирован этот набор данных. Этот способ всегда используется по умолчанию и, следовательно, был неявно использован и нами при создании наборов данных Table1 и Table2.

Второй способ состоит в использовании *подмножества* полей таблицы БД, с которой ассоциирован набор данных. Для этой цели используется *редактор полей* набора данных, который позволяет включить в состав обрабатываемых для набора данных полей все поля или подмножество полей таблицы БД.

Выберем в контейнере DataModule5 при помощи мыши компонент Table2 и нажмем правую кнопку мыши. В появившемся на экране всплывающем меню выберем элемент *Fields Editor*. В появившемся списке редактора полей (пока он пуст, рис. 4.12а) нажмем правую кнопку мыши и во всплывающем меню выберем элемент меню *Add Fields*. Будет показан список всех полей таблицы БД Prihod.DB. Отметим (при помощи мыши и кнопки Shift) все поля, кроме N_Prih (рис. 4.12 б) и нажмем кнопку *Add*. Теперь список редактора полей будет включать все отмеченные поля (рис. 4.12 в).

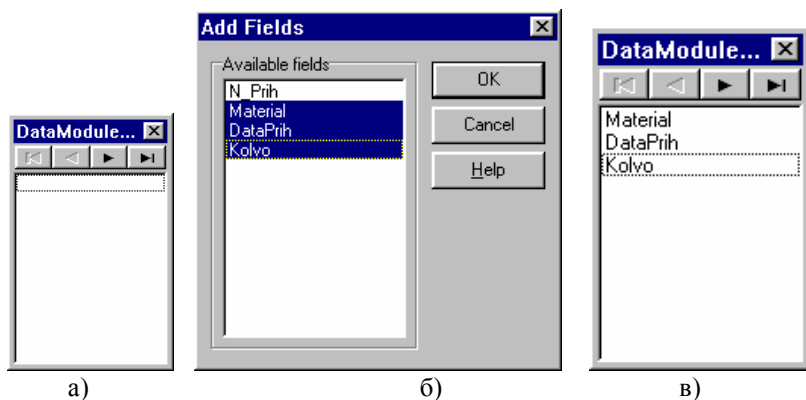


Рис. 4.12. Работа с редактором полей: а) пустой список редактора полей;
б) добавление полей; в) заполненный список редактора полей

Как можно заметить, в составе столбцов в компоненте DBGrid2 на форме № 3 теперь присутствуют только те поля, которые добавлены для набора данных Table2 в редакторе полей (рис. 4.13).

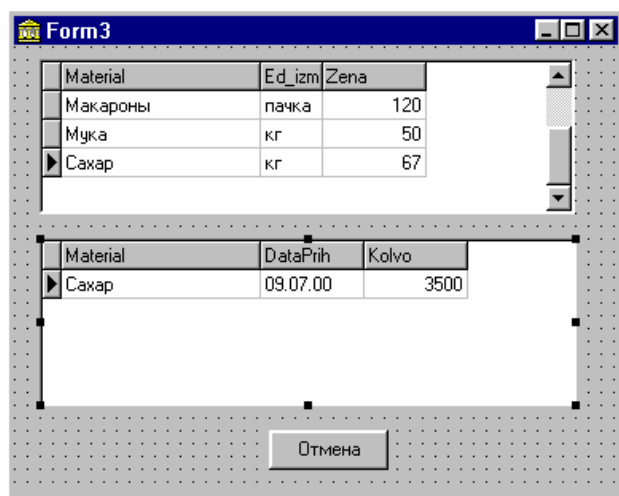
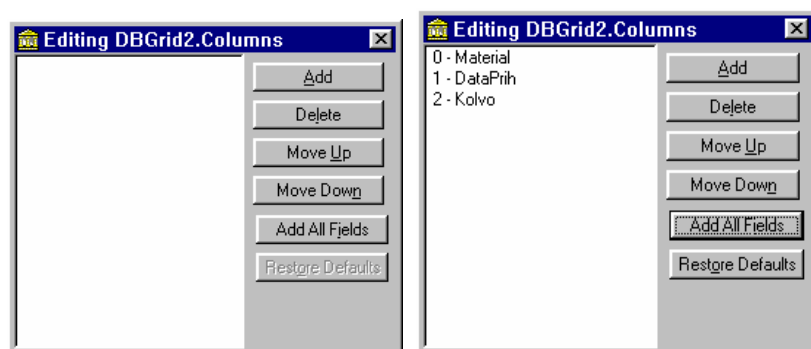


Рис. 4.13. Состав полей DBGrid2 определяется списком редактора полей

Определение для набора данных списка полей в редакторе полей (нажатие правой кнопки мыши по компоненту Table2) приводит к тому, что для каждого добавленного таким образом поля в приложении Delphi автоматически создается компонент TField (поле набора данных). Каждый такой компонент по умолчанию именуется уникальным именем – в качестве первой составляющей имени поля берется имя набора данных (Table2), а в качестве второй составляющей – имя поля в таблице БД. Так, компонент TField соответствующий полю Material, будет называться как Table2Material. Если в редакторе полей щелкнуть по имени соответствующего поля, в инспекторе объектов можно установить или изменить свойства поля, а также определить обработчики события для конкретного поля.



а)

б)

Рис. 4.14. Характеристики столбцов DBGrid2:

а) пустой список столбцов DBGrid2;

б) заполненный список столбцов.

Изменим параметры компонента DBGrid2 так, чтобы названия его столбцов содержали русские наименования. Для этого нажмем правой кнопкой мыши на компоненте DBGrid2, и во всплывающем меню выделим элемент *Columns Editor*. На экране появится окно редактора столбцов компонента (рис. 4.14 а). Для того чтобы изменить характеристики столбцов в TDBGrid, нужно перейти от неявно определяемых столбцов к определяемым. Для этого нужно щелкнуть по кнопке *Add All Fields*, в результате чего будут добавлены столбцы, каждый из которых соответствует полю, определенному в редакторе полей компонента Table2 (рис. 4.14 б). Что бы изменить заголовок каждого столбца, следует выбрать при помощи мыши имя столбца в редакторе столбцов, и в инспекторе объектов раскрыть список свойства *Title* (для чего следует щелкнуть мышью по крестику рядом с именем свойства). В элементе *Caption* этого списка содержится заголовок столбца; изменим соответствующим образом заголовки и затем выйдем из редактора столбцов DBGrid2. То же сделаем для набора данных Table1 (рис. 4.15).

The screenshot shows a Windows application window titled "Form3". It contains two data grids. The top grid has three columns: "Материал", "Ед.изм.", and "Цена". It contains three rows of data: "Макароны" (пачка, 120), "Мука" (кг, 50), and "Сахар" (кг, 67). The bottom grid has three columns: "Материал", "Дата прихода", and "Кол-во". It contains one row of data: "Сахар" (09.07.00, 3500). Both grids have a vertical scrollbar on the right. An "Отмена" button is located at the bottom center of the form.

Материал	Ед.изм.	Цена
Макароны	пачка	120
Мука	кг	50
Сахар	кг	67

Материал	Дата прихода	Кол-во
Сахар	09.07.00	3500

Отмена

Рис. 4.15. Установка заголовков столбцов DBGrid1 и DBGrid2

Аналогичные изменения проделаем и в форме № 2.

Изменим также порядок сортировки записей в наборе данных Table3 в форме № 2. Для этого в *инспекторе объектов* компонента Table3 (в DataModule5) установим свойство *IndexFieldNames* в значение "DatPrih;Material" путем выбора из выпадающего списка, содержащего названия индексных полей, определенных для каждого существующего индекса таблицы Prihod.DB. После этого войдем еще раз в редактор колонок DBGrid и при помощи мыши "перетащим" столбец DatPrih так, чтобы он предшествовал столбцу Material. Откомпилируем приложение и запустим его на выполнение. Как видно из рис. 4.16, набор данных Table2, ассоциированный с таблицей БД Prihod.DB, в приложении отсортирован по дате прихода, а внутри каждой даты прихода – по наименованию материала.

Дата прихода	Материал	Кол-во
20.05.00	Мука	1000
09.07.00	Сахар	3500
07.09.00	Макароны	200
12.01.01	Мука	12300
14.01.01	Макароны	560

20.05.00	Мука	1000
----------	------	------

Добавить	Изменить	Удалить	Запомнить	Заккрыть
----------	----------	---------	-----------	----------

Рис. 4.16. Материалы отсортированы по дате поступления

Создание вычисляемых полей на экранной форме

Создание вычисляемых полей – одно из наиболее ценных свойств *редактора полей*. Это свойство можно использовать для различных целей, в нашем примере рассмотрим случай вычисления по двум или более полям с отображением результата вычислений в третьем поле.

В экранной форме № 3 создадим новое вычисляемое поле в DBGrid2, значение которого вычисляется по значениям других полей, для этого выполним следующие действия:

1. Перейдем в DataModule5, при помощи мыши сделаем текущим необходимый набор данных (в нашем примере Table2), нажав на правую кнопку мыши, выберем в меню пункт *Field Editor (редактор полей)* и снова, нажав на правую кнопку мыши, выберем в меню пункт *New Field*.

2. Затем в диалоговом окне (рис. 4.17) необходимо указать имя поля (в поле Name введем “vychisl”), его тип (в поле Type выберем “String”) и длину (в поле Size введем “3”). Помечаем поле как вычисляемое (Calculated). Нажимаем Ok и закрываем *редактор полей*.

Таким образом, для нового поля “vychisl” создан компонент TField, доступ к которому можно осуществлять в *редакторе полей*.

Процесс, описанный выше, показывает как создать вычисляемое поле. Если посмотреть на DBGrid2, то можно видеть, что там теперь

еще одно пустое поле. Для того, чтобы поместить значение в это поле необходимо выполнить следующий шаг.

3. Для компонента набора данных Table2, к которому принадлежит вычисляемое поле, необходимо определить обработчик события OnCalcFields.

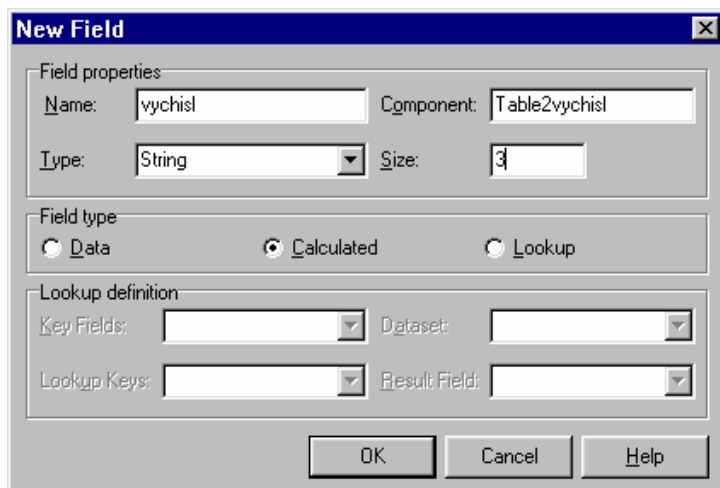


Рис. 4.17. Окно определения свойств нового поля

Например, нам необходимо знать наличие на складе товаров более 100 единиц. Для этого будем заносить в созданное вычисляемое поле Table2Vychisl набора данных Table2, ассоциированному с таблицей БД “Приход”, значение “Да”, если в поле Table2Kolvo этой записи содержится значение больше 100. В противном случае в поле Table2Vychisl будем заносить пустое значение.

