Государственный комитет связи, информатизации и
телекомуникационных технологии Республики Узбекистан
НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ

Кафедра Информационных технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Елмуратова Мадияра студента 4-го курса факультета компьютерный инжинирии по направлению информатики и информационных технологии

ТЕМА: Создание автоматизированной системы учета материальной технической базы организации

Научный руководитель_____к.т.н., Б.Ш.Айтмуратов

Зав. кафедрой:_____ к.т.н., Т.З.Арзымбетов

Н У К У С 2 0 1 4

О Г Л А В Л Е Н И Е

Введение

В настоящее время практически все системы обработки информации в той или иной степени связаны с функциями долговременного хранения и преобразования данных. Фактически качество системы управления базой данных, становится одним из факторов, определяющим эффективность любой сферы деятельности. Увеличились информационные потоки и повысились требования к скорости обработки данных, и теперь уже большинство операций не может быть выполнено вручную, они требуют применения наиболее перспективных компьютерных технологий. И, конечно, обойтись без информационной модели производства в этом случае невозможно. Особенно во многих областях, где сама информация становится предметом производства.

Базы данных являются одним из основных компонентов систем всех уровней и типов. Также и на примере создания сайта мы убедимся в необходимости продуманного создания не только самих таблиц с данными, но и связей между ними, удобного и понятного интерфейса. Здесь от успешности выполнения поставленных задач будет, зависеть насколько прибыльным будет работа всего проекта.

В настоящее время СУБД ACCESS не используется при создании базы данных, но на разобранном примере можно убедиться в том, какие широкие возможности предоставляются при проектировании простых реляционных баз данных, для которых важным критерием является поддержание ссылочной целостности. Кроме того, основные принципы построения этой СУБД можно использовать как платформу для разработки системы с гибкой функциональной интегрированной средой.

Глава I. Анализ предметной области.

В современном обществе то место, которое занимает торговый бизнес невозможно переоценить. Каждый день любой из нас сталкивается с тем, что ему приходиться что-то покупать, или же наоборот, сами продают или оказывает услуги. Постоянно мы вынуждены вступать в товарно-денежные отношения, часто даже не задумываясь о том, что это - наиболее распространенная форма взаимодействия между людьми.

Появление в 90-х гг. XX века электронной коммерции стало возможным благодаря в первую очередь развитию на планете сети Internet. Это, в свою очередь, было причиной возникновения и развития целых отраслей, связанных с обработкой информации. К примеру, деньги в наше время стали в представлении многих людей ассоциироваться не только, да уже и не столько, как бумажки с портретами и памятниками архитектуры, а с виртуальными счетами в Интернет и обычных банках, с всевозможными платежными системами, кредитными картами, картами оплаты, балансами и т. д.

Именно благодаря развитию информационного пространства, вовлечению в него миллионов пользователей, организаций и структур стало возможным появление в начале в США и Европе, а затем и в России и остальном мире. Особенно оказались они востребованы у так называемого среднего класса: людей, отвечающих за хранение и использование материальных баз организации и ценящих при этом своё время, удобство и комфорт. Все меньше в наши дни остается людей готовых тратить время на хождение по складам организации, вывести суммы количество товаров от руки и прочие неудобства. И все больше число тех, кто просто хочет кликнуть мышкой и потом лишь открыть входную дверь, чтобы получить то, что ему необходимо.

Приведение экономических расчетов, выгодной работы учета материальных ценностей организации выходит за рамки данной работы.

Поэтому мы будем заострять внимание в первую очередь на вопросах, связанных с организацией системы управления базой данных нашего проекта, который будет называться «Создание автоматизированной системы учета материальной технической базы организации».

Вся необходимая для работы информация, а точнее все данные будут содержаться в специальных таблицах-отношениях. Все таблицы являются связанными между собой, при этом выполняются все требования, предъявляемые к реляционным базам данных. Клиент при запросе товара может выбирать между различными категориями (с разными сроками хранениями и платой), а также между различными формами оплаты. Данный проект упростит внесение и изменение всех персональных данных, а также не допустит появление противоречивой информации и различных аномалий. Он сделает процесс учета товаров для организации максимально удобным и понятным, а обслуживание для администраторов простым и единственно правильным.

1.1 Постановка задачи

Целью выполнения работы является разработка информационного и программного обеспечения предметной области, связанной с работой учета и управление материальной базы организации. Программа создана на языке программирование Delphi совместно с СУБД ACCESS. В составе проекта реализованы функции учета поставленных различными поставщиками товаров, учета клиентской базы, а также предоставление других широких возможностей для упрощения процедур учета товаров, оплаты за приобретенные товары. Основными функциями программы являются:

- Нахождение наименование (цены, количество и т.д.) товара;
- Возможность введения (изменения) категорий товаров;
- Печать статистики по выбранным группам товаров.

1.2. Основы работы с базами данных

Под базой данных понимается некоторая унифицированная совокупность данных, совместно используемая персоналом/населением группы, предприятия, региона, страны, мира. Задача базы данных состоит в хранении всех представляющих интерес данных в одном или нескольких местах, причем таким способом, который заведомо исключает ненужную избыточность. В хорошо спроектированной базе данных избыточность данных исключается, и вероятность сохранения противоречивых данных минимизируется. Таким образом, создание баз данных преследует две основные цели: понизить избыточность данных и повысить их надежность.

В данном разделе рассмотрим процесс проектирования баз данных, общий для обеих технологий. И лишь детали его реализации будут различаться в разных архитектурах. Сразу оговоримся, что мы будем рассматривать только реляционные базы данных: во-первых, реляционные базы получили наибольшее распространение в мире; во-вторых, они наиболее “продвинуты” в научном плане; а в-третьих, ядро баз данных ***Borland Database Engine***, на основе которого работают все последние продукты компании Borland, предназначено именно для работы с реляционными базами данных.

Жизненный цикл любого программного продукта, в том числе и системы управления базой данных, состоит (по-крупному) из стадий *проектирования*, *реализации и эксплуатации*.

Естественно, наиболее значительным фактором в жизненном цикле приложения, работающего с базой данных, является стадия проектирования. От того, насколько тщательно продумана структура базы, насколько четко

определены связи между ее элементами, зависит производительность системы и ее информационная насыщенность, а значит - и время ее жизни.

Требования к базам данных

хорошо спроектированная база данных:

- Удовлетворяет всем требованиям пользователей к содержимому базы данных. Перед проектированием базы необходимо провести обширные исследования требований пользователей к функционированию базы данных.
- Гарантирует непротиворечивость и целостность данных. При проектировании таблиц нужно определить их атрибуты и некоторые правила, ограничивающие возможность ввода пользователем неверных значений. Для верификации данных перед непосредственной записью их в таблицу база данных должна осуществлять вызов правил модели данных и тем самым гарантировать сохранение целостности информации.
- Обеспечивает естественное, легкое для восприятия структурирование информации. Качественное построение базы позволяет делать запросы к базе более “прозрачными” и легкими для понимания; следовательно, снижается вероятность внесения некорректных данных и улучшается качество сопровождения базы.
- Удовлетворяет требованиям пользователей к производительности базы данных. При больших объемах информации вопросы сохранения производительности начинают играть главную роль, сразу “высвечивая” все недочеты этапа проектирования.

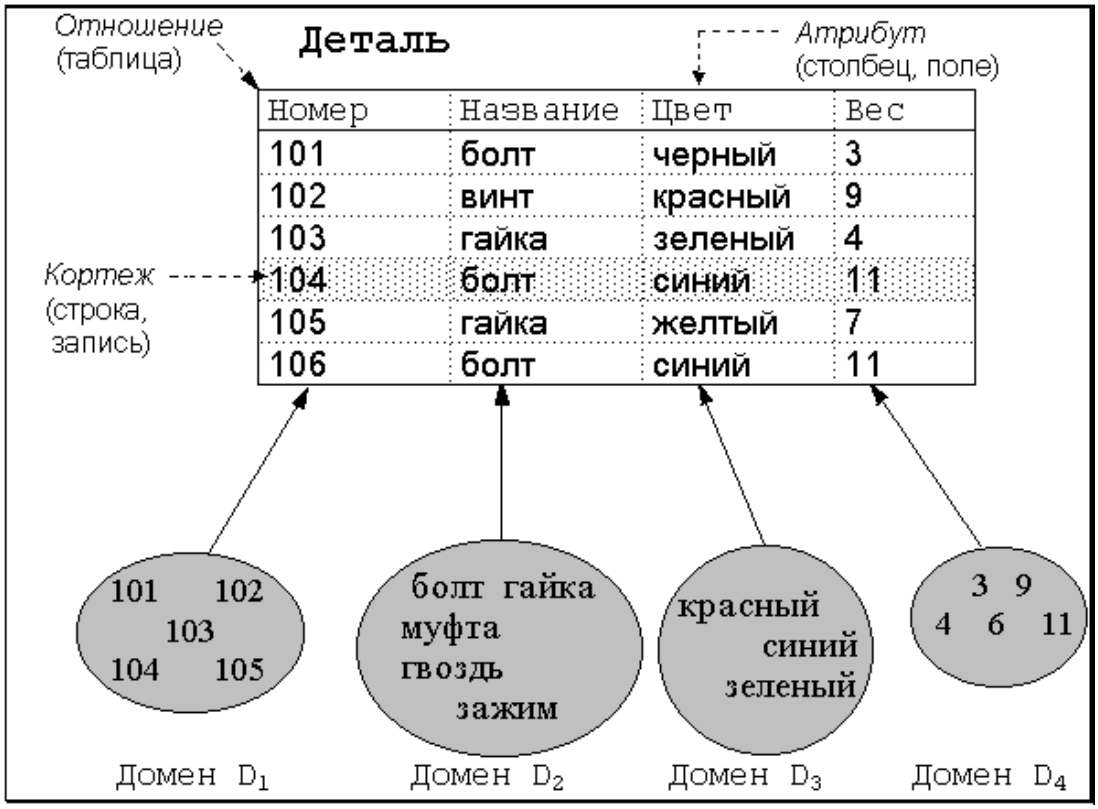
Следующие пункты представляют основные шаги проектирования базы данных:

- Определить информационные потребности базы данных.
- Проанализировать объекты реального мира, которые необходимо смоделировать в базе данных.

- Сформировать из этих объектов сущности и характеристики этих сущностей (например, для сущности “*деталь*” характеристиками могут быть “*название*”, “*цвет*”, “*вес” и т.п.) и сформировать их список.*

Поставить в соответствие сущностям и характеристикам - таблицы и столбцы (поля) в нотации выбранной Вами СУБД (Paradox, dBase, FoxPro, Access, Clipper, InterBase, Sybase, Informix, Oracle и т.д.). Определить атрибуты, которые уникальным образом идентифицируют каждый объект. Выработать правила, которые будут устанавливать, и поддерживать целостность данных. Установить связи между объектами (таблицами и столбцами), провести нормализацию таблиц. Спланировать вопросы надежности данных и, при необходимости, сохранения секретности информации.

Основные концепции реляционных баз данных



Прежде чем подробно рассматривать каждый из этих шагов, остановимся на основных концепциях реляционных баз данных. В реляционной теории одним из главных является понятие **отношения**. Математически отношение определяется следующим образом. Пусть даны n множеств $D_1,D_2,...,D_n$. Тогда R есть отношение над этими множествами, если R есть множество упорядоченных наборов вида $<d_1,d_2,...,d_n>$, где d_1 - элемент из D_1, d_2 - элемент из D_2, ..., d_n - элемент из D_n. При этом наборы вида $<d_1,d_2,...,d_n>$ называются кортежами, а множества $D_1,D_2,...,D_n$ - доменами. Каждый кортеж состоит из элементов, выбираемых из своих доменов. Эти элементы называются атрибутами, а их значения - значениями атрибутов.

Легко заметить, что отношение является отражением некоторой сущности реального мира (в данном случае - сущности “деталь”) и с точки зрения обработки данных представляет собой таблицу. Поскольку в локальных базах данных каждая таблица размещается в отдельном файле, то с точки зрения размещения данных для локальных баз данных отношение можно отождествлять с файлом. Кортеж представляет собой строку в таблице, или, что то же самое, запись. Атрибут же является столбцом таблицы, или - полем в записи. Домен же представляется неким обобщенным типом, который может быть источником для типов полей в записи. Таким образом, следующие тройки терминов являются эквивалентными:

- отношение, таблица, файл (для локальных баз данных)
- кортеж, строка, запись
- атрибут, столбец, поле.

Реляционная база данных представляет собой совокупность отношений, содержащих всю необходимую информацию и объединенных различными связями.

Атрибут (или набор атрибутов), который может быть использован для однозначной идентификации конкретного кортежа (строки, записи),

называется первичным ключом. Первичный ключ не должен иметь дополнительных атрибутов. Это значит, что если из первичного ключа исключить произвольный атрибут, оставшихся атрибутов будет недостаточно для однозначной идентификации отдельных кортежей. Для ускорения доступа по первичному ключу во всех системах управления базами данных (СУБД) имеется механизм, называемый *индексированием*. Грубо говоря, индекс представляет собой инвертированный древовидный список, указывающий на истинное местоположение записи для каждого первичного ключа. Естественно, в разных СУБД индексы реализованы по-разному (в локальных СУБД - как правило, в виде отдельных файлов), однако, принципы их организации одинаковы.

Возможно индексирование отношения с использованием атрибутов, отличных от первичного ключа. Данный тип индекса называется вторичным индексом и применяется в целях уменьшения времени доступа при нахождении данных в отношении, а также для сортировки. Таким образом, если само отношение не упорядочено каким-либо образом и в нем могут присутствовать строки, оставшиеся после удаления некоторых кортежей, то индекс (для локальных СУБД - индексный файл), напротив, отсортирован.

Для поддержания ссылочной целостности данных во многих СУБД имеется механизм так называемых внешних ключей. Смысл этого механизма состоит в том, что некоему атрибуту (или группе атрибутов) одного отношения назначается ссылка на первичный ключ другого отношения; тем самым закрепляются связи подчиненности между этими отношениями. При этом отношение, на первичный ключ которого ссылается внешний ключ другого отношения, называется *master-отношением*, или главным отношением; а отношение, от которого исходит ссылка, называется *detail-отношением*, или подчиненным отношением. После назначения такой ссылки СУБД имеет возможность автоматически отслеживать вопросы “не нарушения“ связей между отношениями, а именно:

- если Вы попытаетесь вставить в подчиненную таблицу запись, для внешнего ключа которой не существует соответствия в главной таблице (например, там нет еще записи с таким первичным ключом), СУБД сгенерирует ошибку;
- если Вы попытаетесь удалить из главной таблицы запись, на первичный ключ которой имеется хотя бы одна ссылка из подчиненной таблицы, СУБД также сгенерирует ошибку.
- если Вы попытаетесь изменить первичный ключ записи главной таблицы, на которую имеется хотя бы одна ссылка из подчиненной таблицы, СУБД также сгенерирует ошибку.

Запретить удаление всех записей, а также изменение первичных ключей главной таблицы, на которые имеются ссылки подчиненной таблицы.

Распространить всякие изменения в первичном ключе главной таблицы на подчиненную таблицу, а именно:

- если в главной таблице удалена запись, то в подчиненной таблице должны быть удалены все записи, ссылающиеся на удаляемую;
- если в главной таблице изменен первичный ключ записи, то в подчиненной таблице должны быть изменены все внешние ключи записей, ссылающихся на изменяемую.