В окне Редактора кода процедуры обработки события OnCalcFields напомним следующий код:

```
procedure TDataModule5.Table2CalcFields(DataSet:
TDataSet);
begin
  IF Table2Kolvo.Value > 100 Then
    Table2Vychisl.AsString := 'Да'
  ELSE
```

```
Table2Vychisl.AsString := '';
end.
```

Если свойство набора данных AutoCalcFields установлено в True, событие OnCalcFields наступает и при модификации значений невычисляемых полей в режимах dsInsert и dsEdit данного набора данных или набора данных, реляционно с ним связанного (когда установлены ограничения целостности в самой таблице БД, а не тогда, когда они подразумеваются).

Процедура-обработчик события OnCalcFields содержит реализацию алгоритма вычисления значения вычисляемого поля или группы полей. Необходимо помнить, что в этом обработчике значение может быть присвоено только вычисляемому полю и не может – полю, определенному в структуре БД (приложение показано на рис. 4.18).

The screenshot shows a Delphi application window titled "Form3". It contains two data tables. The top table has columns: "Материал", "Ед.изм.", and "Цена". The bottom table has columns: "Материал", "Дата прихода", "Кол-во", and "Наличие". A button labeled "Отмена" is at the bottom.

Материал	Ед.изм.	Цена
Макароны	пачка	120
Мука	кг	50
Сахар	кг	67

Материал	Дата прихода	Кол-во	Наличие
Мука	12.01.01	12300	Да
Мука	20.05.00	1000	Да

Отмена

Рис. 4.18. Вид запущенного приложения

Использования SQL-запроса для отображения данных из разных таблиц на экранной форме

Создадим форму для отображения данных из нескольких таблиц. Для этого в контейнере TDataModule5 расположим компонент TQuery (страница *Data Access* палитра компонентов). По умолчанию его имя

Query1. Установим его свойство *DatabaseName* в PRIMER. Расположим компонент TDataSource (имя DataSource4). Установим его свойство *DataSet* в значение Query1.

Расположим в форме № 4 компонент TDBGrid. Установим его свойство *DataSource* в значение DataSource4.

В инспекторе объектов для компонента Query1 найдем свойство *SQL* и нажмем кнопку . Затем в появившемся окне редактора наберем текст SQL-запроса

```
SELECT P.DatPrih, P.Material, P.Kolvo, T.Zena,
(P.Kolvo * T.Zena) As Stoim
FROM Materialy T, Prihod P
WHERE T.Material = P.Material
ORDER BY P.DatPrih, P.Material
```

и нажмем кнопку *Ok*.

После этого установим свойство *Query1.Active* в значение True. Набор данных Query1 содержит сведения о приходе материала на склад. В составе записи этого набора присутствуют поля DatPrih (дата прихода), Material (название материала), Kolvo (количество прихода), Zena (цена за ед. измерения данного материала), Stoim (стоимость прихода материала). Как видно из текста запроса в свойстве *SQL*, набор данных "собирается" из двух таблиц БД PRIMER, Materialy.DB и Prihod.DB. При этом соединяются записи из этих таблиц БД, имеющие одинаковое значение поля Material (рис. 4.19).



Дата прихода	Материал	Кол-во	Цена	Сумма
20.05.00	Мука	1000	50	50000
09.07.00	Сахар	3500	67	234500
07.09.00	Макароны	200	120	24000
12.01.01	Мука	12300	50	615000
14.01.01	Макароны	560	120	67200

Отмена

Рис. 4.19. Соединение данных из разных таблиц в одном наборе данных

В форме № 1 в пункт Отчет введем следующий код:

```
Form1.Visible := False;  
Form4.Visible := True.
```

Заметим, что в набор данных Query1 нельзя добавить новые записи и нельзя изменить или удалить существующие записи в наборе данных. Это происходит потому, что тип набора данных, "собираемого" более чем из одной таблицы БД, является доступным только для чтения.

4.3.4. Создание отчетных форм

В данном пункте описываются вопросы создания отчетных форм и установление их доступа к БД и организация обработки данных.

Создание отчетных форм можно представить как всякую форму и рассмотреть в данной главе вместе с экранными формами. Однако формирования отчета имеет свои особенности. Кроме того, процесс формирования отчетов можно рассмотреть отдельно от вопросов СУБД. Формирование отчетных форм более или менее отдельный вопрос из-за того, что отчет включает три вида компонентов. Это экранная форма отчета в виде текста и таблицы, формы отчетов выводимые на бумажные носители, графические материалы и материалы, формируемые в виде электронных файлов.

Поэтому все эти вопросы вынесем в отдельную главу и в ней рассмотрим вопросы формирования отчетов различного вида.

4.3.5. Запуск и проверка работы системы

Разрабатываемую систему время от времени следует запускать на исполнение и проверять ее работоспособность. Причем проверку работоспособности системы можно проводить на любой стадии создания, необязательно полностью завершать ее создание, достаточно получить минимальную конфигурацию. Минимальную конфигурацию составляет БД и одна экранная форма, которая обращается к БД.

Для запуска системы предварительно следует запомнить все соз-

данные компоненты.

Запуск системы осуществляется по команде: *Run|Run*. Запуск системы можно осуществить и с помощью клавиши F9.

В случае зависания системы или при отсутствии выхода из системы прерывание работы осуществляется по команде *Run|Program Reset*.

После прерывания работы системы можно продолжать ее разработку или отладку.

Работа полного варианта системы, без отчетной части системы, начинается с отображения на экране формы меню. Далее пользователь системы может с помощью меню переходить на любую другую экранную форму и вернуться на главное меню системы.

Если порядок отображения экранных форм не соответствует желаемому варианту, то его можно менять в поле *Main Form* с помощью элемента меню *Proect|Options* закладка *Forms*.

4.4. Вычисляемые поля

Для того, чтобы дать более полное представление о создании вычисляемых полей и об организации процессов вычисления в данном пункте подробно раскроем эти вопросы.

Создание вычисляемых полей – одно из наиболее ценных свойств Редактора DataSet. Вы можете использовать эти поля для различных целей, но два случая выделяются особо:

- выполнение вычислений по двум или более полям в DataSet, и отображение результата вычислений в третьем поле;
- имитация соединения двух таблиц с возможностью редактировать результат соединения.

Программа CALC_SUM.DPR из примеров к данному уроку иллюстрирует первый случай использования вычисляемых полей.

Эта программа связывает три таблицы в отношении один ко многим. В частности, ORDERS и ITEMS связаны по полю OrderNo, а ITEMS и PARTS связаны по полю PartNo. (В таблице ORDERS хранятся все заказы; в таблице ITEMS – предметы, указанные в заказах; PARTS – справочник предметов). В программе можно перемещаться по таблице ORDERS и видеть связанный с текущим заказом список включенных в него предметов. Программа CALC_SUM достаточно сложная, но хорошо иллюстрирует мощность вычисляемых полей.

Последовательность создания проекта CALC_SUM:

1. Создайте новый проект (File|New Project) и удалите из него форму (в Менеджере Проекта View|Project Manager).
2. Выберите эксперта форм БД из меню Help.
3. На первом экране выберите "Create a master/detail form" и "Create a form using TQuery Objects".
4. Нажмите кнопку Next и выберите таблицу ORDERS.DB из псевдонима БД DBDEMOS.
5. Нажмите Next и выберите поля OrderNo, CustNo, SaleDate, ShipDate и ItemsTotal из таблицы ORDERS.DB.
6. Нажмите Next и выберите "Horizontal" из расстановки компонентов dbEdit на форме.
7. Нажмите Next и выберите таблицу ITEMS.DB.
8. В двух следующих экранах выберите все поля из таблицы и поместите их в grid.
9. Нажмите Next и выберите поле OrderNo из Master и Detail ListBoxes, и нажмите кнопку Add.
10. Нажмите Next и сгенерируйте форму.

Требуется много слов для того, чтобы описать процесс, показанный выше, но, фактически, выполнение команд в Эксперте форм БД легко и интуитивно.

Выделите первый из двух объектов TQuery и установят свойство Active в True. Для Query2 в свойстве SQL напишите текст запроса:

```
select * from Items I, Parts P
where (I.OrderNo =:OrderNo) and
(I.PartNo=P.PartNo).
```

Активизируйте объект Query2 (Active установите в True) и вызовите редактор DataSet (Fields Editor) для него. Вызовите диалог Add Fields и добавьте поля OrderNo, PartNo, Qty и ListPrice.

Нажмите Define и введите слово Total в поле FieldName. Установите Field Type в CurrencyField. Проверьте Calculated CheckBox отмечен ли. Нажмите Ok и закройте редактор DataSet.

Простой процесс, описанный в предыдущем абзаце, показывает как создать вычисляемое поле. Если посмотреть в DBGrid, то можно видеть, что там теперь есть еще одно пустое поле. Для того, чтобы поместить значение в это поле, откройте в Инспекторе Объектов страни-

цу событий для объекта Query2 и сделайте двойной щелчок на OnCalcFields. Заполните созданный метод так

```
procedure TForm2.Query2CalcFields(DataSet: TDataSet);  
begin  
    Query2NewTotalInvoice.Value := 23.0;  
end.
```

После запуска программы, поле Total будет содержать строку \$ 23.00.

Это показывает насколько просто создать вычисляемое поле, которое показывает правильно сформатированные данные. На самом деле это поле должно показывать нечто другое – произведение полей Qty (количество) и ListPrice (цена). Для этого вышеприведенный код для события OnCalcFields нужно изменить следующим образом:

```
procedure TForm1.Query2CalcFields(DataSet: TDataset);  
begin  
    Query2Total.Value:=Query2Qty.Value*Query2ListPrice.Value;  
end.
```

Если теперь запустить программу, то поле Total будет содержать требуемое значение.

В обработчике события OnCalcFields можно выполнять и более сложные вычисления (это будет показано позже), однако следует помнить, что это вызывает соответствующее замедление скорости работы программы.

Теперь давайте добавим вычисляемое поле для первой таблицы (Query1, ORDERS), которое будет отображать сумму значений из поля Total второй таблицы (Query2) для данного заказа. Вызовите редактор DataSet для объекта Query1 и добавьте вычисляемое поле NewItemsTotal типа CurrencyField. В обработчике события OnCalcFields для Query1 нужно подсчитать сумму и присвоить ее полю NewItemsTotal

```
procedure TForm1.Query1CalcFields(DataSet: TDataset);  
var R : Double;  
begin  
    R:=0;  
    with Query2 do begin
```

```
DisableControls;  
Close;  
Open;  
repeat  
R:=R+Query2Total.Value;  
Next;  
until EOF;  
First;  
EnableControls;  
end;  
Query1NewItemTotal.Value:=R;  
end.
```

В данном примере сумма подсчитывается с помощью простого перебора записей, это не самый оптимальный вариант - можно, например, для подсчета суммы использовать дополнительный объект типа TQuery. Метод DisableControls вызывается для того, чтобы отменить перерисовку DBGrid при сканировании таблицы. Запрос Query2 перепроверяется для уверенности в том, что его текущий набор записей соответствует текущему заказу.