Итак, после того как мы ознакомились с основными понятиями реляционной теории, можно перейти к детальному рассмотрению шагов проектирования базы данных.

Шаги проектирования базы данных

I. Первый шаг состоит в определении информационных потребностей базы данных. Он включает в себя опрос будущих пользователей для того, чтобы понять и задокументировать их требования. Следует выяснить следующие вопросы:

• сможет ли новая система объединить существующие приложения или их необходимо будет кардинально переделывать для совместной работы с новой системой;

• какие данные используются разными приложениями; смогут ли Ваши приложения совместно использовать какие-либо из этих данных;

• кто будет вводить данные в базу, и в какой форме; как часто будут изменяться данные;

• достаточно ли будет для Вашей предметной области одной базы или Вам потребуется несколько баз данных с различными структурами;

• какая информация является наиболее чувствительной к скорости ее извлечения и изменения.

II. Следующий шаг включает в себя анализ объектов реального мира, которые необходимо смоделировать в базе данных.

Формирование концептуальной модели базы данных включает в себя:

• идентификацию функциональной деятельности Вашей предметной области. Например, если речь идет о деятельности предприятия, то в качестве функциональной деятельности можно идентифицировать ведение учета работающих, отгрузку продукции, оформление заказов и т.п.

• идентификацию объектов, которые осуществляют эту функциональную деятельность, и формирование из их операций последовательности событий, которые помогут Вам идентифицировать все сущности и взаимосвязи между ними. Например, процесс “ведение учета работающих” идентифицирует такие сущности как РАБОТНИК, ПРОФЕССИЯ, ОТДЕЛ.

• идентификацию характеристик этих сущностей. Например, сущность РАБОТНИК может включать такие характеристики как Идентификатор Работника, Фамилия, Имя, Отчество, Профессия, Зарплата.

• идентификацию взаимосвязей между сущностями. Например, каким образом сущности РАБОТНИК, ПРОФЕССИЯ, ОТДЕЛ взаимодействуют

друг с другом? Работник имеет одну профессию и значится в одном отделе, в то время как в одном отделе может находиться много работников.

III. Третий шаг заключается в установлении соответствия между сущностями и характеристиками предметной области и отношениями и атрибутами в нотации выбранной СУБД. Поскольку каждая сущность реального мира обладает некими характеристиками, в совокупности образующими полную картину ее проявления, можно поставить им в соответствие набор отношений (таблиц) и их атрибутов (полей).

Перечислив все отношения и их атрибуты, уже на этом этапе можно начать устранять излишние позиции. Каждый атрибут должен появляться только один раз; и Вы должны решить, какое отношение будет являться владельцем какого набора атрибутов.

IV. На четвертом шаге определяются атрибуты, которые уникальным образом идентифицируют каждый объект. Это необходимо для того, чтобы система могла получить любую единичную строку таблицы. Вы должны определить первичный ключ для каждого из отношений. Если нет возможности идентифицировать кортеж с помощью одного атрибута, то первичный ключ нужно сделать составным - из нескольких атрибутов. Хорошим примером может быть первичный ключ в таблице работников, состоящий из фамилии, имени и отчества. Первичный ключ гарантирует, что в таблице не будет содержаться двух одинаковых строк. Во многих СУБД имеется возможность помимо первичного определять еще ряд уникальных ключей. Отличие уникального ключа от первичного состоит в том, что уникальный ключ не является главным идентифицирующим фактором записи и на него не может ссылаться внешний ключ другой таблицы. Его главная задача - гарантировать уникальность значения поля.

V. Пятый шаг предполагает выработку правил, которые будут устанавливать, и поддерживать целостность данных. Будучи определенными,

такие правила в клиент-серверных СУБД поддерживаются автоматически - сервером баз данных; в локальных же СУБД их поддержание приходится возлагать на пользовательское приложение.

Эти правила включают:

- определение типа данных
- выбор набора символов, соответствующего данной стране
- создание полей, опирающихся на домены
- установка значений по умолчанию
- определение ограничений целостности
- определение проверочных условий.

VI. На шестом шаге устанавливаются связи между объектами (таблицами и столбцами) и производится очень важная операция для исключения избыточности данных - нормализация таблиц.

Каждый из различных типов связей должен быть смоделирован в базе данных. Существует несколько типов связей:

- связь *“один-к-одному”*
- связь *“один-ко-многим”*
- связь *“многие-ко-многим”*.

Связь “один-к-одному” представляет собой простейший вид связи данных, когда первичный ключ таблицы является в то же время внешним ключом, ссылающимся на первичный ключ другой таблицы. Такую связь бывает удобно устанавливать тогда, когда невыгодно держать разные по размеру (или по другим критериям) данные в одной таблице. Например, можно выделить данные с подробным описанием изделия в отдельную таблицу с установлением связи “один-к-одному” для того чтобы не занимать оперативную память, если эти данные используются сравнительно редко.

Связь “один-ко-многим” в большинстве случаев отражает реальную взаимосвязь сущностей в предметной области. Она реализуется уже описанной парой “внешний ключ-первичный ключ”, т.е. когда определен внешний ключ, ссылающийся на первичный ключ другой таблицы. Именно эта связь описывает широко распространенный механизм классификаторов. Имеется справочная таблица, содержащая названия, имена и т.п. и некие коды, причем, первичным ключом является код. В таблице, собирающей информацию - назовем ее информационной таблицей - определяется внешний ключ, ссылающийся на первичный ключ классификатора. После этого в нее заносится не название из классификатора, а код. Такая система становится устойчивой от изменения названия в классификаторах. Имеются способы быстрой “подмены” в отображаемой таблице кодов на их названия как на уровне сервера БД (для клиент-серверных СУБД), так и на уровне пользовательского приложения.

Связь “многие-ко-многим” в явном виде в реляционных базах данных не поддерживается. Однако имеется ряд способов косвенной реализации такой связи, которые с успехом возмещают ее отсутствие. Один из наиболее распространенных способов заключается во введении дополнительной таблицы, строки которой состоят из внешних ключей, ссылающихся на первичные ключи двух таблиц. Например, имеются две таблицы: КЛИЕНТ и ГРУППА_ИНТЕРЕСОВ. Один человек может быть включен в различные группы, в то время как группа может объединять различных людей. Для реализации такой связи “многие-ко-многим” вводится дополнительная таблица, назовем, ее КЛИЕНТЫ_В_ГРУППЕ, строка, которая будет иметь два внешних ключа: один будет ссылаться на первичный ключ в таблице КЛИЕНТ, а другой - на первичный ключ в таблице ГРУППА_ИНТЕРЕСОВ. Таким образом, в таблицу КЛИЕНТЫ_В_ГРУППЕ можно записывать любое количество людей и любое количество групп.

Итак, после определения таблиц, полей, индексов и связей между таблицами следует посмотреть на проектируемую базу данных в целом и проанализировать ее, используя правила нормализации, с целью устранения логических ошибок. Важность нормализации состоит в том, что она позволяет разбить большие отношения, как правило, содержащие большую избыточность информации, на более мелкие логические единицы, группирующие только данные, объединенные “по природе”. Таким образом, идея нормализации заключается в следующем. Каждая таблица в реляционной базе данных удовлетворяет условию, в соответствии с которым в позиции на пересечении каждой строки и столбца таблицы всегда находится единственное значение, и никогда не может быть множества таких значений.

После применения правил нормализации логические группы данных располагаются не более чем в одной таблице. Это дает следующие преимущества:

- данные легко обновлять или удалять
- исключается возможность рассогласования копий данных
- уменьшается возможность введения некорректных данных.

Процесс нормализации заключается в приведении таблиц в так называемые *нормальные формы*. Существует несколько видов нормальных форм: первая нормальная форма (1НФ), вторая нормальная форма (2НФ), третья нормальная форма (3НФ), нормальная форма Бойса-Кодда (НФБК), четвертая нормальная форма (4НФ), пятая нормальная форма (5НФ). С практической точки зрения, достаточно трех первых форм - следует учитывать время, необходимое системе для “соединения” таблиц при отображении их на экране. Поэтому мы ограничимся изучением процесса приведения отношений к первым трем формам.

Этот процесс включает:

- устранение повторяющихся групп (приведение к 1НФ)
- удаление частично зависимых атрибутов (приведение к 2НФ)
- удаление транзитивно зависимых атрибутов (приведение к 3НФ).

Рассмотрим каждый из этих процессов подробней.

Приведение к первой нормальной форме

Когда поле в данной записи содержит более одного значения для каждого вхождения первичного ключа, такие группы данных называются *повторяющимися группами*. 1НФ не допускает наличия таких многозначных полей. Рассмотрим пример базы данных предприятия, содержащей таблицу ОТДЕЛ со следующими значениями (атрибут, выделенный курсивом, является первичным ключом):

Табл. А: ОТДЕЛ

Номер_отдела	**Название**	**Руководитель**	**Бюджет**	**Расположение**
100	продаж	000	1000000	Москва
100	продаж	000	1000000	Зеленоград
600	разработок	120	1100000	Тверь
100	продаж	000	1000000	Калуга

Для приведения этой таблицы к 1НФ мы должны устранить атрибут (поле) Расположение из таблицы ОТДЕЛ и создать новую таблицу РАСПОЛОЖЕНИЕ_ОТДЕЛОВ, в которой следует определить первичный ключ, являющийся комбинацией номера отдела и его расположения (Номер_отдела+Расположение - см. табл. b). Теперь для каждого расположения отдела существуют различные строки; тем самым мы

устранили повторяющиеся группы.

Табл. B: РАСПОЛОЖЕНИЕ_ОТДЕЛОВ

Номер_отдела	***Расположение***
100	Москва
100	Зеленоград
600	Тверь
100	Калуга

Приведение ко второй нормальной форме

Следующий важный шаг в процессе нормализации состоит в удалении всех не ключевых атрибутов, которые зависят только от части первичного ключа. Такие атрибуты называются *частично зависимыми*. Не ключевые атрибуты заключают в себе информацию о данной сущности предметной области, но не идентифицируют ее уникальным образом. Например, предположим, что мы хотим распределить работников по проектам, ведущимся на предприятии. Для этого создадим таблицу ПРОЕКТ с составным первичным ключом, включающим номер работника и идентификатор проекта (Номер_работника+ИД_проекта, в табл. с выделенным курсивом).

Табл. C: ПРОЕКТ

Номер_ работника	***ИД_ проекта***	**Фамилия**	**Назв_ проекта**	**Описание_ проекта**	**Продукт**
28	БРЖ	Иванов	Биржа	<blob>	программа
17	ДОК	Петров	Документы	<blob>	программа
06	УПР	Сидоров	Управление	<blob>	адм.меры

В этой таблице возникает следующая проблема. Атрибуты Назв_проекта, Описание_проекта и Продукт относятся к проекту как сущности и, следовательно, зависят от атрибута ИД_проекта (являющегося

частью первичного ключа), но не от атрибута Номер_работника. Следовательно, они являются частично зависимыми от составного первичного ключа. То же самое можно сказать и об атрибуте Фамилия, который зависит от атрибута Номер_работника, но не зависит от атрибута ИД_проекта. Для нормализации этой таблицы (приведения ее в 2НФ) удалим из нее атрибуты Номер_работника и Фамилия и создадим другую таблицу (назовем, ее РАБОТНИК_В_ПРОЕКТЕ), которая будет содержать только эти два атрибута, и они же будут составлять ее первичный ключ.

Приведение к третьей нормальной форме

Третий этап процесса приведения таблиц к нормальной форме состоит в удалении всех не ключевых атрибутов, которые зависят от других не ключевых атрибутов. Каждый не ключевой атрибут должен быть логически связан с атрибутом (атрибутами), являющимся первичным ключом. Предположим, например, что мы добавили поля Номер_руководителя и Телефон в таблицу ПРОЕКТ, находящуюся в 2НФ (первичным ключом является поле ИД_проекта). Атрибут Телефон логически связан с атрибутом Номер_руководителя, не ключевым полем, но не с атрибутом ИД_проекта, являющимся первичным ключом (табл. d).

Табл. D: ПРОЕКТ

ИД_проекта	**Номер_ руководителя**	**Телефон**	**Назв_ проекта**	**Описание_ проекта**	**Продукт**
БРЖ	02	2-21	Биржа	<blob>	программа
ДОК	12	2-43	Документы	<blob>	программа
УПР	08	2-56	Управление	<blob>	адм.меры

Для нормализации этой таблицы (приведения ее в 3НФ) удалим атрибут Телефон, изменим Номер_руководителя на Руководитель и сделаем атрибут Руководитель внешним ключом, ссылающимся на атрибут Номер_работника

(первичный ключ) в таблице РАБОТНИКИ. После этого таблицы ПРОЕКТ и РАБОТНИКИ будут выглядеть следующим образом:

Табл. E: ПРОЕКТ

ИД_проекта	**Руководитель**	**Назв_ проекта**	**Описание_ проекта**	**Продукт**
БРЖ	02	Биржа	<blob>	программа
ДОК	12	Документы	<blob>	программа
УПР	08	Управление	<blob>	адм.меры

Табл. F: РАБОТНИКИ

Номер_ работника	**Фамилия**	**Имя**	**Отчество**	**Номер_ отдела**	**Код_ профес**	**Тел.**	**Зарплата**
04	Иванов	Иван	Иванович	100	инж	2-69	500
08	Петров	Петр	Петрович	200	мндж	2-56	1000
23	Сидоров	Иван	Петрович	200	мндж	2-45	800

Теперь, когда мы научились проводить нормализацию таблиц с целью устранения избыточного дублирования данных и группирования информации в логически связанных единицах, необходимо сделать ряд замечаний по вопросам проектирования баз данных. Необходимо четко понимать, что разбиение информации на более мелкие единицы с одной стороны, способствует повышению надежности и непротиворечивости базы данных, а с другой стороны, снижает ее производительность, так как требуются дополнительные затраты процессорного времени (серверного или машины пользователя) на обратное “соединение” таблиц при представлении

информации на экране. Иногда для достижения требуемой производительности нужно сделать отход от канонической нормализации, при этом ясно осознавая, что необходимо обеспечить меры по предотвращению противоречивости в данных. Поэтому всякое решение о необходимости того или иного действия по нормализации можно принимать только тщательно проанализировав предметную область и класс поставленной задачи. Может потребоваться несколько итераций для достижения состояния, которое будет желаемым компромиссом между четкостью представления и реальными возможностями техники. Здесь уже начинается искусство.

VII. Седьмой шаг является последним в нашем списке, но не последним по важности в процессе проектирования базы данных. На этом шаге мы должны спланировать вопросы надежности данных и, при необходимости, сохранения секретности информации. Для этого необходимо ответить на следующие вопросы:

- кто будет иметь права (и какие) на использование базы данных
- кто будет иметь права на модификацию, вставку и удаление данных
- нужно ли делать различие в правах доступа
- каким образом обеспечить общий режим защиты информации и т.п.

Вывод по главе I.

В данной главе мы познакомились с основами теории реляционных баз данных, изучили требования к базам данных, а также основные шаги по проектированию баз данных. Кроме того, рассмотрен очень важный для проектирования баз данных вопрос нормализации таблиц и проблемы, связанные с этим процессом. Теперь мы можем осознанно приступать к созданию приложений, работающих с базами данных.