Поместите на форму еще один элемент DBEdit и привяжите его к Query1, полю NewItemsTotal. Запустите программу, ее примерный вид показан на рис. 4.20.

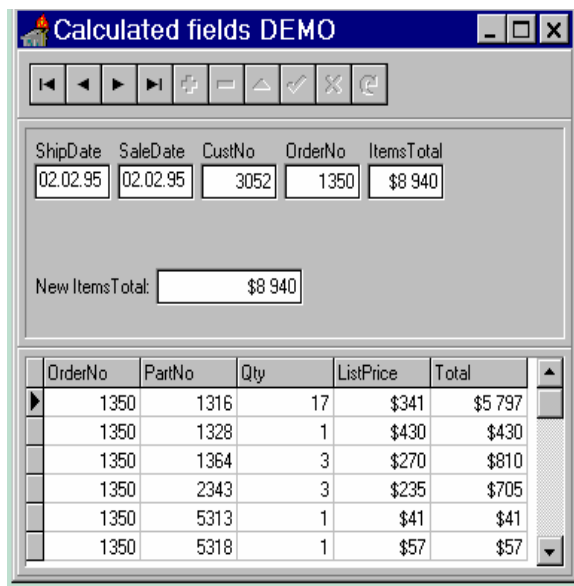


Рис. 4.20. Программа CALC_SUM

Как видно из программы, наличие поля ItemsTotal в таблице ORDERS для данного примера необязательно и его можно было бы удалить (однако, оно необходимо в других случаях).

Управление TDBGrid во время выполнения

Объект DBGrid может быть полностью реконфигурирован во время выполнения программы. Вы можете прятать и показывать колонки, изменять порядок показа колонок и их ширину.

Вы можете использовать свойство Options объекта DBGrid, чтобы изменить ее представление. Свойство Options может принимать следующие возможные значения:

DgEditing установлен по умолчанию в true, позволяет пользователю редактировать grid. Вы можете также установить свойство ReadOnly grid в True или False.

DgTitles – Будут ли видны названия колонок?

DgIndicator – Будут ли видны небольшие иконки слева?

DgColumnResize – Может ли пользователь менять размер колонки?

dgColLines – Показывать ли линии между колонками?

dgRowLines – Показывать ли линии между строками?

dgTabs – Может ли пользователь использовать tab и shift-tab для переключения между колонками?

Как объявлено в этой структуре

```
TDBGridOption = (dgEditing, gdAlwaysShowEditor, dgTitles,  
dgIndicator, dgColumnResize, dgColLines,  
dgRowLines, dgTabs).
```

Например, вы можете установить опции в Runtime, написав такой код

```
DBGrid1.Options := [dgTitles, dgIndicator].
```

Если вы хотите включать и выключать опции, это можно сделать с помощью логических операций. Например, следующий код будет добавлять dgTitles к текущему набору параметров

```
DBGrid1.Options := DBGrid1.Options + [dgTitles].
```

Пусть есть переменная ShowTitles типа Boolean, тогда следующий код позволяет включать и выключать параметр одной кнопкой

```
procedure TForm1.Button3Click(Sender: TObject).  
begin  
if ShowTitles then  
DBGrid1.Options := DBGrid1.Options + [dgTitles]  
else  
DBGrid1.Options := DBGrid1.Options - [dgTitles];  
ShowTitles := not ShowTitles;  
end.
```

Если вы хотите скрыть поле в run-time, то можете установить свойство visible в false

```
Query1.FieldByName('CustNo').Visible := False;  
Query1CustNo.Visible := False.
```

Обе строки кода выполняют идентичную задачу. Чтобы показать поле снова, установите видимый в true

```
Query1.FieldByName('CustNo').Visible := True;
```

```
Query1CustNo.Visible := True.
```

Если вы хотите изменить положение колонки в Runtime, можете просто изменить индекс (первое поле в записи имеет индекс ноль)

```
Query1.FieldName('CustNo').Index := 1;
```

```
Query1CustNo.Index := 2
```

По умолчанию, поле CustNo в таблице Customer является первым. Код в первой строке перемещает это поле во вторую позицию, а следующая строка перемещает его в третью позицию. Помните, что нумерация полей начинается с нуля, так присвоение свойству Index 1 делает поле вторым в записи. Первое поле имеет Index 0.

Когда Вы изменяете индекс поля, индексы других полей в записи изменяются автоматически.

Если Вы хотите изменить ширину колонки в Runtime, только измените свойство DisplayWidth соответствующего TField.

```
Query1.FieldName('CustNo').DisplayWidth := 12;
```

```
Query1CustNo.DisplayWidth := 12.
```

Величина 12 относится к числу символов, которые могут быть показаны в видимом элементе.

Программа DBGR_RT показывает как работать с DBGrid в Runtime. Программа достаточно проста, кроме двух небольших частей, которые описаны ниже. Первая часть показывает, как создать check box в Runtime, а вторая показывает, как изменить порядок пунктов в listbox в Runtime.

При создании формы (событие OnCreate) ListBox заполняется именами полей, далее создается массив объектов CheckBox, соответствующий полям в таблице. Сначала все CheckBox'ы выбраны и все поля в таблице видимы. Программа узнает через TTable1 имена полей и присваивает их свойству Caption соответствующего CheckBox. Кроме того, обработчику события OnClick всех CheckBox'ов присваивается процедура ChBClick, которая и включает/выключает поля в DBGrid.

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var i : Word;
```

```
R : Array[0..49] of TCheckBox;
```

```
begin
```

```
{Fill ListBox}
```

```
Listbox1.Clear;
```

```
for i:=0 to Table1.FieldCount-1 do
```

```
Listbox1.Items.Add(Table1.Fields[i].FieldName);
```

```
{Make CheckBoxes}
```

```
for i:=0 to Table1.FieldCount-1 do begin
```

```
R[I] := TCheckBox.Create(Self);
R[I].Parent := ScrollBox1;
R[I].Caption := Table1.Fields[i].FieldName;
R[I].Left := 10;
R[I].Top := I * CheckBox1.Height + 5;
R[I].Width := 200;
R[I].Checked := True;
R[I].OnClick := ChBClick;
end;
end.
```

Большая часть кода в этом примере выполняет относительно простые задачи, типа назначения имен и положений check boxes. Вот две ключевых строки:

```
R[I] := TCheckBox.Create(Self);
R[I].Parent := ScrollBox1.
```

Первая строка создает CheckBox с заданным Owner (Владельцем). Вторая строка назначает Parent (Родителя) для CheckBox. Чтобы понять различия между Родителем и Владелцем, посмотрите соответствующие свойства в online-help.

Программа содержит ListBox, который показывает текущий порядок полей в DataSet. Для изменения порядка полей в DataSet (а, следовательно, в DBGrid) используются две кнопки. При нажатии на одну из кнопок, выбранное в ListBox' поле перемещается на одну позицию вверх или вниз. Синхронно с этим меняется и порядок полей в DBGrid. Код, показанный ниже, изменяет Index поля для Table1, изменяя, таким образом, позицию поля в DBGrid. Эти изменения касаются только визуального представления DataSet. Физически данные на диске не изменяются.