Глава II. Среда программирование

2.1. Среда программирования Delphi

Delphi это высокопроизводительный инструмент создания приложений. Текущая версия является 16-разрядным компилятором для создания программ, работающих в среде Windows Delphi 7.0, которая появился в начале 2000 годах, будет включать полный 32-разрядный компилятор для использования в Windows 95 или в Windows NT.

Для запуска Delphi требуется 386DX компьютер с 4MB памяти. Более подходящей машиной будет 486DX 66MHz с 8MB ОЗУ.

Небольшие программы, созданные на Delphi, будут работать на любом компьютере. Другими словами, они не требуют того ОЗУ или скорости процессора, что необходимо для среды Delphi.

Структура среды программирования

Внешний вид среды программирования Delphi отличается от многих других из тех, что можно увидеть в Windows. К примеру, Borland Pascal for Windows 7.0, Borland C++ 4.0, Word for Windows, Program Manager - это все MDI приложения и выглядят по-другому, чем Delphi. MDI (Multiple Document Interface) - определяет особый способ управления нескольких дочерних окон внутри одного большого окна.

Среда Delphi же следует другой спецификации, называемой Single Document Interface (SDI), и состоит из нескольких отдельно расположенных окон. Это было сделано из-за того, что SDI близок к той модели приложений, что используется в Windows 95.

Если Вы используете SDI приложение типа Delphi, то уже знаете, что перед началом работы лучше минимизировать другие приложения, чтобы их окна не загромождали рабочее пространство. Если нужно переключиться на другое приложение, то просто щелкните мышкой на системную кнопку

минимизации Delphi. Вместе с главным окном свернутся все остальные окна среды программирования, освободив место для работы других программ.

Главные составные части среды программирования

Ниже перечислены основные составные части Delphi:

Дизайнер Форм (Form Designer)

Окно Редактора Исходного Текста (Editor Window)

Палитра Компонент (Component Palette)

Инспектор Объектов (Object Inspector)

Справочник (On-line help)

Программисты на Delphi проводят большинство времени, переключаясь между Дизайнером Форм и Окном Редактора Исходного Текста (которое для краткости называют Редактор). Дизайнер Форм показан на рис.2.1.1, окно Редактора - на рис.2.1.2.

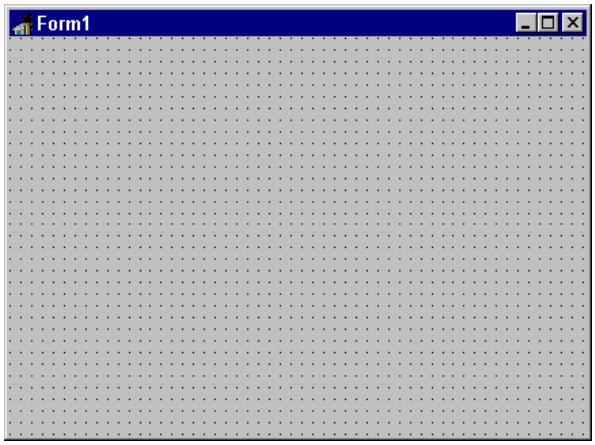


Рис. 2.1.1 Дизайнер Форм

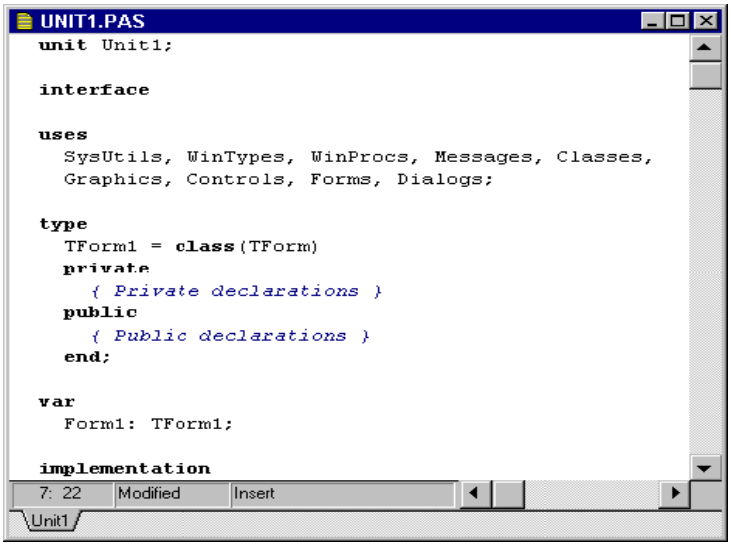


Рис. 2.1.2 Окно редактора

Дизайнер Форм в Delphi столь интуитивно понятен и прост в использовании. Дизайнер Форм первоначально состоит из одного пустого окна, которое Вы заполняете всевозможными объектами, выбранными на Палитре Компонент.

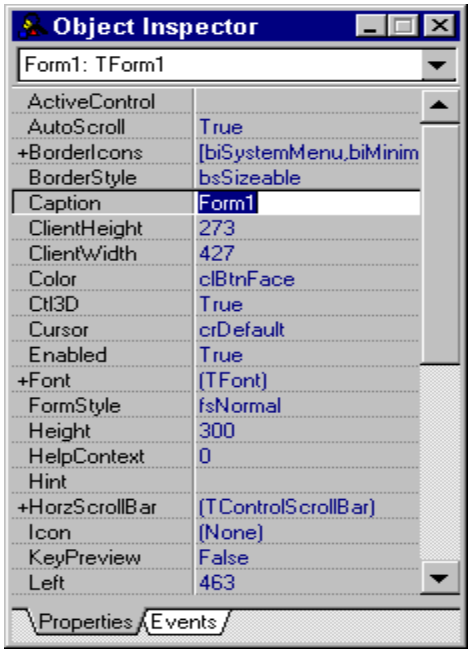
Несмотря на всю важность Дизайнера Форм, местом, где программисты проводят основное время, является Редактор. Логика является движущей силой программы и Редактор - то место, где Вы ее “кодируете”.

Палитра Компонент (см. рис.2.1.3) позволяет выбрать нужные объекты для размещения их на Дизайнере Форм. Для использования Палитры Компонент просто первый раз щелкните мышкой на один из объектов, и потом второй раз - на Дизайнере Форм.

Палитра Компонент использует постраничную группировку объектов. Внизу Палитры находится набор закладок - Standard, Additional, Dialogs и т.д. Если Вы щелкнете мышью на одну из закладок, то Вы можете перейти на следующую страницу Палитры Компонент. Принцип разбиения на страницы широко используется в среде программирования Delphi и его легко можно использовать в своей программе

Рис.2.1.3: Палитра Компонент

Слева от Дизайнера Форм Вы можете видеть Инспектор Объектов (рис.2.1.4). Заметьте, что информация в Инспекторе Объектов меняется в зависимости от объекта, выбранного на форме. Важно понять, что каждый компонент является настоящим объектом, и Вы можете менять его вид и поведение с помощью Инспектора Объектов.



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.

Можете использовать закладки внизу Инспектора Объектов для переключения между страницами свойств и событий. Страница событий связана с Редактором.

Последняя важная часть среды Delphi - Справочник (on-line help). Для доступа к этому инструменту нужно просто выбрать в системном меню пункт Help и затем Contents. На экране появится Справочник, показанный на рис.2.1.5

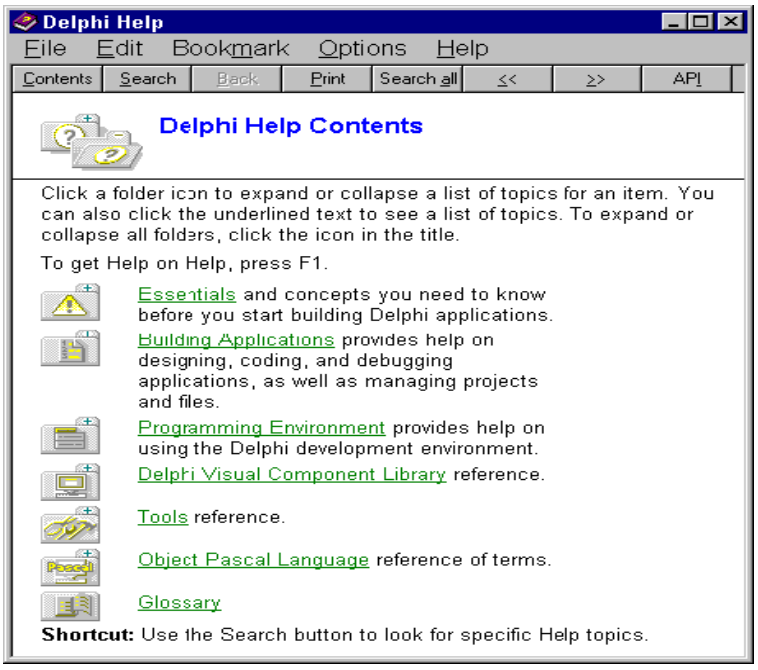


Рис.2.1.5: Справочник - быстрый поиск любой информации.

Справочник является контекстно-зависимым; при нажатии клавиши F1, Вы получите подсказку, соответствующую текущей ситуации.. Если в любой момент работы в среде Delphi возникает неясность или затруднение - жмите F1 и необходимая информация появится на экране.

Дополнительные элементы

В данном разделе внимание фокусируется на трех инструментах, которые можно воспринимать как вспомогательные для среды программирования:

Меню (Menu System)

Панель с кнопками для быстрого доступа (SpeedBar)

Редактор картинок (Image Editor)

Меню предоставляет быстрый и гибкий интерфейс к среде Delphi, потому что может управляться по набору “горячих клавиш”. Это удобно еще и потому, что здесь используются слова или короткие фразы, более точные и понятные, нежели иконки или пиктограммы.

SpeedBar находится непосредственно под меню, слева от Палитры Компонент (рис.2.16). SpeedBar выполняет много из того, что можно сделать через меню. Если задержать мышь над любой из иконок на SpeedBar, то Вы увидите, что появится подсказка, объясняющая назначение данной иконки.

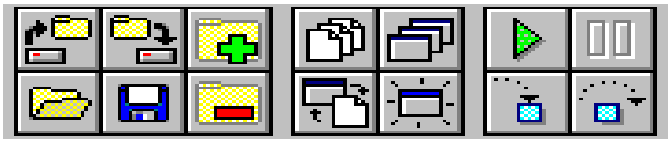


Рис.2.1.6: SpeedBar находится слева от Палитры Компонент.

Редактор Картинок, показанный на рис.2.1.7, работает аналогично программе Paintbrush из Windows. Вы можете получить доступ, к этому модулю выбрав пункт меню Tools | Image Editor.

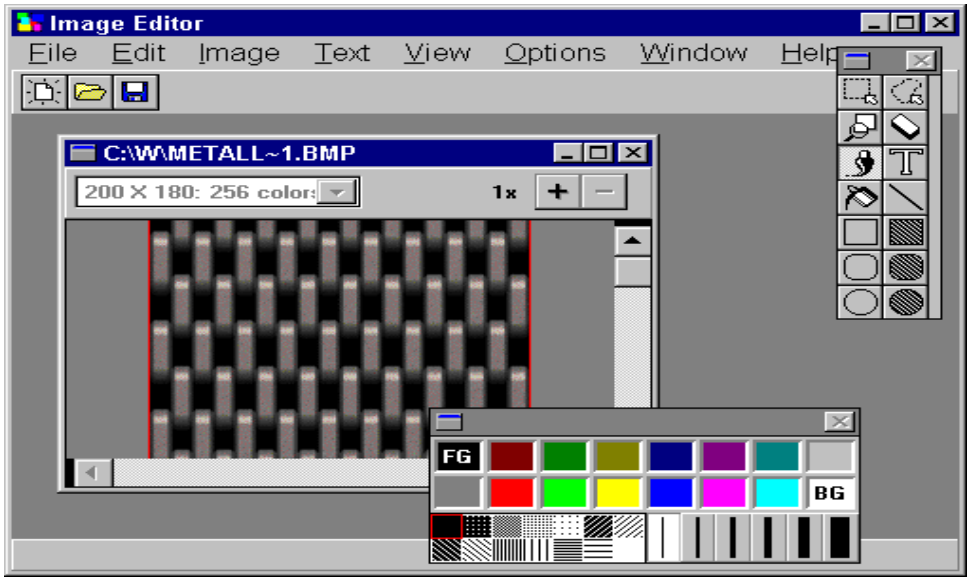


Рис.2.1.7: Редактор Картинок можно использовать для создания картинок для кнопок, иконок и др.

Стандартные компоненты

На первой странице Палитры Компонент размещены 14 объектов (рис.2.1.8) определенно важных для использования. Мало кто обойдется

длительное время без кнопок, списков, окон ввода и т.д. Все эти объекты такая же часть Windows, как мышь или окно.

Набор и порядок компонент на каждой странице являются конфигурируемыми. Так, Вы можете добавить к имеющимся компонентам новые, изменить их количество и порядок.

Рис.2.1.8: Компоненты, расположенные на первой странице Палитры.

· TMainMenu позволяет Вам поместить главное меню в программу. При помещении TMainMenu на форму это выглядит, как просто иконка. Иконки данного типа называют "невидимыми компонентом", поскольку они невидимы во время выполнения программы. Создание меню включает три шага: (1) помещение TMainMenu на форму, (2) вызов Дизайнера Меню через свойство Items в Инспекторе Объектов, (3) определение пунктов меню в Дизайнере Меню.

·TPopupMenu позволяет создавать всплывающие меню. Этот тип меню появляется по щелчку правой кнопки мыши.

· TLabel служит для отображения текста на экране. Вы можете изменить шрифт и цвет метки, если дважды щелкнете на свойство Font в Инспекторе Объектов. Вы увидите, что это легко сделать и во время выполнения программы, написав всего одну строчку кода.

· TEdit - стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.

· TMemo - иная форма TEdit. Подразумевает работу с большими текстами. TMemo может переносить слова, сохранять в Clipboard фрагменты текста и восстанавливать их, и другие основные функции редактора. TMemo имеет ограничения на объем текста в 32Кб, это составляет 10-20 страниц. (Есть VBX и “родные” компоненты Delphi, где этот предел снят).

· TButton позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы. В Delphi все делается очень просто.

·TCheckBox отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано. Например, если посмотреть окно диалога настроек компилятора (пункт меню Options | Project, страница Compiler), то можно увидеть, что оно состоит преимущественно из CheckBox'ов.

·TRadioButton позволяет выбрать только одну опцию из нескольких. Если Вы опять откроете диалог Options | Project и выберете страницу Linker Options, то Вы можете видеть, что секции Map file и Link buffer file состоят из наборов RadioButton.

·TListBox нужен для показа прокручиваемого списка. Классический пример ListBox'а в среде Windows - выбор файла из списка в пункте меню File | Open многих приложений. Названия файлов или директорий и находятся в ListBox'е.

·TComboBox во многом напоминает ListBox, за исключением того, что позволяет водить информацию в маленьком поле ввода сверху ListBox. Есть несколько типов ComboBox, но наиболее популярен выпадающий вниз (drop-down combo box), который можно видеть внизу окна диалога выбора файла.