```
procedure TForm1.downButtonClick(Sender: TObject);
var i : Integer;
begin
  with ListBox1 do
    if (ItemIndex<Items.Count-1)and(ItemIndex<>-1) then begin
      i := ItemIndex;
      {move ListBox item}
      Items.Move(i, i+1);
      ItemIndex := i+1;
      {move Field}
      Table1.Fields[i].Index:=i+1;
    end;
```

end.

Последняя строка в примере как раз та, которая фактически изменяет индекс колонки, которую пользователь хочет переместить. Две строки кода непосредственно перед ней перемещают текущую строку в ListBox на новую позицию.

Внешний вид программы DBGR_RT показан на рис. 4.21.

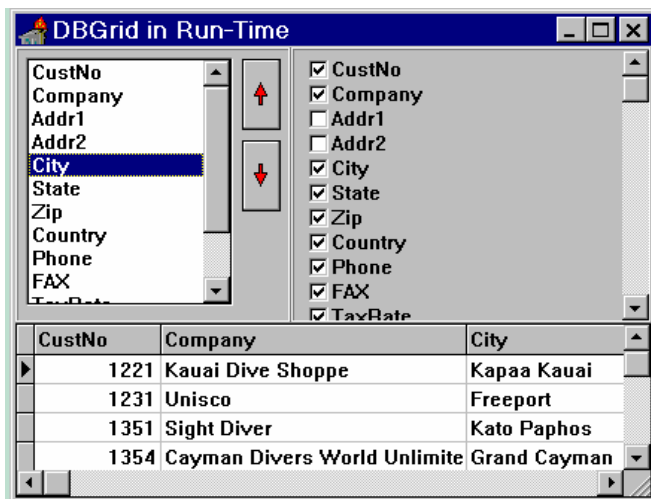


Рис. 4.21. Программа DBGR_RT

Заключение по разделу

Создание СУБД для решения поставленной задачи сводится к следующему:

1. Формирование экранной формы для меню.
2. Формирование экранной формы для отображения БД, причем:
 - формирование экранной формы для отображения данных одной таблицы;
 - формирование экранной формы для отображения 2-х таблиц (отображение данных двух таблиц без учета связи между ними; отображение данных таблиц с учетом связи между таблицами);
 - включение в состав экранной формы вычисляемого поля;
 - создание экранной формы с использованием SQL-запроса.

3. Формирование экранной формы для ввода данных в БД с кнопками навигации и управления состоянием БД.

ГЛАВА V. СОЗДАНИЕ ОТЧЕТНЫХ ФОРМ В DELPHI

Результаты работы системы необходимо распечатать на принтере или хранить в виде файла. Для этой цели создается отчетная форма.

Создание отчетной формы и организация отчета процедуры, несколько отличающаяся от процедуры создания экранной формы, поэтому эти материалы рассмотрим в отдельной главе.

Возможны различные варианты создания отчетов с предварительным отображением на экране:

- отображение таблицы на экране, а затем выбор записи из нее для отчета;
- отображение отчета на экране с выбранной строки;
- отображение отчета.

5.1. Компоненты для построения отчетов

На странице палитры компонентов *QReport Delphi* имеется более двадцати компонентов, применяемых для построения отчетов. Основными из которых являются следующие:

TQuickRep, TQRBand, TQRSubDetail, TQRGroup, TQRLabel, TQRDBText, TQRExpr, TQRSysDat, TQRMemo, TQRRichText, TQRDBRich Text, TQRShape, TQRImage, TQRChart.

Назначение этих компонентов таково:

1) ***TQuickRep*** – является базовым для всех остальных и определяет поведение отчета в целом. Другие компоненты определяют составные части отчета;

2) ***TQRBand*** – заготовка для расположения данных, заголовков, титула отчета и др.; отчет, в основном, строится из компонентов ***TQRBand***, которые реализуют:

- область заголовка отчета;
- область заголовка страницы;
- область заголовка группы;
- область названий столбцов отчета;
- область детальных данных, предназначенную для отображения данных самого нижнего уровня детализации;
- область подвала группы;
- область подвала страницы;
- область подвала отчета.

3) ***TQRSubDetail*** – определяет область, в которой располагаются

данные подчиненной таблицы при реализации в отчете связи Master-Detail на основе существующей связи между ТБД;

4) **TQRGroup** – применяется для группировок данных в отчете;

5) **TQRLabel** – позволяет разместить в отчете статический текст;

6) **TQRDBText** – позволяет разместить в отчете содержимое поля набора данных;

7) **TQRExpr** – применяется для вывода значений, являющихся результатом вычисления выражений; алгоритм вычисления выражений строится при помощи редактора формул данного компонента;

8) **TQRSysDate** – служит для вывода в отчете даты, времени, номера страницы, счетчика повторений какого-либо значения и т.д.;

9) **TQRMemo** – служит для вывода в отчете содержимого полей комментария;

10) **TQRRichText** – служит для вывода в отчете содержимого полей форматированных комментариев;

11) **TQRDBRichText** – служит для вывода в отчете содержимого полей форматированных комментариев, источником которых является поле набора данных;

12) **TQRShape** – служит для вывода в отчете графических фигур, например, прямоугольников;

13) **TQRImage** – служит для вывода в отчете графической информации, источником которой является поле набора данных;

14) **TQRChart** – служит для встраивания в отчет графиков.

Компонент TQuickRep

Компонент *TQuickRep* определяет поведение и характеристики отчета в целом. При размещении этого компонента в форме появляется сетка отчета (рис. 5.1).

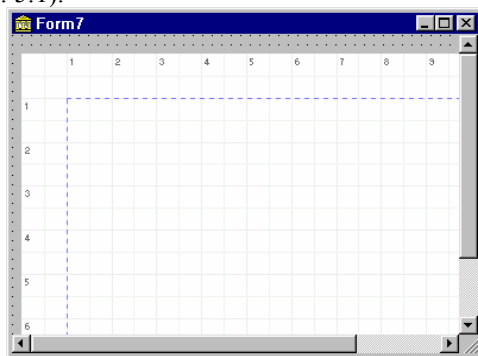


Рис. 5.1. Пустая сетка отчета. Образуется после размещения в форме компонента TquickRep

В дальнейшем в этой сетке располагаются составные части отчета, например, группы *TQRBand* (рис. 5.2).

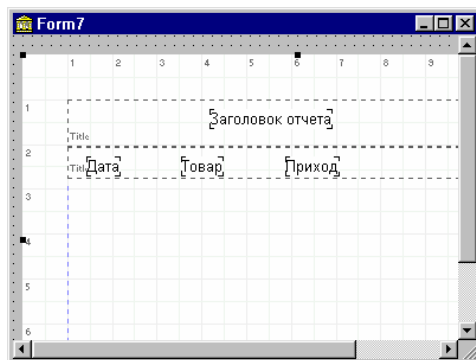


Рис. 5.2. Сетка отчета с размещенными в ней компонентами отчета

Перечислим важнейшие свойства, методы и события компонента TQuickRep.

Свойства данного компонента.

1. Property Bands: TquickRepBands.

Данные свойства состоят из множества логических значений (False/True), которые определяют включение в отчет отдельных видов составляющих:

- *HasColumnHeader* – заголовка столбцов отчета;
- *HasDetail* – детальной информации;
- *HasPageFooter* – подвала страницы;
- *HasPageHeader* – заголовка страницы;
- *HasSummary* – подвала отчета;
- *HasTitle* – заголовка отчета.

2. Property DataSet: TDataSet.

Оно указывает на набор данных на основе которого и создается отчет. Обычно для выдачи отчета используется один набор данных (НД).

Если нужно вывести связанную информацию из нескольких таблиц БД, ее объединяют в одном НД при помощи оператора SELECT. В этом случае в качестве НД для отчета может использоваться компонент TQuery. Информацию из нескольких связанных НД можно включать в

отчет, если эти наборы данных связаны в приложении отношением Master-Detail. В этом случае в качестве НД отчета указывается Master-набор, а ссылка на соответствующие Detail-наборы осуществляется в компонентах TQRSubDetail.

Если в отчет нужно включить информацию из несвязанных наборов данных применяют композитный отчет, т. е. отчет, составленный из группы других отчетов.

3. Property Frame: TQRFrame.

Данное свойство определяет параметры рамки отчета:

- *Color* – цвет линии рамки;
- *DrawBottom* – определяет, следует ли выводить линию снизу;
- *DrawLeft* – определяет, следует ли выводить линию слева;
- *DrawRight* – определяет, следует ли выводить линию справа;
- *DrawTop* – определяет, следует ли выводить линию сверху;
- *Style* – определяет стиль линии;
- *Width* – определяет ширину линии в пикселях.

4. Property Page: TQRPage – определяет параметры страницы.

5. Property PrinterSettings: TquickRepPrinterSettings – определяет параметры принтера.

6. Property PrintIfEmpty: Boolean – указывает (True), что следует печатать отчет даже в том случае, если он не содержит данных.

Методы данного компонента.

1. Procedure NewPage;

Данный метод выполняет переход на новую страницу. Может использоваться в обработчиках событий компонентов отчета *BeforePrint* или *AfterPrint* и не может – в обработчиках событий *OnPrint*, *OnStartPage* и *OnEndPage*.

2. Procedure Preview – выводит отчет в окно предварительного просмотра.

Чтобы во время разработки отчета просмотреть в окне предварительного просмотра содержимое отчета в том виде, как он будет выводиться на печать, необходимо:

- выбрать отчет при помощи мыши;
- нажать правую кнопку мыши;
- во всплывающем меню выбрать элемент *Preview*.

Следует заметить, что при этом не будут видны некоторые данные, например, значения вычисляемых полей наборов данных. Они будут выводиться только во время выполнения.

3. Procedure Print – печатает отчет на принтере.

4. Procedure PrinterSetup – обеспечивает установки параметров принтера.

События данного компонента.

1. Property AfterPreview : TQRAfterPreviewEvent – наступает после закрытия окна предварительного просмотра отчета.

2. Property AfterPrint: TQRAfterPrintEvent – наступает после вывода отчета на печать.

3. Property BeforePrint: TQRBeforePrintEvent – наступает в момент генерации отчета, до выдачи окна предварительного просмотра отчета и до вывода отчета на печать.

4. Property OnEndPage : procedure(Sender : TObject) – наступает в момент подготовки к генерации последней страницы отчета

5. Property OnStartPage : procedure(Sender : TObject) – наступает в момент подготовки к генерации первой страницы отчета.

Компонент TQRBand

Компоненты *TQRBand* являются основными составными частями отчета и используются для размещения в них статического текста и данных.

Данный компонент имеет следующие свойства:

1. Property BandType: TQRBandType – определяет месторасположение компонента в отчете и его поведение.

Возможные значения этого свойства таковы.

1) *rbTitle* – определяет компонент заголовка отчета. Информация, размещенная в компоненте *TQRBand*, располагается перед всеми другими частями отчета. Этот вид компонента *TQRBand* используется для вывода заголовочной информации отчета.

2) *rbPageHeader* – определяет компонент заголовка страницы. Информация, размещенная в компоненте с этим значением свойства **BandType**, выводится всякий раз при печати новой страницы отчета прежде всех иных частей отчета (но после информации, размещенной в компоненте заголовка отчета - для первой страницы).

3) *rbDetail* – компонент детальной информации. Выводится всякий раз при переходе на новую запись в НД отчета. Отчет печатается для всех записей НД, определяемого свойством отчета *DataSet*, начиная с первой и заканчивая последней. Позиционирование на первую запись и последовательный перебор записей в НД осуществляется компонентом *TQuickRep* автоматически.

4) *rbPageFooter* – компонент подвала страницы. Выводится для каждой страницы отчета после всех иных данных на странице.

5) *rbSummary* – компонент подвала отчета. Выводится на последней странице отчета после всей иной информации, но перед подвалом последней страницы отчета.

6) *rbGroupHeader* – компонент заголовка группы. Применяется при группировках информации в отчете. Выводится всякий раз при выводе новой группы.

7) *rbGroupFooter* – компонент подвала группы. Применяется при группировках информации в отчете. Выводится всякий раз при окончании вывода группы, после всех данных группы.

8) *rbSubDetail* – компонент для выдачи детальной информации из подчиненного набора данных, при выводе в отчете информации из двух или более наборов данных, связанных в приложении при помощи механизма Master-Detail. Это значение присваивается компоненту автоматически, когда генерируется компонент TQRBand при размещении в форме компонента TQRSubDetail. Программа не должна устанавливать это значение в свойство *BandType*.

9) *rbColumnHeader* – компонент для размещения заголовков столбцов. Размещается в отчете на каждой странице после заголовка страницы.

10) *rbOverlay* – используется для совместимости с более ранними версиями отчетов.

2. Property Enabled: Boolean – указывает, печатается в отчете (True) или нет (False) информация, содержащаяся в компоненте TQRBand.

3. Property ForceNewPage: Boolean – указывает, должна ли информация в составе TQRBand всегда печататься с новой страницы (True) или нет (False).

Компонент TQRBand имеет следующее событие.

Property BeforePrint: TQRBeforePrintEvent – наступает перед печатью информации, размещенной в области компонента TQRBand.

5.2. Способы создания простейших отчетов

В данном пункте описываются способы создания простейших отчетов с помощью компонентов TQuickRep и TQRBand.

Компоненты *TQuickRep* и *TQRBand* являются минимально достаточными для создания простого отчета, не содержащего внутри себя группировок информации.

Таблица БД Prihod.DB содержит сведения о поступлении материалов на склада. В состав ТБД входят поля:

- N_Prih – уникальный номер события прихода материала;
- DataPrih – дата поступления;
- MATERIAL – наименование материала;
- KOLVO – количество единиц поступившего материала.

Создадим простейший отчет, состоящий из заголовка и сведений о поступлении материала. В отчет включаются все факты поступления материала. Сортировка производится по номеру события поступления материала.

Разместим в форме № 6 компонент TTable (имя Table1), свяжем с таблицей БД Prihod.DB и откроем его (*Active* = True). Разместим в форме компонент TQuickRep (имя QuickRep1) из страницы QReport палитры компонентов. Установим в свойство *DataSet* отчета значение Table1, назначив, таким образом, отчет набора данных, записи из которого будут выводиться в отчет. Добавим в отчет компонент TQRBand (имя QRBand1). Свойство *BandType* компонента QRBand1 по умолчанию установлено значение *rbTitle*, т. е. компонент QRBand1 определяет заголовок отчета. Разместим в пространстве отчета, занимаемом компонентом QRBand1, компонент TQRLabel (статический текст) с именем QRLabel1. Установим в свойство *Caption* этого компонента значение 'Поступление материалов на склад', установим в свойстве *Font* жирный наклонный шрифт высотой 16 пунктов. Вид отчета показан на рис. 5.3.

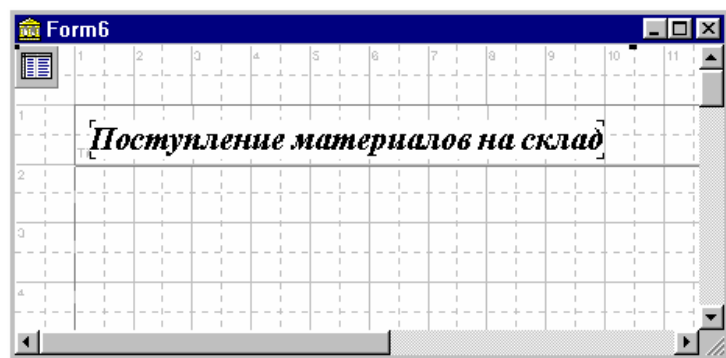


Рис. 5.3. В отчете определен только его заголовок

Теперь разместим в отчете данные, соответствующие текущей за-

писи таблицы Prihod. Для этого поместим в отчет новый компонент TQRBand (имя QRBand2) и установим в его свойство *BandType* значение *rbDetail*. Затем разместим в группе четыре компонента TQRDBText с именами QRDBText1 ... QRDBText4. Свяжем данные компоненты соответственно с полями НД – N_Prih, MATERIAL, DataPrih, KOLVO. Для этого в свойство *DataSet* каждого компонента QRDBText установим значение Table1, а в свойство *DataField* – значение имени соответствующего поля. Вид отчета показан на рис. 5.4.

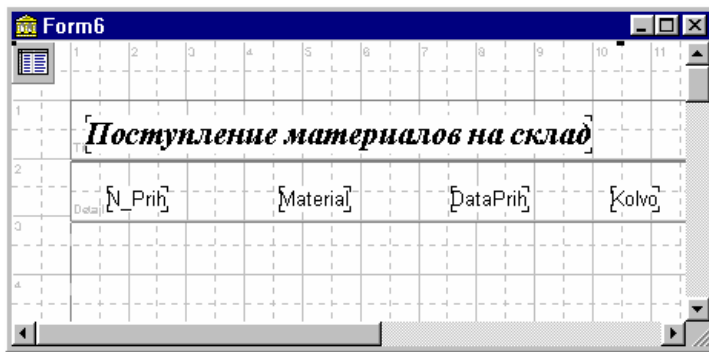
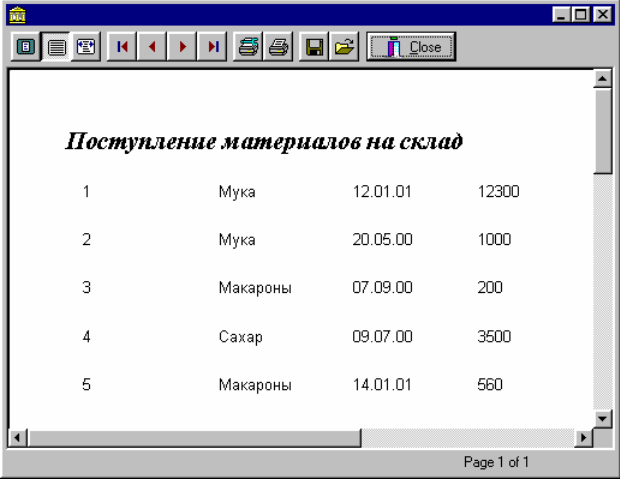


Рис. 5.4. Отчет с заголовком и группой детальной информации

Для просмотра получившегося отчета щелкнем по нему правой кнопкой мыши и из всплывающего меню выберем элемент *Preview*. Получим окно предварительного просмотра отчета (рис. 5.5).



Поступление материалов на склад

1	Мука	12.01.01	12300
2	Мука	20.05.00	1000
3	Макароны	07.09.00	200
4	Сахар	09.07.00	3500
5	Макароны	14.01.01	560

Page 1 of 1

Рис. 5.5. Содержимое отчета в окне предварительного просмотра

Вызов окна предварительного просмотра организуем при нажатии на кнопку “Просмотр” из формы № 4.

Для этого добавим в форму № 4 компонент `TButton`. Изменим заголовок этой кнопки (свойство *Caption*), используя *инспектор объектов*, на “Просмотр”. Определим для кнопки обработчик события нажатия кнопки, *OnClick*:

Form6. QuickRep1.Preview.

5.2.1. Представление заголовков столбцов отчета

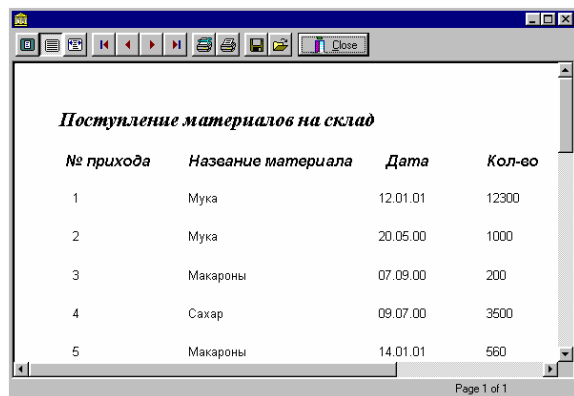
В данном пункте описывается представление заголовков столбцов отчета с помощью компонента `TQRBand`.

Компонент `TQRBand`, у которого в свойство *BandType* установлено значение *rbColumnHeader*, используется для представления заголовков столбцов. Заголовки столбцов определяются при помощи компонентов `TQRLabel`.

В отчете, рассмотренном в предыдущих разделах, добавим компонент `TQRBand` (имя `QRBand3`) и установим в его свойство *BandType* значение *rbColumnHeader*. Разместим в пространстве отчета, определяемом `QRBand3`, четыре компонента `TQRLabel` (имена `QRLabel2 ... QRLabel5`) и установим в свойства *Caption* этих компонентов соответственно значения “№ Прихода”, “Наименование материала”, “Дата”, “Количество”. В свойстве *Font* данных компонентов установим режим

подчеркивания шрифта, а сам шрифт определим как наклонный.

Выйдем в окно предварительного просмотра отчета. Для каждой страницы отчета вверху страницы будут выводиться названия столбцов (рис. 5.6).



№ прихода	Название материала	Дата	Кол-во
1	Мука	12.01.01	12300
2	Мука	20.05.00	1000
3	Макароны	07.09.00	200
4	Сахар	09.07.00	3500
5	Макароны	14.01.01	560

Рис. 5.6. В отчете появились заголовки столбцов

5.2.2. Оформление заголовка и подвала страницы отчета

В данном пункте описывается оформление заголовка и подвала страницы отчета с помощью компонента TQRBand.

Компонент TQRBand, у которого в свойство *BandType* установлено значение *rbPageHeader*, используется для показа заголовка страницы. Он выводится для каждой новой страницы перед выводом другой информации. Компонент TQRBand, у которого в свойство *BandType* установлено значение *rbPageFooter*, используется для показа подвала страницы. Он выводится для каждой страницы после вывода любой иной информации.

Информация в заголовке и подвале страницы может формироваться на основе статического текста (компоненты TQRLabel), значений полей (компоненты TQRDBText) и результатов вычисления выражений (компоненты TQRExpr).

В отчете, рассмотренном в предыдущих разделах, разместим компонент TQRBand (имя QRBand4) и установим в его свойство *BandType* значение *rbPageHeader*. Не будем размещать в заголовке страницы никакого текста, просто подчеркнем линию вверху страницы. Для этого установим в свойство компонента заголовка страницы *Frame. DrawTop* значение *True*, что обеспечивает вывод линии по

верхнему краю области, занимаемой компонентом.

Аналогичным образом определим в отчете компонент подвала страницы (имя *QRBand5*) и установим в его свойство *BandType* значение *rbPageFooter*, а в *Frame. DrawBottom* значение *True*, что обеспечивает вывод линии по нижнему краю области, занимаемой компонентом.

Войдя в режим предварительного просмотра отчета, увидим, что вверху и внизу каждой страницы отчета выводятся линии.

5.2.3. Представление в отчете вспомогательной и системной информации

В данном пункте описывается представление в отчете вспомогательной и системной информации с помощью компонента *TQRSysData*.

Компонент *TQRSysData* используется для показа вспомогательной и системной информации. Вид показываемой информации определяется свойством

property Data: *TQRSysDataType*.

Ниже указаны возможные значения этого свойства:

- 1) *qrsColumnNo* – номер текущей колонки отчета (для одноколоночного отчета всегда 1).
- 2) *qrsDate* – текущая дата.
- 3) *qrsDate Time* – текущие дата и время.
- 4) *qrsDetailCount* – число записей в НД; при использовании нескольких НД – число записей в master-наборе. Для случая, когда НД представлен компонентом *TQuery*, эта возможность может быть недоступной, что связано с характером работы компонента *TQuery*, который возвращает столько записей, сколько необходимо для использования в текущий момент, а остальные предоставляет по мере надобности;
- 5) *qrsDetailUNo* – номер текущей записи в НД. При наличии нескольких наборов – номер текущей записи в master-наборе.
- 6) *qrsPageNumber* – номер текущей страницы отчета.
- 7) *qrsPageCount* – общее число страниц отчета.
- 8) *qrsReport Title* – заголовок отчета.
- 9) *qrsTime* – текущее время.

Разместим в компоненте подвала отчета QRBand5 два компонента TQRSysData (имена QRSysData1... QRSysData2). В свойство *Data* первого из них установим значение *qrsDate* (текущая дата), второго – значение *qrsPageNumber* (номер текущей страницы отчета). Войдем в режим предварительного просмотра результатов отчета. Теперь в подвале каждой страницы отчета выводятся номер страницы и текущая дата.

5.3. Способы группировки данных в отчете

Выше мы рассмотрели отчет, в котором информация о поступлении материалов из ТБД Prihod.DB на склад выводилась "как есть". Такое представление информации в отчете не всегда информативно. Пусть, например, в нашем случае требуется сгруппировать информацию по материалам.

Для группировки информации используется компонент TQRGroup. Его свойство *Expression* указывает *выражение*. В группу входят записи НД, удовлетворяющие условию *выражения*. При смене значения *выражения* происходит смена группы. Для каждой группы, если определены, выводятся заголовок группы и подвал группы. В качестве заголовка группы служит компонент TQRBand со значением свойства *BanaType*, равным *rbColumnHeader*. В качестве подвала группы служит компонент TQRBand со значением свойства *BanaType*, равным *rbGroup Footer*.

Свойство *FooterBand* компонента TQRGroup содержит ссылку на компонент подвала группы.

В заголовке группы, как правило, выводится выражение, по которому происходит группировка, и различные заголовки, если они нужны. В подвале группы обычно выводится агрегированная информация – суммарные, средние и т. п. значения по группе.

Пример. Построим новый отчет о поступлении материала на склад, в котором информация группируется по наименованию материала. Для этого в форме № 7 определим набор данных отчета (компонент TTable, имя Table1, свойства TableName – Prihod.DB, Active – True). Установим у НД текущий индекс по полю Material (в свойстве *FieldIndexNames* или *IndexName*). Разместим в отчете:

- 1) заголовок отчета – компонент TQRBand с именем QRBand1, свойство *BandType* = *rbTitle*;
- 2) заголовок столбцов – компонент TQRBand с именем QRBand2, свойство *BandType* = *rbColumnHeader*;
- 3) группу – компонент TQRGroup с именем QRGroup1.
- 4) область детальной информации – компонент TQRBand с именем QRBand3, свойство *BandType* = *rbDetail*;
- 5) подвал группы – компонент TQRBand с именем QRBand4, свойство *BandType* = *rbGroupFooter*.

В компоненте QRGroup1 установим:

- в свойство *FooterBand* значение QRBand4;
- в свойство *Expression* значение Table1.MATERIAL, которое является формулой и строится в редакторе формул.

Поскольку свойство *Expression* не визуализирует значения выражения, необходимо разместить в группе компонент QRExpr (имя QRExpr1) и определить значение его свойства *Expression* так, чтобы оно содержало Table1.MATERIAL (рис. 5.7).

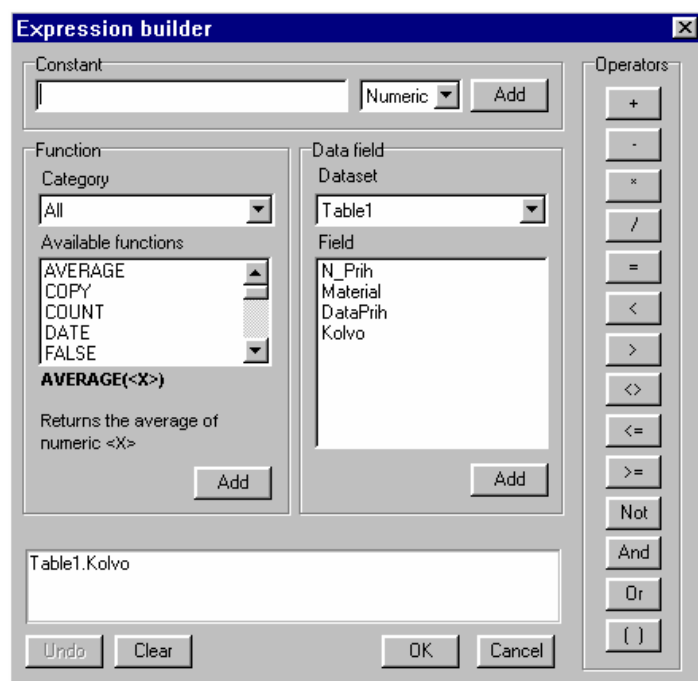


Рис. 5.7. Построение формулы

В компоненте подвала группы QRBand4 будем подсчитывать сумму по полю Kolvo (сумму поступившего конкретного материала). Для этого разместим в подвале группы компонент TQRExpr (имя QRExpr2) и определим значение его свойства *Expression* так, чтобы оно содержало формулу SUM(Table1.KOLVO).

В группе детальной информации разместим компоненты TQRDBText, связанные с полями N_Prih, Material, DataPrih и Kolvo набора данных (свойства DataSet и DataField).

Заполним другие области отчет статическим текстом, как это показано на рис. 5.8.

Рис. 5.8. Макет отчета с группировкой по товару

На рис. 5.9 показан отчет в режиме предварительного просмотра.

Вызов окна предварительного просмотра отчета организуем при нажатии на кнопку "Группа" из формы № 4. Добавим в форму № 4 компонент TButton. Определим для кнопки обработчик события нажатия кнопки, *OnClick*:

Form7. QuickRep1.Preview.

На рис. 5.9 показан подвал одной из групп – там выводится информация о суммарном приходе материала.

5.4. Формирование отчета с использованием фильтра

Свойство *Filtered*, установленное в *True*, инициирует фильтрацию, условие которой записано в обработчике события *OnFilterRecord*. Например, если в НД установлен фильтр