·TScrollbar - полоса прокрутки, появляется автоматически в объектах редактирования, ListBox'ах при необходимости прокрутки текста для просмотра.

· TGroupBox используется для визуальных целей и для указания Windows, каков порядок перемещения по компонентам на форме (при нажатии клавиши TAB).

· TPanel - управляющий элемент, похожий на TGroupBox, используется в декоративных целях. Чтобы использовать TPanel, просто поместите его на форму и затем положите другие компоненты на него. Теперь при перемещении TPanel будут передвигаться и эти компоненты. TPanel используется также для создания линейки инструментов и окна статуса.

TScrollBox представляет собой место на форме, которое можно скроллировать в вертикальном и горизонтальном направлениях. Пока Вы в явном виде не отключите эту возможность, форма сама по себе действует так же. Однако, могут быть случаи, когда понадобится прокручивать только часть формы. В таких случаях используется TScrollBox.

Это полный список объектов на первой странице Палитры Компонент. Если Вам нужна дополнительная информация, то выберите на Палитре объект и нажмите клавишу F1 - появится Справочник с полным описанием данного объекта.

Сохранение программы

Вы приложили некоторые усилия по созданию программы и можете захотеть ее сохранить. Это позволит загрузить программу позже и снова с ней поработать.

Первый шаг - создать поддиректорию для программы. Лучше всего создать директорию, где будут храниться все Ваши программы и в ней - создать поддиректорию для данной конкретной программы. Например, Вы можете создать директорию MYCODE и внутри нее - вторую директорию TIPS1, которая содержала бы программу, над которой Вы только что работали.

После создания поддиректории для хранения Вашей программы нужно выбрать пункт меню File | Save Project. Сохранить нужно будет два файла. Первый - модуль (unit), над которым Вы работали, второй - главный файл проекта, который "владеет" Вашей программой. Сохраните модуль под именем MAIN.PAS и проект под именем TIPS1.DPR. (Любой файл с расширением PAS и словом “unit” в начале является модулем.)

TButton, исходный текст, заголовки и Z-упорядочивание

Еще несколько возможностей Инспектора Объектов и Дизайнера Форм.

Создайте новый проект. Поместите на форму объект TMemo, а затем TEdit так, чтобы он наполовину перекрывал TMemo, как показано на рис.2.1.9. Теперь выберите пункт меню Edit | Send to Back, что приведет к

перемещению TEdit вглубь формы, за объект TMemo. Это называется изменением Z-порядка компонент. Буква Z используется потому, что обычно математики обозначают третье измерение буквой Z. Так, X и Y используются для обозначения ширины и высоты, и Z используется для обозначения глубины.

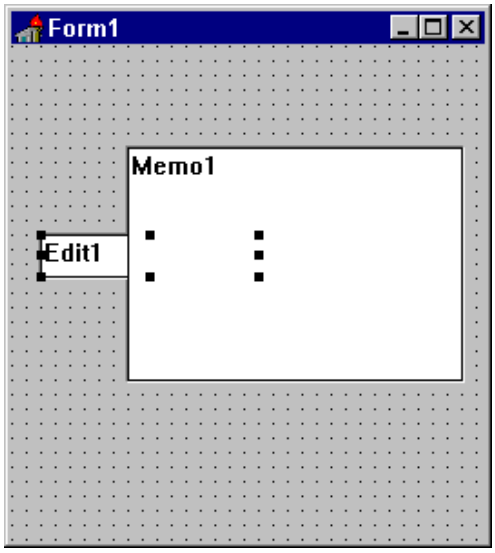


Рис.2.1.9: Объект TEdit перекрывается наполовину объектом TMemo.

Если Вы “потеряли” на форме какой-то объект, то найти его можно в списке Combobox’а, который находится в верхней части Инспектора Объектов.

Поместите кнопку TButton в нижнюю часть формы. Теперь растяните Инспектор Объектов так, чтобы свойства Name и Caption были видны одновременно на экране. Теперь измените имя кнопки на Terminate. Заметьте, что заголовок (Caption) поменялся в тот же момент. Такое двойное изменение наблюдается, только если ранее не изменялось свойство Caption.

Текст, который Вы видите на поверхности кнопки - это содержимое свойства Caption, свойство Name служит для внутренних ссылок, Вы будете использовать его при написании кода программы. Если Вы откроете сейчас окно Редактора, то увидите следующий фрагмент кода:

```
TForm1 = class(TForm)
Edit1: TEdit;
Memo1: TMemo;
```

Terminate: TButton;

private

{ Private declarations }

public

{ Public declarations }

end;

В этом фрагменте кнопка TButton называется Terminate из-за того, что Вы присвоили это название свойству Name. Заметьте, что TMemo имеет имя, которое присваивается по умолчанию.

Перейдите на форму и дважды щелкните мышкой на объект TButton. Вы сразу попадете в окно Редактора, в котором увидите фрагмент кода вроде этого:

procedure TForm1.TerminateClick(Sender: TObject);

begin

end;

Данный код был создан автоматически, и будет выполняться всякий раз, когда во время работы программы пользователь нажмет кнопку Terminate.

Потратьте немного времени на усвоение последовательности действий, описанных выше. Изначально Вы смотрите на кнопку на форме. Вы делаете двойной щелчок на эту кнопку, и соответствующий фрагмент кода автоматически заносится в Редактор.

Теперь самое время написать строчку кода. Это очень простой код, состоящий из одного слова Close:

procedure TForm1.TerminateClick(Sender: TObject);

begin

Close;

end;

Когда этот код исполняется, то главная форма закрывается. Для проверки кода запустите программу и нажмите кнопку Terminate. Если все сделано правильно, программа закроется, и Вы вернетесь в режим дизайна.

Прежде, чем перейти к следующему разделу, перейдите в Инспектор Объектов и измените значение свойства Name для кнопки на любое другое, например OK. Нажмите Enter для внесения изменений. Посмотрите в Редактор, Вы увидите, что код, написанный Вами изменился:

```
procedure TForm1.OkClick(Sender: TObject);
begin Close; end;
```

Заметьте, что аналогичные изменения произошли и в определении класса:

```
TForm1 = class(TForm)
Edit1: TEdit;
Memo1: TMemo;
Ok: TButton;
procedure OkClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
```

2.2 Архитектура приложений баз данных

В Репозитории Delphi отсутствует отдельный шаблон для приложения баз данных. Поэтому, как и любое другое приложение Delphi, приложение баз данных начинается с обычной формы. Безусловно, это оправданный подход, т. к. приложение баз данных имеет пользовательский интерфейс. И этот интерфейс создается с использованием стандартных и специализированных визуальных компонентов на обычных формах.

Визуальные компоненты отображения данных расположены на странице Data Controls Палитры компонентов. В большинстве они представляют собой

модификации стандартных элементов управления, приспособленных для работы с набором данных.

Приложение может содержать произвольное число форм и использовать любой интерфейс (MDI или SDI). Обычно одна форма отвечает за выполнение группы однородных операций, объединенных общим назначением.

В основе любого приложения баз данных лежат наборы данных, которые представляют собой группы записей (их удобно представить в виде таблиц в памяти), переданных из базы данных в приложение для просмотра и редактирования. Каждый набор данных инкапсулирован в специальном компоненте доступа к данным. В VCL Delphi реализован набор базовых классов, поддерживающих функциональность наборов данных, и практически идентичные по составу наборы дочерних компонентов для технологий доступа к данным. Их общий предок — класс TDataSet.

Для обеспечения связи набора данных с визуальными компонентами отображения данных используется специальный компонент TDataSource. Его роль заключается в управлении потоками данных между набором данных и связанными с ним компонентами отображения данных. Этот компонент обеспечивает передачу данных в визуальные компоненты и возврат результатов редактирования в набор данных, отвечает за изменение состояния визуальных компонентов при изменении состояния набора данных, передает сигналы управления от пользователя (визуальных компонентов) в набор данных. Компонент TDataSource расположен на странице Data Access Палитры компонентов.

Таким образом, базовый механизм доступа к данным создается триадой компонентов:

компоненты, инкапсулирующие набор данных (потомки класса TDataSet);

компоненты TDataSource;

визуальные компоненты отображения данных.

Рассмотрим схему взаимодействия этих компонентов в приложении баз данных (рис. 2.2.1).

В приложении с источником данных или промежуточным программным обеспечением взаимодействует компонент доступа к данным, который инкапсулирует набор данных и обращается к функциям соответствующей технологии доступа к данным для выполнения различных операций. Компонент доступа к данным представляет собой "образ" таблицы базы данных в приложении. Общее число таких компонентов в приложении не ограничено.

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

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

С одним компонентом TDataSource могут быть связаны несколько визуальных компонентов отображения данных. Эти компоненты представляют собой модифицированные элементы управления, которые предназначены для показа информации из наборов данных.

При открытии набора данных компонент обеспечивает передачу в набор данных записей из требуемой таблицы БД. Курсор набора данных устанавливается на первую запись. Компонент TDataSource организует передачу в компоненты отображения данных значений необходимых полей

из текущей записи. При перемещении по записям набора данных текущие значения полей в компонентах отображения данных автоматически обновляются.

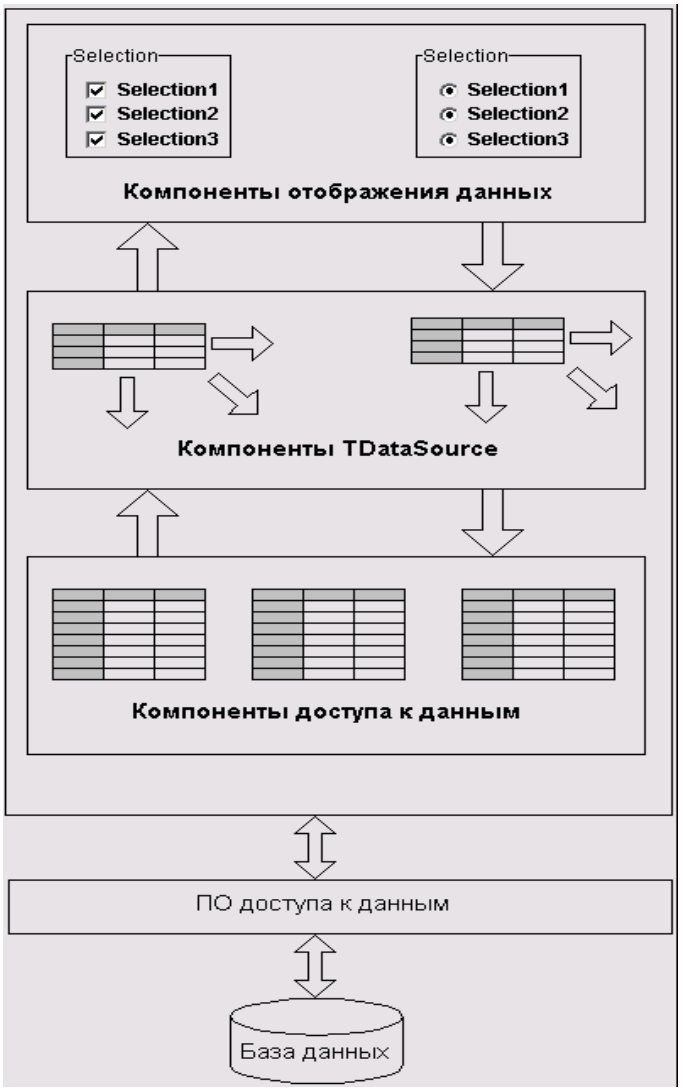


Рис. 2.2.1. Механизм доступа к данным приложения баз данных

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

Модуль данных

Для размещения компонентов доступа к данным в приложении баз данных желательно использовать специальную "форму" — модуль данных (класс TDataModule). Обратите внимание, что модуль данных не имеет

ничего общего с обычной формой приложения, ведь его непосредственным предком является класс TComponent. В модуле данных можно размещать только невизуальные компоненты. Модуль данных доступен разработчику, как и любой другой модуль проекта, на этапе разработки. Пользователь приложения не может увидеть модуль данных во время выполнения.

Для создания модуля данных можно воспользоваться Репозиторием объектов или главным меню Delphi. Значок модуля данных Data Module расположен на странице New.

Как уже говорилось, модуль данных имеет мало общего со стандартной формой, хотя бы потому, что класс TDataModule происходит непосредственно от класса TComponent. У него почти полностью отсутствуют свойства и методы-обработчики событий, ведь от платформы для других невизуальных компонентов почти ничего не требуется, хотя потомки модуля данных, работающие в распределенных приложениях, выполняют весьма важную работу.

Для создания структуры (модели, диаграммы) данных, с которой работает приложение, можно воспользоваться возможностями, предоставляемыми страницей Diagram Редактора кода. Любой элемент из иерархического дерева компонентов модуля данных можно перенести на страницу диаграммы и задать связи между ними.

При помощи управляющих кнопок можно задавать между элементами диаграммы отношения синхронного просмотра и главный/подчиненный. При этом производится автоматическая настройка свойств соответствующих компонентов.

Для создания модуля данных (рис. 2.2.2) можно воспользоваться Репозиторием объектов или главным меню Delphi. Значок модуля данных Data Module расположен на странице New.

Для обращения компонентов доступа к данным, расположенным в модуле данных, из других модулей проекта необходимо включить имя модуля в секцию uses:

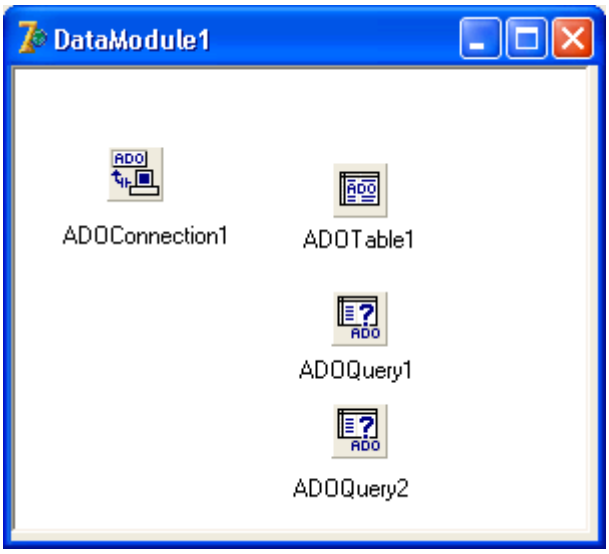


Рис. 2.2.2. Модуль данных

Преимуществом размещения компонентов доступа к данным в модуле данных является то, что изменение значения любого свойства проявится сразу же во всех обычных модулях, к которым подключен этот модуль данных. Кроме этого, все обработчики событий этих компонентов, т. е. вся логика работы с данными приложения, собраны в одном месте, что тоже весьма удобно.

Подключение набора данных

Компонент доступа к данным является основой приложения баз данных. На основе выбранной таблицы БД он создает набор данных и позволяет эффективно управлять им. В процессе работы такой компонент тесно взаимодействует с функциями соответствующей технологии доступа к данным. Обычно доступ к функциональности технологии доступа к данным осуществляется через совокупность интерфейсов. Все компоненты доступа к данным являются не визуальными.

Для создания нового проекта достаточно выбрать команду New Application из меню File или воспользоваться Репозиторием объектов, который открывается командой New из меню File.

Здесь рассматривается простейший вариант создания приложения. В реальных проектах для размещения компонентов доступа к данным следует использовать модуль данных.

Затем на форму нового проекта необходимо перенести компонент, инкапсулирующий набор данных, и выполнить следующие действия. Последовательность действий рассмотрим для компонента, инкапсулирующего функции таблицы.

1. Подключить компонент к базе данных. Для этого, в зависимости от конкретной технологии, используется или специальный компонент, устанавливающий соединение, или прямое обращение к драйверу, интерфейсу или динамической библиотеке.

2. Подключить к компоненту таблицу БД. Для этого используется свойство TableName, доступное в Инспекторе объектов. После выполнения действий первого этапа в списке этого свойства должны появиться имена всех доступных в подключенной базе данных таблиц. После выбора имени таблицы в свойстве TableName компонент оказывается связанным с ней.

3. Переименовать компонент. Это не обязательное действие. Тем не менее, в любых случаях желательно присваивать компонентам доступа к данным осмысленные имена, соответствующие названиям подключенных таблиц. Обычно название компонента копирует название таблицы (например, Orders или OrdTable илт tblOrders).

4. Активизировать связь между компонентом и таблицей БД. Для этого используется свойство Active. Если в Инспекторе объектов присвоить этому свойству значение True, то связь активизируется. Эту операцию можно выполнить и в исходном коде приложения. Также существует метод open, который открывает набор данных, и метод close, закрывающий его.

В качестве примера попробуем создать простейшее приложение баз данных, работающее с таблицей COUNTRY.DB из стандартной демонстрационной базы данных DBDEMOS через драйвер процессора Borland Database Engine.

На форму нового проекта необходимо перенести компонент TTable со страницы BDE Палитры компонентов. Свойство DatabaseName должно ссылаться на псевдоним DBDEMOS, который создается автоматически при

установке Delphi, его можно выбрать из списка свойства DatabaseName. Для свойства TableName необходимо задать имя таблицы "COUNTRY.DB". Его также можно выбрать из списка. Двойной щелчок на свойстве Active в Инспекторе объектов присваивает ему значение True. После этого связь компонента с таблицей активизируется. Свойство Name имеет значение "CountryTable".

Открытие и закрытие набора данных можно предусмотреть как реакцию на действия пользователя или возникновение события. Чаще всего набор данных должен открываться при первом показе формы и закрываться при ее закрытии.

При открытии формы выполняется метод обработчик FormShow. В нем набор данных открывается при помощи метода Open. Обратите внимание на использование конструкции try..except, которая обеспечивает корректное завершение при возникновении исключительных ситуаций.

Так как ошибки в работе приложений баз данных могут привести к серьезным последствиям (потеря или искажение данных), то защитный код должен присутствовать во всех критических местах.

В методе-обработчике FormClose, который вызывается при закрытии формы, набор данных закрывается методом close.

Настройка компонента TDataSource

На втором этапе разработки приложения баз данных необходимо перенести на форму и настроить компонент TDataSource. Он обеспечивает взаимодействие набора данных с компонентами отображения данных. Чаще всего одному набору данных соответствует один компонент TDataSource, хотя их может быть несколько.

Для настройки свойств компонента необходимо выполнить следующие действия.

1. Связать набор данных и компонент TDataSource. Для этого используется свойство DataSet компонента TDataSource, доступное через Инспектор объектов. Это указатель на экземпляр компонента доступа к

данным. В списке этого свойства в Инспекторе объектов перечислены все доступные компоненты наборов данных.

2. Переименовать компонент. Это не обязательное действие. Тем не менее, желательно присваивать компонентам осмысленные имена, соответствующие названиям связанных наборов данных. Обычно название компонента комбинирует имя набора данных (например, OrdSource или dsOrders).

В приложении DemoDBApp компонент countrysource связан с компонентом CountryTable. Поэтому свойство DataSet имеет значение CountryTable.

Компонент TDataSource можно подключить не только к набору данных из той же формы, но и любой другой, модуль которой указан в секции uses.

Компонент TDataSource имеет ряд полезных свойств и методов. Итак, связывание с компонентом набора данных выполняет свойство

property DataSet: TDataSet;

а определить текущее состояние набора данных можно, использовав свойство

type TDataSetState = (dslnactive, dsBrowse, dsEdit, dslnsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue, dsOldValue, dsCurValue, dsBlockRead, dsInternalCalc); property State: TDataSetState;

При помощи свойства *property Enabled: Boolean;* можно включить или отключить все связанные визуальные компоненты. При значении False ни один связанный компонент отображения данных не будет работать.

Свойство *property AutoEdit: Boolean*; при значении True всегда будет переводить набор данных в режим редактирования при получении фокуса одним из связанных визуальных компонентов.

Аналогично, метод *procedure Edit;* переводит связанный набор данных в режим редактирования. Метод

function IsLinkedTo(DataSet: TDataSet): Boolean;

возвращает значение True, если компонент, указанный в параметре DataSet, действительно связан с данным компонентом TDataSource.

Метод-обработчик

type TDataChangeEvent = procedure(Sender: TObject; Field: TField)

of object;

property OnDataChange: TDataChangeEvent;

вызывается при редактировании данных в одном из связанных визуальных компонентов.

Метод-обработчик

property OnUpdateData: TNotifyEvent;

вызывается перед сохранением изменений в базе данных. Метод-обработчик

property OnStateChange: TNotifyEvent;

вызывается при изменении состояния связанного набора данных

Отображение данных

На третьем этапе создания приложения баз данных необходимо разработать пользовательский интерфейс на основе компонентов отображения данных. Эти компоненты предназначены специально для решения задач просмотра и редактирования данных. Внешне большинство этих компонентов ничем не отличаются от стандартных элементов управления. Более того, многие из компонентов отображения данных являются наследниками стандартных компонентов — элементов управления.

Компоненты отображения данных должны быть связаны с компонентом TDataSource и через него с компонентом набора данных. Для этого используется их свойство DataSource. Оно присутствует во всех компонентах отображения данных.

Большинство компонентов предназначены для представления данных из одного единственного поля. В таких компонентах имеется еще одно свойство DataField, которое определяет поле связанного набора данных, отображаемое в компоненте.

Особое значение для приложений баз данных играет компонент TOBGrid, который представляет данные в виде таблицы. В столбцах таблицы размещаются поля набора данных, а в строках — записи. Для этого компонента не имеет смысла определять конкретное поле, но можно задать настраиваемый набор колонок, а для каждой из них определить поле набора данных.

Таким образом, для каждого визуального компонента отображения данных необходимо выполнить следующие операции:

1. Связать компонент отображения данных и компонент TDataSource. Для этого используется свойство Datasource, которое должно указывать на экземпляр требуемого компонента TDataSource. Один компонент отображения данных можно связать только с одним компонентом TDataSource. Необходимый компонент можно выбрать в списке свойств в Инспекторе объектов.

2. Задать поле данных. Для этого используется свойство DataField типа TFields. В нем необходимо указать имя поля связанного набора данных. После задания свойства Datasource поле можно выбрать из списка. Этот этап применяется только для компонентов, отображающих единственное поле.

Отдельное место среди компонентов отображения данных занимает компонент TDBNavigator. Он предназначен для перемещения по записям набора данных. В приложении DemoDBApp использованы компоненты TDBGrid, TDBNavigator и TDBEdit (рис.2.2.3).

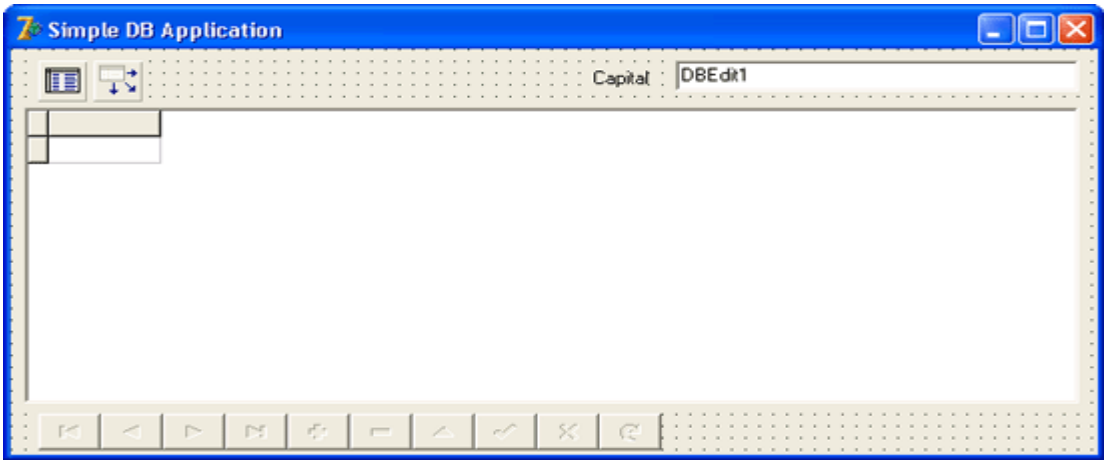


Рис. 2.2.3. Главная форма приложения DemoDBApp

Все три компонента отображения данных связаны с компонентом CountrySource типа TDataSource при помощи свойства DataSource.

Компонент TDBEdit отображает данные из поля capital (столица государства) и позволяет редактировать их.

Компонент TDBGrid показывает набор данных целиком, данные в ячейках можно редактировать.

Компонент TDBNavigator позволяет перемещаться по записям набора данных CountryTable. При этом результат заметен во всех подключенных к набору данных компонентах отображения данных.

2.3. Механизмы управления данными

Стандартный набор данных Delphi инкапсулирует ряд дополнительных механизмов, облегчающих управление записями и полями.

К ним относятся такие полезные функции, как быстрое перемещение по записям, поиск нужной записи по значениям полей, дополнительная фильтрация записей набора данных без использования возможностей СУБД и т. д. Большинство этих механизмов применяют в своей работе индексы таблиц БД.

Абстрактные методы, обеспечивающие управление данными, реализованы в базовом классе TDataSet. А классы-потомки, в свою очередь, реализуют механизмы управления данными в соответствии с возможностями технологий доступа к данным.

Все рассматриваемые в этом разделе методы управления данными в полном объеме доступны только в компонентах, инкапсулирующих таблицу БД. Это связано с тем, что компоненты запросов SQL и хранимых процедур не обеспечивают полноценное использование индексов.

Связанные таблицы. В рамках одного проекта таблицы БД можно связывать отношениями "один - ко - многим" и "многие - ко - многим", при

этом отношения обязательно устанавливаются между индексированными полями двух таблиц.

При создании отношений в качестве главной таблицы можно использовать любой компонент, инкапсулирующий набор данных. Для задания подчиненной таблицы можно использовать только табличные компоненты.

Отношение "один - ко - многим"

Для установления отношения "один - ко - многим" в наборе данных предназначены два свойства — Mastersource и MasterFieids, которые задаются для подчиненной таблицы. Набор данных главной таблицы не требует никаких дополнительных настроек, и заданная связь будет работать только при перемещениях по записям главной таблицы.

Свойство Mastersource определяет компонент TDataSource, который связан с главной таблицей.

Затем при помощи свойства MasterFieids необходимо установить отношения между полями главной и подчиненной таблицы. В нем содержится имя индексированного поля, по которому устанавливается связь. Если таких полей несколько, их имена разделяются точкой с запятой. При этом не все поля, входящие в индекс, обязаны участвовать в создании отношения. Для задания свойства MasterFields можно использовать Редактор связей полей (Field Link Designer), который вызывается щелчком на кнопке в поле редактирования этого свойства в Инспекторе объектов (рис. 2.3.1).

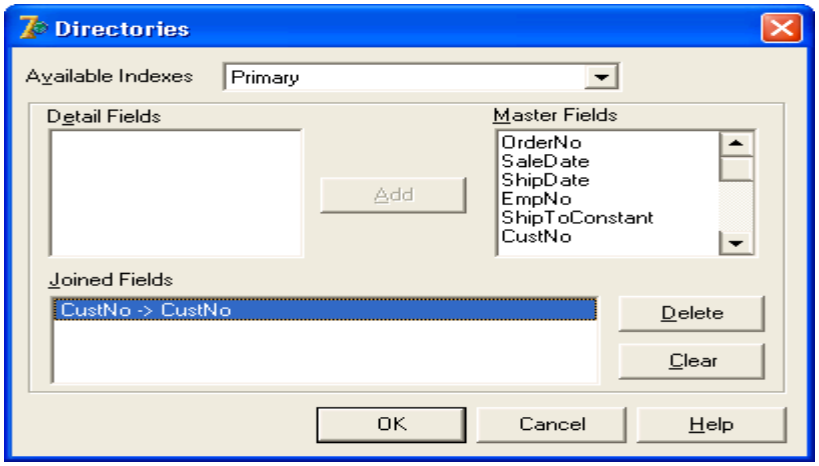


Рис. 2.3.1. Редактор связей полей

Здесь в разворачивающемся списке Available Indexes выбирается требуемый индекс для подчиненной таблицы. После этого в списке Detail Fields появляются имена всех полей, входящих в этот индекс. В списке Master Fields отображаются все поля главной таблицы.

Теперь требуется создать связи между полями. Для этого в левом списке выбирается поле подчиненной таблицы, а затем соответствующее ему поле главной таблицы в правом списке. После этого активизируется кнопка Add, щелчок на которой создает отношение по двум полям главной и подчиненной таблиц. Созданная связь отображается в списке Joined Fields.

После создания связи по индексированным полям данный индекс становится текущим для набора данных. При этом в зависимости от типа СУБД автоматически заполняется свойство indexName или indexFieldNames.

Уже созданные связи можно удалить. Кнопка Delete удаляет выбранную связь, кнопка Clear — все связи.

После создания связей между полями отношение "один- ко- многим" считается установленным. Теперь достаточно открыть оба набора данных, чтобы увидеть работу отношения.

В качестве примера рассмотрим проект DemoJoins, в котором связываются таблицы из демонстрационной базы данных DBDEMOS. Для этого использованы компоненты ADO.

Таблица Customers представлена в наборе данных компонента CustTable, она содержит данные о покупателях. Таблица Orders представлена в наборе данных компонента ordTable, она содержит данные о заказах. Таблица Employee представлена в наборе данных компонента EmpTable, она содержат данные о продавцах (табл. 2.3.1).

Свойство MasterSource должно указывать на компонент custsource, связанный с набором данных CustTable.

Свойство MasterFields указывает на поле custNo таблицы Customers.

В наборе данных OrdTable включен вторичный индекс на основе поля CustNo (indexName = 'CustNo').