```
begin
Accept := DataSet['KOLVO'] >= 500;
end;
```

то установка *Table1.Filtered* в *True* приведет к фильтрации; в результирующем наборе данных будут показаны только записи, у которых поле 'KOLVO' содержит значение 500 и более.

Установка *Filtered* в *False* приведет к отмене фильтрации, усло-

вия которой указаны в событии *OnFilterRecord*.

Поступление материалов на склад

№ прихода	Материал	Дата поступления	Кол-во
ТОВАР: Макароны			
3	Макароны	07.09.00	200
5	Макароны	14.01.01	580
Итого по товару:			780
ТОВАР: Мука			
1	Мука	12.01.01	12300
2	Мука	20.05.00	1000
Итого по товару:			14060
ТОВАР: Сахар			
4	Сахар	09.07.00	3500
Итого по товару:			17560

Page 1 of 1

Рис. 5.9. Отчет с группировкой по товару в окне предварительного просмотра

Заключение по разделу

В данном разделе проведен обзор компонентов для построения отчетов и рассмотрены следующие варианты формирования отчета:

- 1) простой отчет поступления материалов на склад без группировки данных;
- 2) отчет с группировкой данных базы по одному полю;
- 3) отчет с фильтром по определенному полю.

ГЛАВА VI. УСОВЕРШЕНСТВОВАНИЕ СИСТЕМЫ В ПЕРИОД ЭКСПЛУАТАЦИИ

6.1. Оформление системы

Разработанная система (или приложения БД) должна удовлетворять не только функциональным требованиям, она еще должна удовлетворять и потребительские требования, должна быть гибкой и удобной для работы пользователей. Поэтому, после завершения создания базы данных и реализации основных функций системы базы данных следует вести эстетическое оформление интерфейса. Для этого добавим к приложению app1 несколько элементов, которые придадут ему более профессиональный вид и сделают более удобным в использовании.

Зададим приложению заголовок и пиктограмму, по умолчанию Delphi присваиваем одну и ту же стандартную пиктограмму каждому создаваемому приложению. Однако пиктограмма должна быть такой, чтобы по ней можно было легко определить функции приложения. Кроме того, она должна помочь выделить нужное среди других.

В Windows и Delphi содержится большое количество готовых пиктограмм. Можно также использовать редактор изображений для создания собственной пиктограммы.

Готовую пиктограмму следует связать с приложением. Для этого выберем Project|Options и щелкнем на вкладке Application. В поле Title введем текст Systema – Sklad, а затем щелкнем на кнопке Load Icon и укажем файл пиктограммы.

Заданный заголовок будет использоваться в панели задач Windows для идентификации приложения.

6.2. Тестирование и доработка приложения

Для выявления недоработок необходимо провести испытания приложения во всех режимах функционирования.

После тестирования выяснилось, что у нас остался не задействованным пункт меню *Ввод/Название материала*. Создадим форму для добавления новых материалов и их характеристик в существующий список материалов. Разместим на нем компонент TDBGrid, установим свойство *DataSource* в значение DataSource1. Добавим в форму компонент кнопки TButton, дадим имя этому компоненту (свойство *Name*) *CancelBtn*, изменим заголовок кнопки (свойство *Caption*) на "Заккрыть".

Свяжем с кнопкой код, который будет выполняться при нажатии на кнопку пользователем. Для создания процедуры обработки этого события – *OnClick*, находясь в форме, дважды щелкнем мышью на кнопку "Закрыть". В редакторе кода напишем следующий код:

```
Close;  
Form1.Visible := True.
```

Перейдем на форму № 1 и напишем код для пункта меню *Ввод/Название материала* код вызова формы № 8:

```
Form1.Visible := False;  
Form8.Visible := True.
```

На этом подготовку приложения считаем завершенной.

6.3. Сопровождение системы

Разработанная система не функционирует в замкнутом пространстве. Она функционирует в окружении рыночных условий и факторов, характеристики в условиях динамичного рынка, изменения законодательной базы. Внешняя среда системы, как говорится жизнь не стоит на месте, так и наше разработанное приложение может претерпевать изменения в связи с изменившимися внешними условиями. Для какого-то момента времени может потребоваться отчет другой формы, например, показать на какую сумму поступило материалов на склад.

Доработаем наше приложение с учетом новых требований.

Доработка № 1.

Создадим новую форму отчета (форма № 9). Разместим в форме компонент TQuickRep (имя QuickRep1), но обращение к набору данных организуем через Query1. Для этого свойство DataSet установим как DataModule5.Query1. Добавим другие компоненты и области отчета заполним статическим текстом, как это показано на рис. 6.1.

Поступление товаров на склад				
Дата поступления	Название материала	Кол-во	Цена за ед.	Сумма
DataPri	Material	Kolvo	Zena	Stoim
Итого:		QRLabel7		

Рис. 6.1. Макет итогового отчета

В конце отчета добавим итоговую сумму всех поступивших материалов в словесном виде. Для этого добавим компонент `TQRBand` с именем `QRBand4`, свойство `BandType = rbSummary`.

Разместим в `QRBand4` компонент `TQRLabel` (имя `QRLabel7`).

Перейдем в `Unit4` и добавим в обработчик события `OnClick` кнопки "Итого" такой код:

```
procedure TForm4.Button5Click(Sender: TObject);
var ss: String;
begin
  Str(TempTotal,ss);
  Form9.QRLabel7.Caption := SumNumToFull(TempTotal);
  Form9.QuickRep1.Preview;
end.
```

В `Unit5` определим переменную `TempTotal` и добавим код вычисления итоговой суммы поступивших на склад материалов `TempTotal`.

```
var
  DataModule5: TDataModule5;
  TempTotal: Integer;
implementation.
```

Двойным нажатием мыши на свойство `AfterOpen` в инспекторе объектов откроем `Unit5` и добавим следующий код:

```
Procedure TDataModule5.Query1AfterOpen(DataSet: TDataSet);
```

```

begin
Query1.First;
  TempTotal := 0;
  while not Query1.EOF do
  begin
    TempTotal := TempTotal + Query1.Stoim.Value;
    Query1.Next;
  end;
SumNumToFull(TempTotal);
end.

```

Добавим в проект модуль перевода числа в запись прописью Writenum.pas, используя меню Project|Add to Project . Листинг модуля приведен в Приложении.

Продолжим формирование отчета. В поле QRBand4 разместим компонент TQRLabel (имя QRLabel8), свойство *Caption* изменим на "Итого:". Добавим еще компонент TQRExpr (имя QRExpr1) и определим значение его свойства *Expression* так, чтобы оно содержало формулу SUM(Table1.Stoim). На рис.6.2 показан итоговый отчет в окне предварительного просмотра.

Дата поступления	Название материала	Кол-во	Цена за ед.	Сумма
20.05.00	Мука	1000	50	50000
09.07.00	Сахар	3500	67	234500
07.09.00	Макароны	200	120	24000
12.01.01	Мука	12300	50	615000
14.01.01	Макароны	560	120	67200
Итого :		ДЕВЯТЬСОТ ДЕВЯНОСТО ТЫСЯЧ СЕМЬСОТ ТЕНГЕ 00 ТИЫН		

Рис. 6.2. Отчет по товарам с итоговым значением

Доработка № 2.

Для подготовки отчета Excel имеет большие возможности (графические представления и т.д.). Кроме того, иногда бывают для бизнес-процесса удобно ряд операции выполнить в среде Excel. Поэтому, чтобы использовать этот ресурс с помощью процедуры OLE необходимо передать данные в Excel.

Чтобы продемонстрировать это приведем пример-задачу, который необходимо выполнить на этапе сопровождения.

Листинг 1. Модуль перевода числа в запись прописью в отчете Writsum.pas

```
Unit Writsum;
interface
uses SysUtils;
{Функция перевода суммы, записанной цифрами суммы
прописью: например, 23.12 -> двадцать три тенге 12 тиын. Пе-
реводит до 999999999 тенге 99 тиын. Функция не отслеживает
правильное ли значение получено в параметре Number (т. е.
положительное и округленное с точностью до сотых) – эту про-
верку необходимо провести до вызова функции.}
function SumNumToFull(Number:real):string;
implementation
function SumNumToFull(Number:real):string;
var
PartNum, TruncNum, NumTMP, D: integer;
NumStr : string;
i, R : byte;
Flag11 : boolean;
Begin
D:=1000000;
R:=4;
TruncNum:=Trunc(Number); //выделяем тенге
if TruncNum<>0
then
repeat
PartNum:=TruncNum div D;
Dec(R);
D:=D div 1000;
```