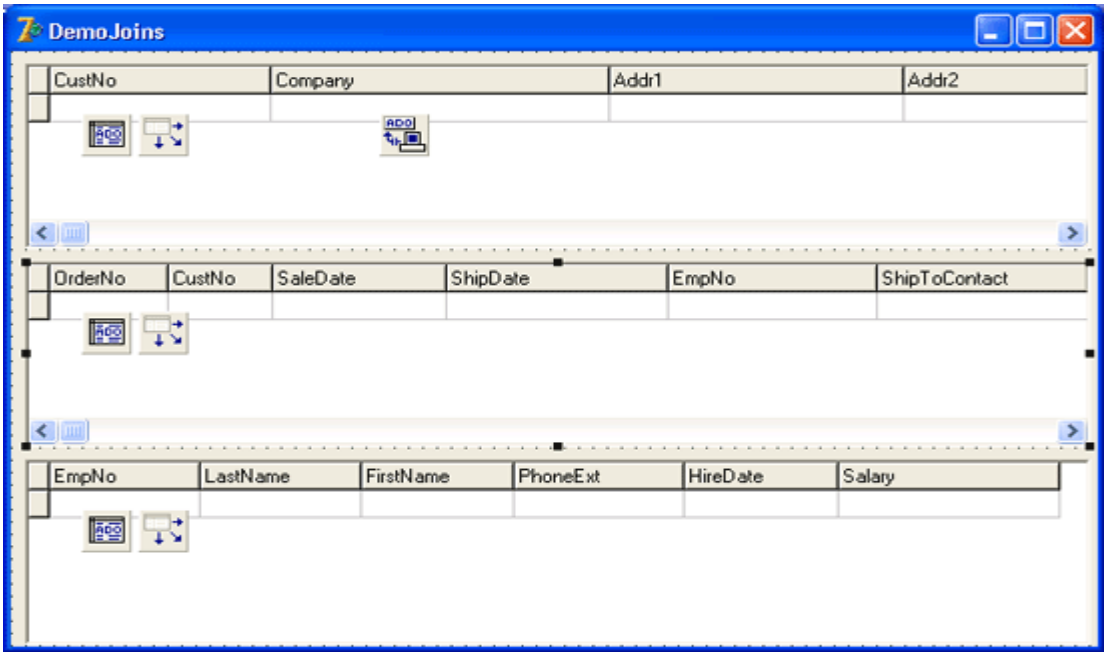


Рис. 2.3.2. Главная форма проекта DemoJoins

Таким образом, две таблицы связаны отношением "один - ко - многим" по индексированным полям custNo (номер покупателя). В результате, при перемещении по записям таблицы покупателей, в таблице заказов будут показаны только те заказы, которые относятся к текущему покупателю

Отношение "многие - ко - многим"

Отношение "многие-ко-многим" отличается тем, что подчиненная таблица еще раз связывается в качестве главной с другой подчиненной таблицей аналогичной последовательностью действий, как и в отношении "один - ко - многим".

В приложении DemoJoins отношением "многие-ко-многим" связаны таблицы заказов (Orders) и продавцов (Employee). Таблица заказов уже работает в отношении "один - ко - многим" в качестве подчиненной.

В наборе данных EtpTable заданы следующие свойства:

свойство MasterSource указывает на компонент Empsource;

свойство MasterFields содержит имя поля EmpNo, по которому осуществляется связь между таблицами. Для подчиненной таблицы поле EmpNo является первичным.

Поиск данных

В наборе данных реализованы два способа поиска записей по заданным значениям полей. Один способ основан на использовании индексов и является более быстрым, но поиск проводится только по индексированным полям. Второй способ применяет специальные методы классов наборов данных и позволяет проводить поиск по любому сочетанию полей, но он более медленный.

Поиск по индексам

Для организации индексного поиска к набору данных должен быть подключен индекс (свойства IndexName ИЛИ IndexFieldNames).

Метод FindKey проводит поиск записи по заданным в параметре значениям ключевых полей текущего индекса набора данных. В случае успеха курсор набора данных устанавливается на найденной записи, а метод возвращает значение True, в противном случае — False.

Если индекс состоит из нескольких полей, значения для поиска записываются в виде множества, причем отсутствующие значения приравниваются к Null.

Рассмотрим простейший пример, в котором реализован поиск по вторичному индексу в таблице CUSTOLY.DB демонстрационной базы данных DBDEMOS. Индекс основан на полях Last_Name И First_Name (рис. 2.3.3).

В компоненте table1, помимо стандартных настроек на таблицу, при помощи свойства IndexName задан и вторичный индекс (его имя Names). Значения для поиска задаются в компонентах Edit1 и Edit2.

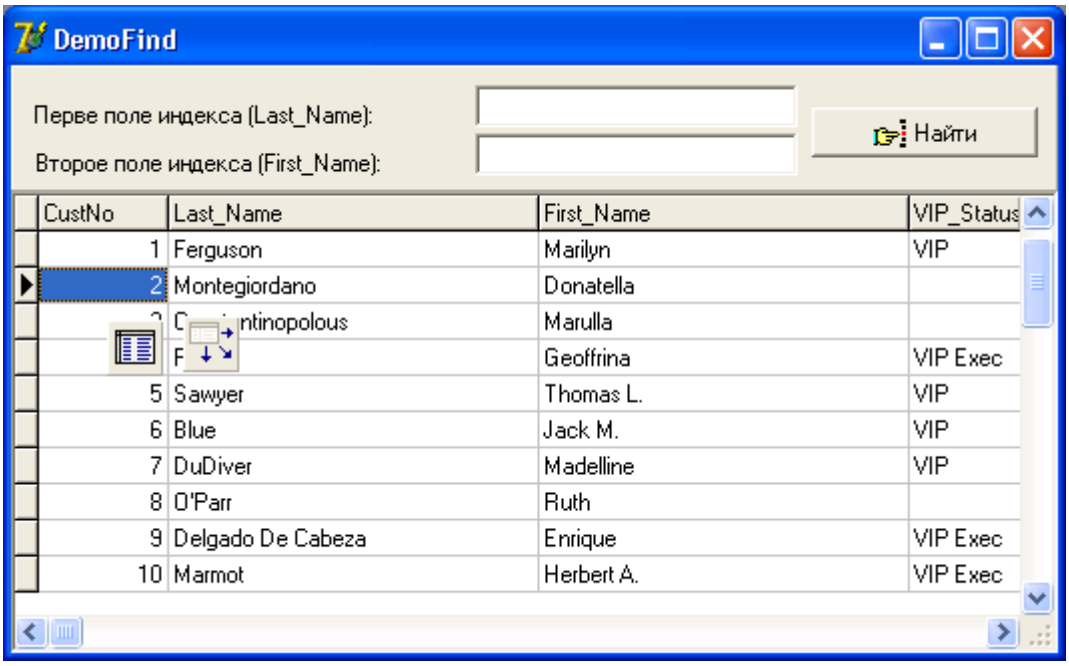


Рис. 2.3.3. Главная форма проекта DemoFind

Набор данных открывается в методе-обработчике FormShow при открытии формы и закрывается в методе-обработчике Formclose. При щелчке на кнопке FindBtn в метод FindKey передаются значения для поиска из компонентов Edit1 и Edit2.

Поиск в диапазоне

Индексный поиск можно организовать группой методов, подобно созданию диапазонов. Метод setKey переводит набор данных в состояние dsSetKey, затем должно следовать присваивание ключевым полям значений для поиска. Сам поиск осуществляется методом GotoKey:

with Tablel do begin

SetKey;

Fields[0].Value := '428';

GotoKey; end;

В случае успеха курсор набора данных устанавливается на найденной записи, а метод возвращает значение True. Вместо этого метода можно применять метод GotoNearest, который в случае неудачного поиска ищет запись, минимально отличающуюся от критерия поиска.

Изменение параметров поиска осуществляется методом EditKey.

Поиск по произвольным полям

Для поиска по произвольной выборке полей можно использовать методы Locate и Lookup.

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

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

В метод Locate необходимо передать список полей, по которым будет идти поиск (параметр KeyFields, имена полей разделяются точкой с запятой), их требуемые значения (параметр KeyValues, значения разделяются запятой) и настройки поиска (параметр options). В настройках можно задать опцию loCaseinsensitive, которая отключает проверку на регистр символов, и опцию loPartiaiKey, которая включает поиск с минимальными отличиями. В случае успеха поиска курсор набора данных устанавливается на найденной записи, а метод возвращает значение True.

Tablel.Locate('Last__Name;First_Name', VarArrayOf(['Editl.Text',

'Edit2.Text']), [] };

В метод Lookup передается список полей для поиска (параметр KeyFields, имена полей разделяются точкой с запятой) и их требуемые значения (параметр KeyValues, значения разделяются запятой). В случае успешного поиска функция возвращает массив значений типа вариант для полей, названия которых содержатся в параметре *ResultFields.*

Tablel.Lookup('Last_Name;First_Name',

VarArrayOf(['Editl.Text', 1Edit2.Text']), 'Last_Name;First_Name');

Оба эти метода автоматически используют быстрый индексный поиск в случае, если в параметре KeyFields задать поля индекса.

Фильтры

Наиболее эффективным способом отбора записей в набор данных является создание и выполнение соответствующего запроса SQL. Но что

делать, если набор данных функционирует на базе табличного компонента? В этом случае на помощь приходит встроенный в набор данных механизм фильтрации данных.

Применение фильтра основано всего на двух основных свойствах и одном вспомогательном. Текст фильтра должен содержаться в свойстве Filter, а свойство Filtered включает и выключает фильтр. Параметры фильтра определяются свойством FilterOptions.

Компонент TQuery также может использовать фильтры. Эта возможность подчас позволяет легко и изящно решать довольно сложные проблемы, которые иначе требуют изменения текста запроса или создания нового компонента запроса.

При использовании фильтра его текст транслируется в синтаксис SQL и передается для выполнения на сервер или через соответствующий драйвер в локальную СУБД.

Фильтры можно разделить на статические и динамические.

Статические фильтры создаются во время разработки приложения и могут использовать как свойство Filter, так и метод OnFilterRecord.

Динамические фильтры можно создавать и редактировать во время выполнения приложения, для них используется только свойство Filter.

При создании текста фильтра для свойства Filter используются имена полей соответствующей таблицы БД, а для задания отношений применяются все операторы сравнения (>, >=, <, <=, =, <>) и логические операторы (AND, OR, NOT):

Fieldl>100 AND Field2=20

Сравнивать между собой два поля нельзя. Следующий фильтр вызовет ошибку при попытке использования:

ItemCount=Balance AND InputPrice>OutputPrice

При создании динамических фильтров можно изменять как выражение фильтра целиком, так и его части. Например, ограничивающее значение для

поля можно задавать при помощи элементов управления формы, что позволяет пользователю приложения управлять фильтрацией набора данных:

```
procedure TForml.EditlChange(Sender: TObject);
begin
with Tablel do begin
Filtered := False;
Filter := 'Fieldl>=' + TEdit(Sender).Text; Filtered := True;
end;
end;
```

Фильтр начинает работать только после того, как свойству Filtered присваивается истинное значение. Перед изменением текста динамического фильтра или для отключения фильтра свойству Filtered присваивается значение False.

Параметры фильтра определяются свойством FilterOptions:

```
property FiiterOptions: TFilterOptions;
TFilterOption = (foCaselnsensitive, foNoPartialCompare);
TFilterOptions = set of TFilterOption;
```

Параметр foCaselnsensitive, будучи включенным в свойстве, отключает сравнение строковых значений с учетом регистра символов.

Параметр foNoPartialCompare отключает отбор строковых значений по части строки.

Метод-обработчик onFilterRecord имеет следующее объявление:

```
type TFilterRecordEvent = procedure(DataSet: TDataSet; var Accept:
Boolean) of object;
property OnFiiterRecord: TFilterRecordEvent;
```

Если этот метод создан для набора данных, то он вызывается для каждой его записи. Программный код метода должен присваивать параметру Accept истинное или ложное значение. В результате запись передается в набор данных или отсекается:

```
procedure TForml.TablelFilterRecord(DataSet: TDataSet;
 var Accept: Boolean);
begin
Accept := ArchOrdersArchDat.AsString >= DateEditl.Text;
end;
```

Важнейшее преимущество метода *onFilterRecord,* по сравнению со свойством *Filter,* заключается в том, что в этом методе-обработчике можно сравнивать поля и производить вычисления над их значениями.

Недостатком метода является недостаточная гибкость, хотя такой фильтр можно модифицировать путем присвоения методу процедурной переменной, содержащей ссылку на новый метод.

Быстрый переход к помеченным записям

Закладки, как инструмент работы с записями набора данных, позволяют осуществлять быстрое перемещение на нужную запись. Набор данных может содержать неограниченное число закладок, каждая из которых представляет собой указатель. Закладку можно создать только для текущей записи набора данных.

При работе с закладками используются три основных метода:

метод GetBookmark создает новую закладку для текущей записи;

метод GotoBookmark осуществляет переход к закладке, переданной в параметре;

метод FreeBookmark удаляет закладку, переданную в параметре.

Кроме этого, можно использовать метод Bookmarkvalid, который проверяет, указывает ли закладка на реально существующую запись. Метод compareBookmark позволяет сравнить между собой две закладки:

```
var Bookmarkl, Bookmark2: TBookmark;
...
if Tablel.CompareBookmark(Bookmarkl, Bookmark2) = 1
```

then ShowMessage (' Закладки одинаковы') ;

В наборе данных имеется свойство Bookmark, которое содержит название текущей закладки.

Рассмотрим небольшой пример, где право управлять закладками предоставлено пользователю (рис. 2.3.4). На форме, помимо других элементов управления (среди которых есть компонент TDBGrid), имеются две кнопки. Кнопка startBookmark помечает текущую запись, кнопка stopBookmark переходит к закладке, а затем уничтожает ее.

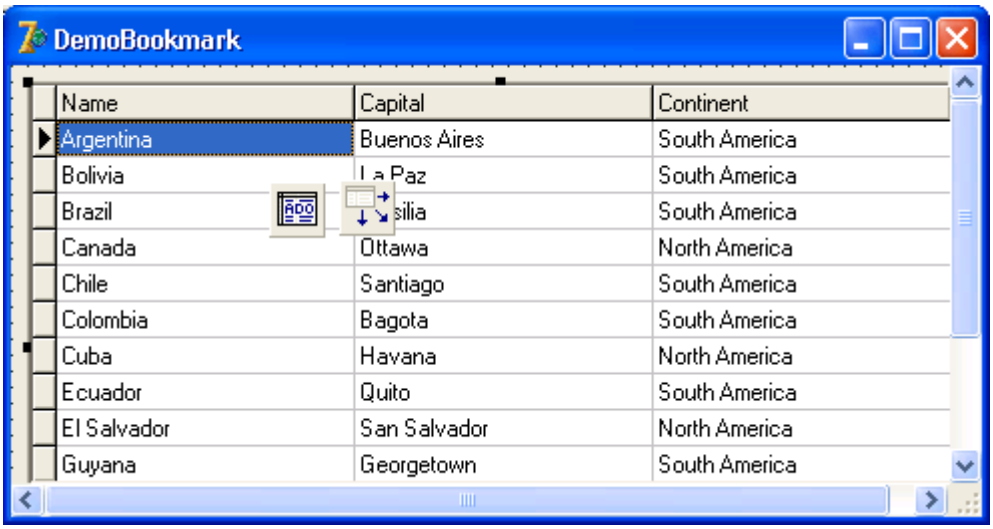


Рис. 2.3.4. Главная форма проекта DemoBookmark

Использование метода Bookmarkvaiid позволяет корректно переопределять закладку, если она уже установлена, и избежать ошибок при произвольных нажатиях кнопок. Компонент BookmarkControl типа TShape сигнализирует о том, что закладка установлена или удалена.

Закладки также используются в компоненте TDBGrid. Он имеет свойство SelectedRows типа TBookmarkList, которое представляет собой список закладок, указывающих на одновременно выделенные записи.