```
Until PartNum<>0
else
R:=0;
// перевод тенге
FOR i:=R DOWNT0 1 DO
BEGIN
Flag11:=False;
NumTMP:=PartNum div 100; {выделение цифры сотен}
Case NumTMP of
1: NumStr:=NumStr+'СТО ';
2: NumStr:=NumStr+'ДВЕСТИ ';
3: NumStr:=NumStr+'ТРИСТА ';
4: NumStr:=NumStr+'ЧЕТЫРЕСТА ';
5: NumStr:=NumStr+'ПЯТЬСОТ ';
6: NumStr:=NumStr+'ШЕСТЬСОТ ';
7: NumStr:=NumStr+'СЕМЬСОТ ';
8: NumStr:=NumStr+'ВОСЕМЬСОТ ';
9: NumStr:=NumStr+'ДЕВЯТЬСОТ ';
end;
NumTMP:=(PartNum mod 100) div 10; {выделение цифры
десятков }
Case NumTMP of
1: begin
NumTMP:=PartNum mod 100;
case NumTMP of
10: NumStr:=NumStr+'ДЕСЯТЬ ';
11: NumStr:=NumStr+'ОДИНАДЦАТЬ ';
12: NumStr:=NumStr+'ДВЕНАДЦАТЬ ';
13: NumStr:=NumStr+'ТРИНАДЦАТЬ ';
14: NumStr:=NumStr+'ЧЕТЫРНАДЦАТЬ ';
15: NumStr:=NumStr+'ПЯТНАДЦАТЬ ';
16: NumStr:=NumStr+'ШЕСТНАДЦАТЬ ';
17: NumStr:=NumStr+'СЕМНАДЦАТЬ ';
18: NumStr:=NumStr+'ВОСЕМНАДЦАТЬ ';
19: NumStr:=NumStr+'ДЕВЯТНАДЦАТЬ ';
end;
case i of
3: NumStr:=NumStr+'МИЛЛИОНОВ ';
2: NumStr:=NumStr+'ТЫСЯЧ ';
1: NumStr:=NumStr+'ТЕНГЕ ';
end;
```

```
        Flag11:=True;
    end;
    2: NumStr:=NumStr+'ДВАДЦАТЬ ';
    3: NumStr:=NumStr+'ТРИДЦАТЬ ';
    4: NumStr:=NumStr+'СОРОК ';
    5: NumStr:=NumStr+'ПЯТЬДЕСЯТ ';
    6: NumStr:=NumStr+'ШЕСТЬДЕСЯТ ';
    7: NumStr:=NumStr+'СЕМЬДЕСЯТ ';
    8: NumStr:=NumStr+'ВОСЕМЬДЕСЯТ ';
    9: NumStr:=NumStr+'ДЕВЯНОСТО ';
    end;
    NumTMP:=PartNum mod 10; {выделение цифры единиц}
    if not Flag11 then
    begin
        case NumTMP of
            1: if i=2 then NumStr:=NumStr+'одна ' else
NumStr:=NumStr+'ОДИН ';
            2: if i=2 then NumStr:=NumStr+'две ' else
NumStr:=NumStr+'ДВА ';
            3: NumStr:=NumStr+'ТРИ ';
            4: NumStr:=NumStr+'ЧЕТЫРЕ ';
            5: NumStr:=NumStr+'ПЯТЬ ';
            6: NumStr:=NumStr+'ШЕСТЬ ';
            7: NumStr:=NumStr+'СЕМЬ ';
            8: NumStr:=NumStr+'ВОСЕМЬ ';
            9: NumStr:=NumStr+'ДЕВЯТЬ ';
        end;
        case i of
            3: case NumTMP of
                1 : NumStr:=NumStr+'МИЛЛИОН ';
                2,3,4: NumStr:=NumStr+'МИЛЛИОНА ';
                else NumStr:=NumStr+'МИЛЛИОНОВ ';
            end;
            2: case NumTMP of
                1 : NumStr:=NumStr+'ТЫСЯЧА ';
                2,3,4: NumStr:=NumStr+'ТЫСЯЧИ ';
                else if PartNum<>0 then NumStr:=NumStr+'ТЫСЯЧ ';
            end;
            1: case NumTMP of
                1 : NumStr:=NumStr+'ТЕНГЕ ';
                2,3,4: NumStr:=NumStr+'ТЕНГЕ ';
```

```
        else NumStr:=NumStr+'ТЕНГЕ ';
        end;
        end;
        end;
    if i>1 then begin
        PartNum:=(TruncNum mod (D*1000)) div D;
        D:=D div 1000;
        end;
    END;
//перевод тиын
    PartNum:=Round(Frac(Number)*100);
    if PartNum=0 then
        begin
            SumNumToFull:=NumStr+'00 ТИЫН';
            Exit;
            End;
        NumTMP:=PartNum div 10; {выделение цифры десятков }
        if NumTMP=0 then NumStr:=NumStr+'0'+IntToStr(PartNum)+'
    ,
else NumStr:=NumStr+IntToStr(PartNum)+' ';
    NumTMP:=PartNum mod 10; {выделение цифры единиц}
    Case NumTMP of
    1: if PartNum<>11 then NumStr:=NumStr+'ТИЫН'
        else NumStr:=NumStr+'ТИЫН';
    2,3,4: if (PartNum<5) or (PartNum>14)
        then NumStr:=NumStr+'ТИЫН'
        else NumStr:=NumStr+'ТИЫН';
        else NumStr:=NumStr+'ТИЫН';
        end;
    SumNumToFull:=NumStr;
    end; //---SumNumToFull
end.
```

Заключение по разделу

В данном разделе проведены некоторые способы улучшения системы после завершения разработки основной части системы.

Крупные системы после внедрения и ввода в эксплуатацию параллельно и одновременно ведут обслуживание, администрирование и

сопровождение, а также периодические их тестирования.

В данном разделе продемонстрированы простые варианты всех этих процессов.

Библиографическое описание

4. *Ускенбаева Р.К., Булабаев Ж.Т., Куандыков А.А.* Руководство по созданию системы базы данных в среде Delphi: Учеб. пособие. – Алматы: КазНТУ, 2001. – 88 с.

ПРИЛОЖЕНИЯ

III. Глоссарий

ER – диаграмма – диаграмма "сущность-связь" – "мән-байланыс" диаграммасы;

QBE – Query-by-Example – кескін бойынша сұраныс;

SQL – Structured Query Language – құрылымдық сұраныс терезесі;

SQL – Язык структурированных запросов – құрылымдық сұраныс тілі;

1 НФ – первая нормальная форма – бірінші қалыпты форма;

2 НФ – вторая нормальная форма – екінші қалыпты форма;

3 НФ – третья нормальная форма – үшінші қалыпты форма;

БД – база данных – деректер қоры;

ИС – информационная система ақпараттық жүйе;

КМД – концептуальная модель данных – деректердің концептуальды моделі;

СУБД – система управления базами данных – деректер қорын басқару жүйесі;

ФМД – физическая модель данных – деректердің физикалық моделі;

ЯОД – Язык описания данных – деректерді сипаттау тілі;

QBE – Язык запросов по образцу – кескін бойынша сұраныс тілі;

ЯИМ – Язык инфологического моделирования – инфологиялық модельдеу тілі;

Домен – множество атомарных значений одного и того же типа – сол бір типтегі атомардық мән көпмүшесі;

Отношение – таблица (иногда Файл) – кесте (кейде Файл);

Кортеж – строка (иногда Запись) – жол (кейде Жазба);

Атрибут – столбец, Поле – тік жол, Өріс;

Степень отношения – это число его атрибутов – оның атрибуттар саны;

Блок – диск, блок, данных – диск, деректер блогы;

ACID – транзакция свойство – қасиет транзакциясы;

Bag – мультимножество – мультикөпмүше;

CLI – интерфейс, уровня вызовов – интерфейс, шақыру деңгейі;

DBA – база данных администратор – деректер қоры әкімшілігі;

DBMS – СУБД – деректер қорын басқару жүйесі (ДҚБЖ);

DDL – язык определения данных, компилятор – деректерді анықтау тілі, компилятор;

DML – язык управления данными – деректерді басқару тілі;

DRAM – память, динамическая с произвольным доступом – еркін катынауы бар динамикалық жады;

FD – зависимость, функциональная – функциональды тәуелділік;

IDL – язык описания интерфейса – интерфейсті сипаттау тілі;

ODL – язык определения объектов – объектіні анықтау тілі;

ODBC – интерфейс, уровня вызовов – интерфейс, шақыру деңгейі;

ROLAP – оперативная аналитическая обработка реляционных данных;

Kd-дерево – группирование промежуточных вершин – аралық төбелер тобы;

OLAP – On-Line Analytical Processing – ключевой компонент организации хранилищ данных – деректерді сақтаудың ұйымдастырудың кілттік компонентті;

CORBA – Common Object Request Broker Architecture;

ADDNEW – добавить новую запись – жаңа жазба үстемелеу;

APPEND – добавить новый объект в коллекцию – коллекцияға жаңа объект үстемелеу;

BEGINTRANS – начать транзакцию – транзакцияны бастау;

COMMIT TRANS – сохранить все изменения и завершить текущую транзакцию – ағымдағы транзакция-ның барлық өзгерістерін сақтау және аяқтау;

ROLLBACKTRANS – откат транзакции – транзакция шегінісі;

CLOSE – закрыть указанный открытый объект и все ему подчиненные – көрсетілген ашық объектіні және оның барлық бағыңыңқы объектілерін жабу;

DELETE – удалить текущую запись открытого объекта – ашық объектінің ағымдағы жазбасын жою;

REQUERY – обновить данные объекта путем повторного осуществления запроса, который является базовым для этого объекта – осы объектінің қоры болып табылатын деректерді сұраныс көмегімен жаңарту;

UPDATE – обновить данные текущей записи объекта – ағымдағы жазба объектісінің деректерін жаңарту;

CANCELUPDATE – отменить обновление – жаңартуға тыйым салу;

Концептуальное проектирование – сбор, анализ и редактирование требований к данным;

Логическое проектирование – преобразование требований к данным в структуры данных;

Физическое проектирование – определение особенностей хранения данных, методов доступа и т. д.

II 2. Тестовые задания для самоконтроля

1. Отношение в терминах реляционной алгебры, это:

- А) Любая таблица;
- В) Взаимосвязь между двумя сущностями;
- С) Таблица, характеризующаяся рядом свойств;
- Д) Таблица с повторяющимися строками;
- Е) Таблица с повторяющимися столбцами.

2. В результате операции декартового произведения $R_1 \times R_2$ получено отношение R_3

R_1

Фамилия	№ зачётки
Иванов	9801
Петров	9805

R_2

Предмет	Дата
БД и СТ	10/06/00
СПО	16/06/00

А) R_3

Фамилия	№ зач	Предмет	Дата
Иванов	9801	БДиСТ	10/06/00
Петров	9805	СПО	16/06/00

В) R_3

Фамилия	№ зач	Предмет	Дата
Иванов	9801	БДиСТ	10/06/00
Иванов	9801	СПО	16/06/00
Петров	9805	БДиСТ	10/06/00
Петров	9805	СПО	16/06/00

С) R_3

Фамилия	№ Зачётки
---------	-----------

Иванов	9801
Петров	9805

D) R₃

№ зач	Дата
9801	10/06/00
9801	16/06/00
9805	10/06/00
9805	16/06/00

E) R₃

Фамилия	Дата
Иванов	10/06/00
Иванов	16/06/00
Петров	10/06/00
Петров	16/06/00

3. Для следующего отношения что является первичным ключом ПРОДАЖА

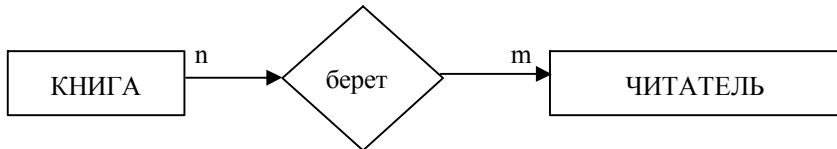
№ Товара	Дата покупки	Количество
10	15/01/00	100
11	15/01/00	55
10	16/01/00	40

- A) № Товара;
- B) № Товара, дата покупки;
- C) дата покупки;
- D) № Товара, дата покупки, количество;
- E) количество.

4. Отношение находится во II нормальной форме, если

- A) оно находится в III нормальной форме и каждый атрибут зависит от части составленного ключа;
- B) отношение находится в I нормальной форме и каждый не ключевой атрибут функционально полно зависит от составного ключа;
- C) имеются многозначные зависимости между атрибутами;
- D) есть повторяющиеся строки;
- E) есть повторяющиеся столбцы.

5. Если сущности КНИГА и ЧИТАТЕЛЬ связаны связью n:m, то по правилам



ER-метода достаточно:

- А) два отношения – КНИГА и ЧИТАТЕЛЬ;
- В) три отношения – КНИГА, ЧИТАТЕЛЬ и БЕРЕТ (формуляр);
- С) одно отношение – БЕРЕТ_КНИГУ (формуляр);
- Д) ни одного отношения;
- Е) два отношения – БЕРЕТ (формуляр) и ЧИТАТЕЛЬ.

6. Оператор GROUP BY в SQL запросе играет роль:

- А) упорядочивает записи;
- В) группирует записи;
- С) исключает записи согласно условию;
- Д) исключает группы согласно условию;
- Е) исключает повторяющиеся записи.

7. Функция SUM в SQL запросе используется для:

- А) вычисления среднего арифметического числового атрибута;
- В) вычисления количества записей;
- С) вычисления суммы значений числового атрибута;
- Д) нахождения максимального значения числового атрибута;
- Е) нахождения минимального значения числового атрибута.

8. Ключевое слово DESC в выражении ORDER BY SQL запроса используется для:

- А) упорядочение записей по возрастанию;

- В) упорядочение записей по убыванию;
- С) отмена упорядочения;
- Д) исключает группы согласно условию;
- Е) исключает повторяющиеся записи.

9. ER – метод (сущность – связь) есть:

- А) метод проектирования базы данных;
- В) метод проектирования программного обеспечения;
- С) метод разбиения универсального отношения путем устранения функциональных зависимостей;
- Д) метод поиска записи;
- Е) метод группировки записей.

10. Архитектура БД "клиент-сервер"

- А) запросы обрабатываются на сервере и результат отправляется на клиентскую часть;
- В) запросы обрабатываются на клиентской части;
- С) база данных хранится на клиентской части;
- Д) база данных хранится на сервере, запросы обрабатываются на клиентской части;
- Е) тоже, что и файл-сервер.

11. Какая команда SQL используется для представления доступа к БД?

- А) DROP TABLE;
- В) GRANT;
- С) ALTER TABLE;
- Д) CREATE TABLE;
- Е) CREATE VIEW.

12. На какой странице (вкладке) расположены компоненты для создания баз данных?

- А) Additional;
- В) InterBase;
- С) ADO;
- Д) Data Controls;

Е) MIDAS.

13. Какая компонента является однострочным редактором?

- А) Memo;
- В) Button;
- С) Edit;
- Д) RadioButton;
- Е) ScrollButton.

14. Чем является средство DataBaseDesktop?

- А) процессор баз данных;
- В) утилита для настройки BDE;
- С) проводником БД;
- Д) драйвером для доступа к удаленным СУБД;
- Е) программой для создания и редактирования таблиц.

15. Визуальный компонент DBGrid есть:

- А) экранное отображение таблицы;
- В) навигационный интерфейс;
- С) графический образ;
- Д) независимый переключатель;
- Е) простой список.

16. Какой метод используется для обновления элемента управления?

- А) SetFocus;
- В) CanFocus;
- С) Refresh;
- Д) Clear;
- Е) PerForm.

17. Режим поиска записи, удовлетворяющей заданному критерию, это:

- А) dsEdit;
- В) dsFilter;

- C) dsSetKey;
- Д) dsPost;
- Е) dsCalcFields.

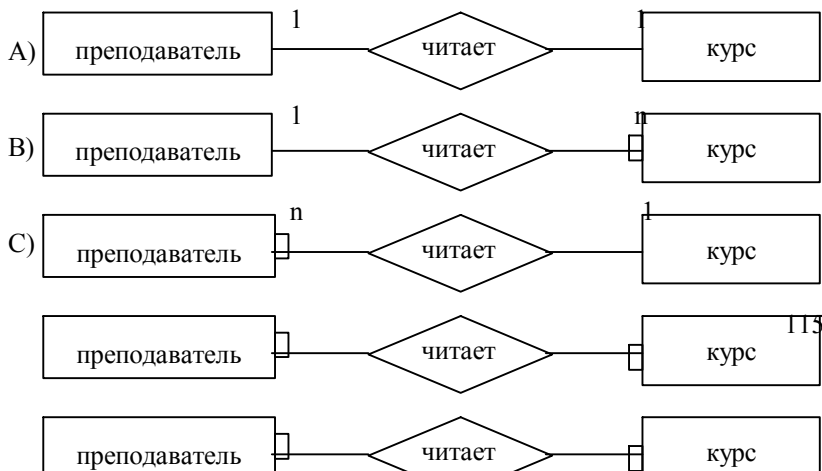
18. Для изменения хранимой процедуры используется оператор:

- A) CREATE PROCEDURE;
- B) ALTER PROCEDURE;
- C) DROP PROCEDURE;
- Д) CREATE VIEW;
- Е) CREATE TABLE.

19. Для изменения записей в таблице используется оператор SQL:

- A) UPDATE;
- B) INSERT;
- C) DELETE;
- Д) CREATE;
- Е) AFTER.

20. ER-диаграмма, представленная ниже, описывает следующую ситуацию: каждый преподаватель обязан читать не менее одного курса, каждый курс должен читаться одним преподавателем.



	n	m
D)		

	M	m
D)		

Правильные ответы тестовых вопросов

	№ вопроса	Правильный ответ
1	10	С
2	15	В
3	20	В
4	25	В
5	30	В
6	35	В
7	40	С
8	45	В
9	50	А
10	55	А
11	60	В
12	65	Д
13	70	С
14	75	Е
15	80	А
16	85	С
17	90	С
18	95	В
19	100	А
20	5	С

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
.....	
ГЛАВА I. МЕТОДОЛОГИЯ И ОБЩИЕ ВОПРОСЫ СОЗДАНИЯ СИСТЕМЫ БАЗЫ ДАННЫХ	7
.....	
1.1. Общая технология создания информационных систем	7
1.2. Общая схема проектирования системы базы данных	9
1.3. Общая схема современных архитектур системы базы дан- ных...	13
.....	
ГЛАВА II. СОВРЕМЕННЫЕ ИНСТРУМЕНТАРИИ СОЗДАНИЯ ПРОГРАММНЫХ СРЕДСТВ И DELPHI	16
.....	
2.1. Назначение IDE Delphi	16
.....	
2.2. Структура и состав IDE Delphi	17
.....	
2.3. Главное окно	17
.....	
2.4. Панель инструментов	18
.....	
2.5. Палитра компонентов	18
.....	
2.6. Проектировщик форм	19
2.7. Окно редактора кода	20
2.8. Окно свойств (инспектор объектов)	20
ГЛАВА III. СОЗДАНИЕ СИСТЕМЫ БАЗЫ ДАННЫХ В DELPHI	21
3.1. Выбор задачи автоматизации и ее формулировка	21
.....	
3.2. План работы по созданию СБД	22
3.3. Построение базы данных	22
.....	
3.3.1. Проектирование базы данных	22
.....	
3.3.2. Создание базы данных	25
.....	

Заключение по разделу	41
ГЛАВА IV. СОЗДАНИЕ СУБД В DELPHI	42
4.1. Определение состава СУБД	42
4.2. Проект и план создания компонентов СУБД	43
4.3. Создание СУБД	46
4.3.1. Способы организации доступа экранных форм и отчет- ных форм к базе данных	46
4.3.2. Создание простых экранных форм	48
4.3.3. Создание сложных экранных форм	59
4.3.4. Создание отчетных форм	68
4.3.5. Запуск и проверка работы системы	68
4.4. Вычисляемые поля	69
Заключение по разделу	77
ГЛАВА V. СОЗДАНИЕ ОТЧЕТНЫХ ФОРМ В DELPHI	78
5.1. Компоненты для построения отчетов	78
5.2. Способы создания простейших отчетов	83
5.2.1. Представление заголовков столбцов отчета	86
5.2.2. Оформление заголовка и подвала страницы отчета	87
5.2.3. Представление в отчете вспомогательной и системной информации	87
5.3. Способы группировки данных в отчете	88
5.4. Формирование отчета с использованием фильтра	91
Заключение по разделу	92

ГЛАВА VI. УСОВЕРШЕНСТВОВАНИЕ СИСТЕМЫ В ПЕРИОД ЭКСПЛУАТАЦИИ	93
.....	
6.1. Оформление системы	93
.....	
6.2. Тестирование и доработка приложения	93
6.3. Сопровождение системы	94
Листинг 1. Модуль перевода числа в запись прописью в отчете Writsum.pas	97
.....	
Заключение по разделу	100
.....	
Библиографическое описание	101
ПРИЛОЖЕНИЯ	102
.....	
III. Глоссарий	102
.....	
П2. Тестовые задания для самоконтроля	104

УЧЕБНОЕ ИЗДАНИЕ

Куандыков Абу Абдикадырович

**СОЗДАНИЕ СИСТЕМЫ БАЗЫ ДАННЫХ
В СРЕДЕ DELPHI**

Учебное пособие

Зав. РО	<i>З.А. Губайдулина</i>
Редактор	<i>Г.М. Дюсенбаева</i>
Техн. редактор	<i>Ж. Н. Байменова</i>
Компьютерный набор и верстка	<i>А.Н. Оразалиева</i>

Подписано в печать 2007 г.

Тираж 500 экз. Формат 60х84. Бумага типографская №1.

Объем 7,0 уч.- изд. л. Заказ №85. Цена договорная

Издание Казахского национального технического университета
им. К.И. Сатпаева НТИЦ КазНТУ, Алматы, ул. Ладыгина, 32