Диапазоны

В наборе данных, помимо фильтров, имеется еще одно средство отбора записей. Группа методов позволяет на основе использования индексов

отбирать в набор данных только те записи, значения индексированных полей которых (для текущего индекса) соответствуют диапазону заданных величин.

Диапазоны работают быстрее фильтров, но менее гибки и не так удобны в работе. При использовании диапазонов набор данных обязательно должен находиться в состоянии dsSetKey

Для того чтобы включить диапазон, необходимо задать стартовое и конечное значение диапазона для ключевых полей, затем применить созданный диапазон к набору данных. Работающий диапазон можно модифицировать.

Все методы работы с диапазонами используют те поля, которые заданы в текущем индексе. Для таблиц Paradox и dBASE это свойство indexName. Для таблиц серверов SQL это свойство indexFieldNames.

Метод setRangestart переводит набор данных в режим dsSetKey, следующее за этим присваивание ключевым полям значений означает задание начальной границы диапазона.

Метод setRangeEnd переводит набор данных в режим dsSetKey, следующее за этим присваивание ключевым полям значений означает задание конечной границы диапазона.

Работающий диапазон можно модифицировать аналогичным образом: после вызова методов EditRangestart и EditRangeEnd необходимо задать новые границы для ключевых полей и снова вызвать метод AppiyRange:

```
with Tablel do
 begin
EditRangeStart;
Fields[0].Value := '500';
EditRangeEnd;
Fields[1].Value := '522';
AppiyRange;
end;
```

Отмена диапазона осуществляется методом CancelRange.

Если индекс содержит несколько полей, то перед вызовом метода AppiyRange необходимо задать значения для всех ключевых полей.

Для одновременного задания верхней и нижней границы диапазона можно использовать Метод SetRange.

```
with Tabiel do
 begin
SetRange(['500'], ['522']);
AppiyRange;
end;
```

Тем, какая граница будет у диапазона — открытая или закрытая, управляет свойство KeyExclusive. Если оно имеет значение True, граничные значения в диапазон не включаются, в противном случае — включаются.

Разработчик приложений БД в Delphi может использовать ряд полезных механизмов набора данных, которые реализованы для компонентов всех технологий доступа к данным.

К этим механизмам относятся методы быстрого поиска и перехода к найденным записям; связывания наборов данных по индексированным полям; метод дополнительной фильтрации записей набора данных.

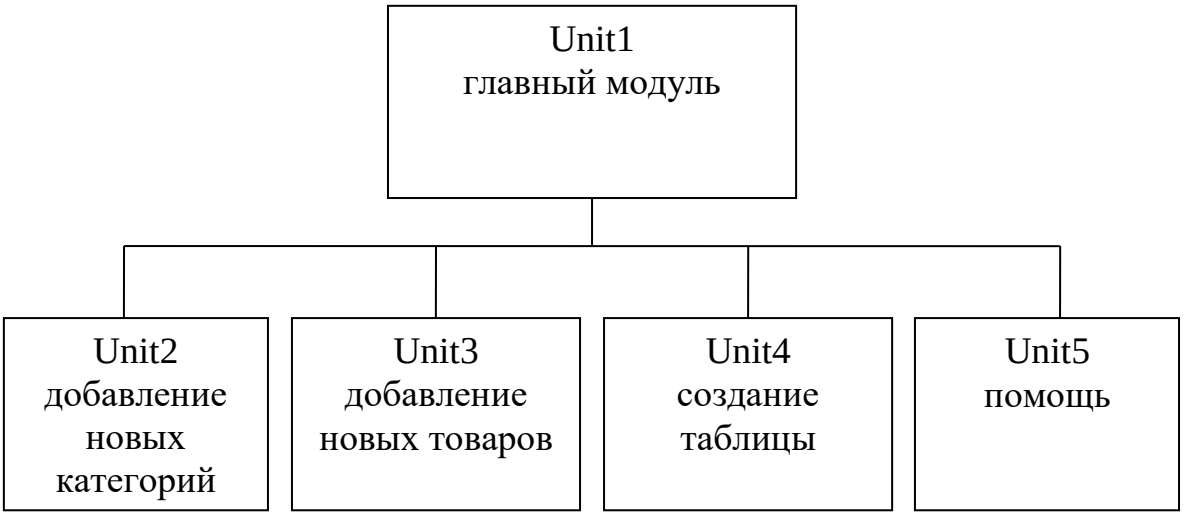
Вывод по главе II.

Данная глава состоит из трех разделов. В первом разделе главы приведено общие понятие о среде Дельфи. Втором разделе описано архитектура приложений баз данных, т.е. о приложении Delphi. Приложение баз данных имеет пользовательский интерфейс. И этот интерфейс создается с использованием стандартных и специализированных визуальных компонентов на обычных формах. В третьем разделе приведены механизмы управления данными. Здесь основном приведены о таблицах и связах между ними, о поисках данных, фильтрах.

Глава III Создание базы данных учета материальных ценностей организации

3.1 Структура и описание программы

В данном разделе рассматривается структура программного обеспечения и описание модулей. Программное обеспечение состоит из следующих модулей: Unit1 (главный модуль), Unit2(добавление новых категорий),Unit3 (добавление новых товаров),Unit4(создание таблицы),Unit5(помощь).



Теперь рассмотрим описание моделей и процедур.

Unit1 состоит из следующих процедур 14 процедур:

TForm1.tsbtnAddClick(Sender: TObject),

TForm1.tsbtnDeleteClick(Sender: TObject),

TForm1.tsbtnChangeClick(Sender: TObject),

TForm1.tsbtnReportClick(Sender: TObject),

TForm1.ToolButton1Click(Sender: TObject),

TForm1.btnSearchClick(Sender: TObject),

TForm1.Button1Click(Sender: TObject),

TForm1.Close1Click(Sender: TObject),

TForm1.ADd1Click(Sender: TObject),

TForm1.Changeproduct1Click(Sender: TObject),

TForm1.Change1Click(Sender: TObject),

TForm1.Delete1Click(Sender: TObject),

TForm1.Delete2Click(Sender: TObject),

TForm1.Productslist1Click(Sender: TObject),

TForm1.About1Click(Sender: TObject).

Теперь приводим тело некоторые процедур выполняющих следующие операции: поиск, сортировка, добавление, удаление и.т.д.

TForm1.tsbtnAddClick(Sender: TObject)- процедура выполняет следующую функцию показать форму добавление категорий который расположен в Форме2. Тело процедуры имеет вид:

```
procedure TForm1.tsbtnAddClick(Sender: TObject);
begin add := True;
 CategoryID :=  Table1.FieldByName('CategoryID').Value;
   if DBGrid1.Focused then
     begin
       Form2.Edit1.Text := '';
       Form2.ShowModal;
     end;
   if DBGrid2.Focused then
     begin       FProducts.ShowModal;
     end;
end;
```

TForm1.tsbtnChangeClick(Sender: TObject)- процедура выполняет изменение название продукции и имеет вид:

```
procedure TForm1.tsbtnChangeClick(Sender: TObject);
begin
 add := False;
     if DBGrid1.Focused then
       begin
```

```
        Form2.Edit1.Text := Table1.FieldByName('CategoryName').Text;
        Form2.ShowModal;
      end;
    if DBGrid2.Focused then
      begin
        FProducts.ShowModal;
      end;
end;
```

TForm2.btnOKClick(Sender: TObject)- процедура добавляет новую категорию и изменяет существующую категорию.

```
procedure TForm2.btnOKClick(Sender: TObject);
begin
 if add then
   begin
     Form1.Table1.Insert;
   end
 else
   begin
     Form1.Table1.Edit;
   end;
 Form1.Table1.FieldByName('CategoryName').AsString := Edit1.Text;
 Form1.Table1.Post;
 Form1.Table1.Refresh;
 Form2.Close;
end;
```

TFProducts.btnOKClick(Sender: TObject)- процедура добавляет новую продукцию и изменяет существующую название товара. А процедура имеет вид:

```
procedure TFProducts.btnOKClick(Sender: TObject);
begin
 if add then
   begin
     Form1.Table2.Insert;
```

```
    end
  else
    begin
      Form1.Table2.Edit;
    end;
  Form1.Table2.FieldByName('CategoryID').AsInteger := Form1.CategoryID;
  Form1.Table2.FieldByName('ProductName').AsString := txbProductName.Text;
  Form1.Table2.FieldByName('Maker').AsString := txbMaker.Text;
  Form1.Table2.FieldByName('Price').AsString := txbPrice.Text;
  Form1.Table2.FieldByName('Count').AsString := txbCount.Text;
  Form1.Table2.FieldByName('Description').AsString := txbDescription.Text;
  Form1.Table2.Post;
  Form1.Table2.Refresh;
  FProducts.Close;
end;
```

TForm1.tsbtnDeleteClick(Sender: TObject)- процедура выполняет удаление ненужных категорий и товаров.

```
procedure TForm1.tsbtnDeleteClick(Sender: TObject);
begin
 if DBGrid1.Focused then
    begin
       if MessageDlg('Удалить ' + Table1.FieldByName('CategoryName').Text +'
категорию ?', mtConfirmation, mbOKCancel, 0 ) = mrOK then
        Table1.Delete;
    end;
 if DbGrid2.Focused then
    begin
       if MessageDlg('Удалить ' + Table2.FieldByName('ProductName').Text +' продукт ?',
mtConfirmation, mbOKCancel, 0 ) = mrOK then
        Table2.Delete;
    end;
```

```
end;
```

TForm1.ToolButton1Click(Sender: TObject)- процедура открывает окно поиска тело процедуры имеет вид:

```
procedure TForm1.ToolButton1Click(Sender: TObject);
begin
  PageControl1.ActivePage :=  tpSearch;
end;
```

TForm1.tsbtnReportClick(Sender: TObject)- процедура выполняет отчет и печатает через принтер:

```
procedure TForm1.tsbtnReportClick(Sender: TObject);
var s,s1: string;
  i,j: integer;
begin
  s:= Application.ExeName;
  for i:=length(s) downto 1 do
  begin
    if s[i] = '\' then break;
  end;
  s1:='';
  for j := 1 to i do
  begin
    s1 := s1 + s[j];
  end;
  s1 := s1 + 'Products.rav';
 begin
      RvDataSetConnection1.DataSet := Table3;
    RvProject1.ProjectFile := s1;
    RvProject1.ExecuteReport('Products');
    end;
end;
```

TForm1.Button1Click(Sender: TObject) - процедура выполняет поиск по всем параметрам категорий и товаров:

```
procedure TForm1.Button1Click(Sender: TObject);
var i: string;
begin
  case (RadioGroup1.ItemIndex) of
    0: i:='select p.productId, c.categoryName, p.productName, p.maker, p.price, p.count, description from products p inner join categories c on p.categoryId=c.categoryId where p.categoryId in (select categoryId from categories where categoryname like '"+txtSearch.Text+'%")';
    1: i:='select p.productId, c.categoryName, p.productName, p.maker, p.price, p.count, description from products p inner join categories c on p.categoryId=c.categoryId where p.productName like '"+txtSearch.Text+'%"';
    2: i:='select p.productId, c.categoryName, p.productName, p.maker, p.price, p.count, description from products p inner join categories c on p.categoryId=c.categoryId where p.price like '"+txtSearch.Text+'%"';
    3: i:='select p.productId, c.categoryName, p.productName, p.maker, p.price, p.count, description from products p inner join categories c on p.categoryId=c.categoryId where p.maker like '"+txtSearch.Text+'%"';
    4: i:='select p.productId, c.categoryName, p.productName, p.maker, p.price, p.count, description from products p inner join categories c on p.categoryId=c.categoryId where p.count like '"+txtSearch.Text+'%"';
    end;
    ADOQuery1.SQL.Text:=i;
    ADOQuery1.Prepared:=true;
    ADOQuery1.Active:=true;
end;
```

Теперь приводим процедуры Unit2:

TForm2.btnOKClick(Sender: TObject)- выполняет добавление новых категорий в таблицу1.

```
procedure TForm2.btnOKClick(Sender: TObject);
begin
  if add then
    begin
      Form1.Table1.Insert;
    end
  else
    begin
      Form1.Table1.Edit;
    end;
  Form1.Table1.FieldByName('CategoryName').AsString := Edit1.Text;
  Form1.Table1.Post;
  Form1.Table1.Refresh;
  Form2.Close;
end;
```

Следующая процедура показывает форму, где добавляется название категории и изменение название категорий

```
procedure TForm2.FormShow(Sender: TObject);
begin
    if  Form1.add then
        begin
         add := True;
         Form2.Caption := «Добавить новую категорию»;
        end;
    if Form1.add = False then
        begin
         add := False;
         Form2.Caption := « Правка категорий »;
        end;
end;
```

Слдеующая процедура закрывает форму 2

```
procedure TForm2.btnCancelClick(Sender: TObject);
begin
Form2.Close;
end;
```

В модуле Unit 3 выполняется ввод название товара и цена товара, производитель товара, количества товара. А также выполняется изменение введенных параметров товара.

```
procedure TFProducts.FormShow(Sender: TObject);
begin
    if Form1.add = True then
        begin
         add := True;
```

```
        FProducts.Caption := 'Добавить новую продукт';
        txbProductName.Text := '';
        txbMaker.Text := '';
        txbPrice.Text := '0';
        txbCount.Text := '0';
        txbDescription.Text := '';
       end;
   if Form1.add = False then
      begin
       add := False;
       FProducts.Caption := 'Правка продукта';
       txbProductName.Text := Form1.Table2.FieldByName('ProductName').Text;
       txbMaker.Text := Form1.Table2.FieldByName('Maker').Text;
       txbPrice.Text := Form1.Table2.FieldByName('Price').Text;
       txbCount.Text := Form1.Table2.FieldByName('Count').Text;
       txbDescription.Text := Form1.Table2.FieldByName('Description').Text;
      end;
end;

procedure TFProducts.btnCancelClick(Sender: TObject);
begin
  FProducts.Close;
end;

procedure TFProducts.btnOKClick(Sender: TObject);
begin
  if add then
    begin
      Form1.Table2.Insert;
    end
  else
    begin
      Form1.Table2.Edit;
    end;
  Form1.Table2.FieldByName('CategoryID').AsInteger := Form1.CategoryID;
  Form1.Table2.FieldByName('ProductName').AsString := txbProductName.Text;
  Form1.Table2.FieldByName('Maker').AsString := txbMaker.Text;
  Form1.Table2.FieldByName('Price').AsString := txbPrice.Text;
  Form1.Table2.FieldByName('Count').AsString := txbCount.Text;
  Form1.Table2.FieldByName('Description').AsString := txbDescription.Text;
  Form1.Table2.Post;
  Form1.Table2.Refresh;
  FProducts.Close;
end;
```

В модуле Unit4 приводится данные о таблицах а Unit5 приведено инструкция с использованием программы.

3.2. Инструкция по использованию программного обеспечения

Чтобы работать с базой данных организации надо с начало установить некоторые приложения базы данных Delphi для этого выполняется следующие шаги.

1. Программа Product Catalog Manager

1. Создать базу данных MS Access.
    a. Добавить таблицу Categories.(Код, Наименование категории).
    b. Добавить таблицу Products.(Код, Код Категории, Наименование продукта, Производитель, Количество, Цена, Описание).

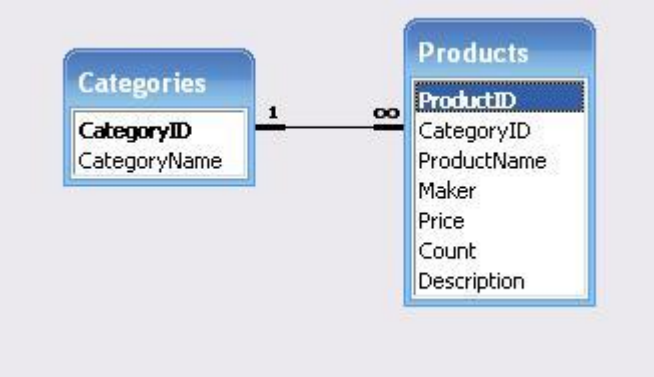


рис.3.2.1.Связь между таблицами Categories иProducts.

2. Методы, чтобы получить доступ к данным в Базе данных MS Access от Delphi, используя стандартные Компоненты BDE

2. Создать новый ODBC DNS.

a. Открытый BDE Администратор (от Панели управления).

b. Правый щелчок и выбирает Администратора ODBC.

c. Нажать на вкладку System DSN.

d. Нажать на кнопку Add.

e. Выбрать Microsoft Access Drive(*.MDB) из списка.

f. Вводите, Название Источника Данных (DSN) в предоставленной коробке (назовите это Delphi_Jds_Test например),

g. Нажать на кнопку Select и найти вашу Базу данных Доступа (DelphiSample.mdb).

h. Нажать OK, чтобы сохранить новый DSN.

i. Закрыть Администратора ODBC

3. Создать BDE Alias

a. Правый щелчок на Базу данных в Администраторе BDE

b. Выбрать New из контекстного меню

c. Выбрать Microsoft Access Driver (*.MDB)

d. Нажать OK.

e. Дать новому псевдониму название (типа Bde_Jds_Test)

f. Щелкнуть кнопкой Apply (синяя стрелка на панели инструментов)

g. Нажать на ODBC DNS вход справа Администратора BDE и

h. Найти наш новый ODBC DSN (мы назвали это Jds_Test прежде).

i. Нажать на кнопку Apply снова.

j. Закрыть Администратора BDE

4. Вызов к данным

a. Открыть Delphi

b. Создать новый Проект

c. Добавить Компонент TDatabase.

d. Выбрать новый псевдоним из собственности AliasName (мы назвали это Bde_Jds_Test прежде),

e. Вводить, новое название в DatabaseName (назовите это Delphi_Jds_Test например),

f. Изменить LoginPrompt к false (не нуждаются в этом для Access DBs).

g. Добавить компонент TTable на форме.

h. Изменить DatabaseName к TDatabase компанентам DatabaseName (Delphi_Jds_Test)

i. Щелчок на TableName и выбрать имя Таблицы из списка

j. Изменить Active к true.

k. Добавить компонент TDataSource на форме.

l. Дать DataSet к имени TTable (Table1)

5. Показать данные

a. Поместить данные, знающие средства управления как TDBGrid, TDBEdit, TDBNavigator на форме

b. Установить свойство Datasource к названию (DataSource1){имени DataSource1)} TDatasource

c. Для TDBGrid, формируйте свойства Column

d. Для TDBEdit, дать свойства DataField каждого контроля{управления} к соответствующей области{полю} db field.

2.1 Создать файл с расширением *.udl т.е.

C:\Program Files\connection.udl и здесь provider=Microsoft Jet.4.0 baza

3. Работа с программным обеспечением.

Цель программы: Регистрация товаров, и хранение информации о товарах. Быстрый поиск по базе.

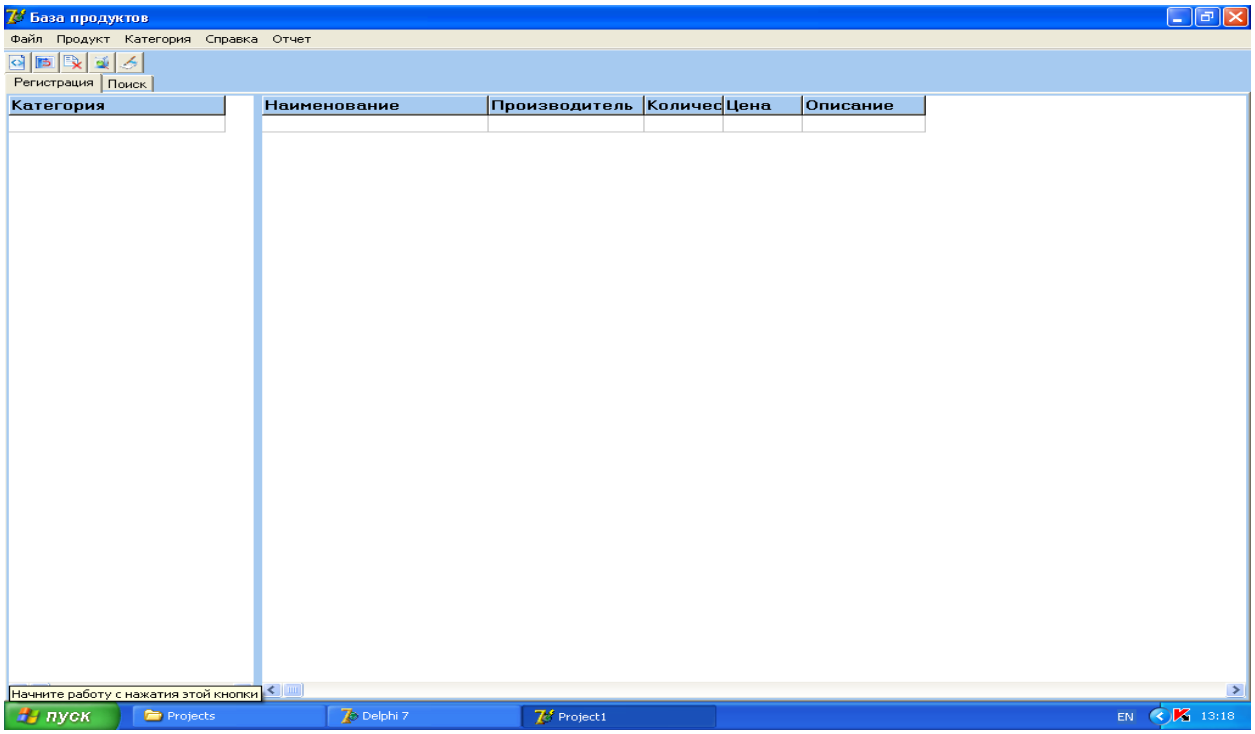


Рис. 3.2.2 Основная форма программы.

1. Информация как будет отражаться в форме пользователя.
    a. В левой стороне экрана будет список категории.
    b. Выбирает категорию, и в правой стране экрана появиться список товаров по этой категорий.
2. Регистрация нового товара или категории.

a. Для того чтобы добавить новую категорию выбираем список

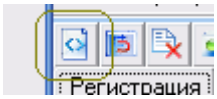


Рис.3.2.3. Икона для добавления новых категорий (наименование товаров)

b. категории и нажимается кнопка добавить.

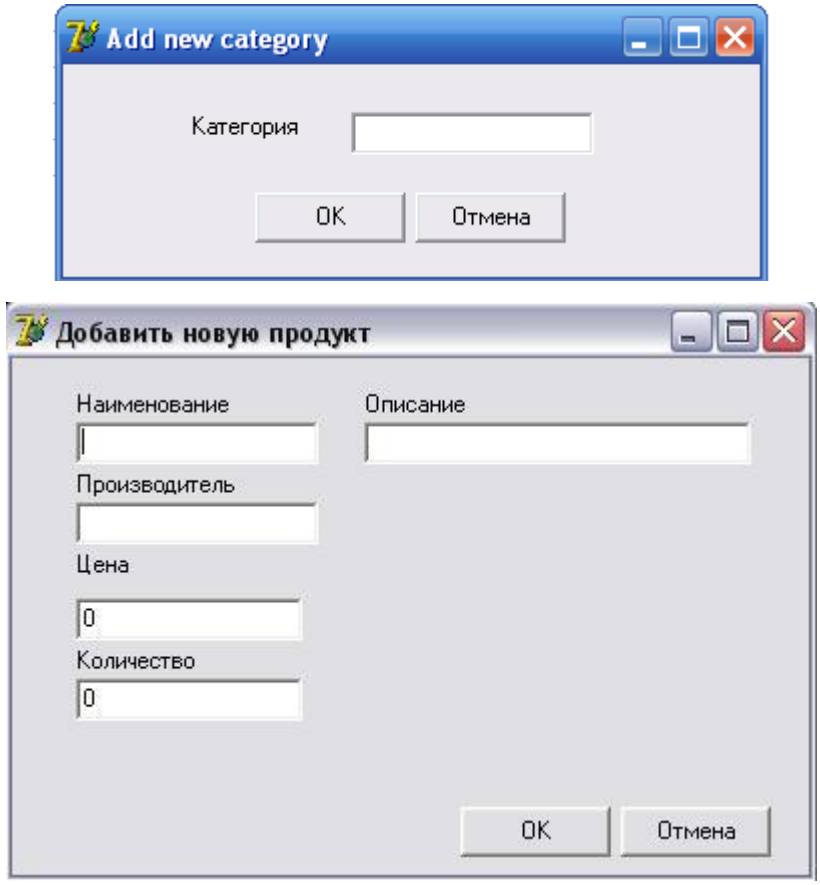


Рис.3.2.4. Добавление новых категорий (наименование товаров)

c. После этого откроется форма для ввода новыой категории

d. В поле «Категория» вводим новую категорию.

e. Регистрация нового продукта:

Далее для изменения введенных данных используется следующие формы

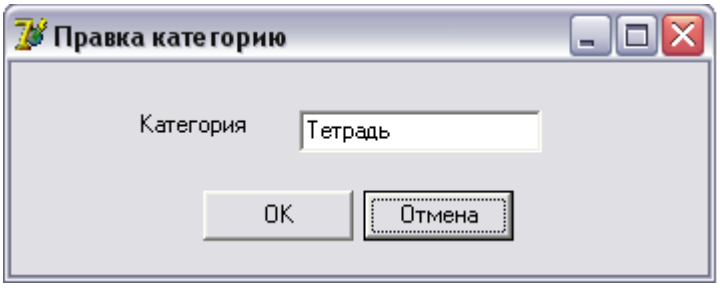


Рис.3.2.5. Изменение название категорий (наименование товаров)

Для поиска по заданным параметрами категорий выполняется следующие процедуры: выбирается «категория» и в поле поиск вводится название искомой категории, после выполнения операции в правой части окна выводится список найденных категорий.

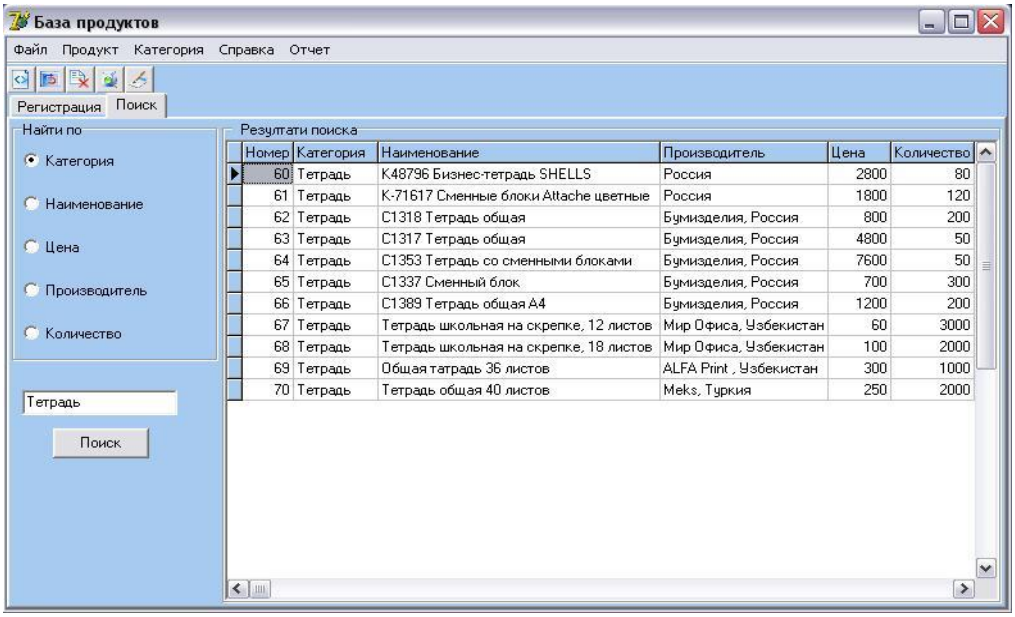


Рис.3.2.5. Поиск категорий (наименование товаров)

Если нам необходимо удалить ненужные категории то выполняем следующие операции: Из списка расположенный в левой части окна выбираем ненужную категорию и из списка иконки (см. рис.3.2.3) используется третья пиктограмма, после нажатия иконки на экран выводится сообщение об удалении категории, если нажать на кнопку ОК то удаляется категория в противном случае удаление отменяется.

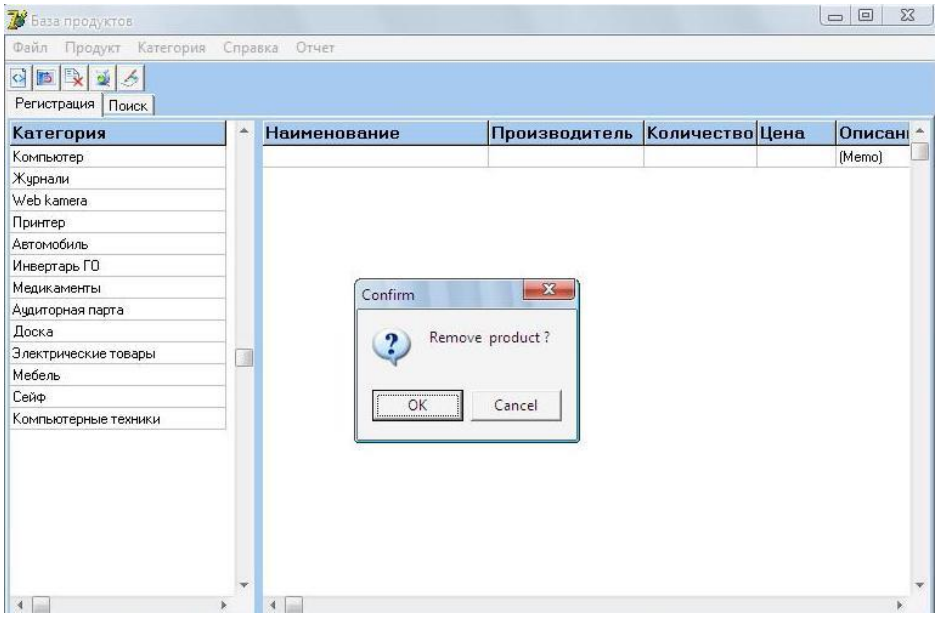


Рис.3.2.6. Удаление категорий (наименование товаров)

Если нам необходим отчет, то выполняется следующие операции. Из меню выбирает пункт «отчет», чтобы получить распечатку отчета то выполняется следующие функций приведенные на рис.:

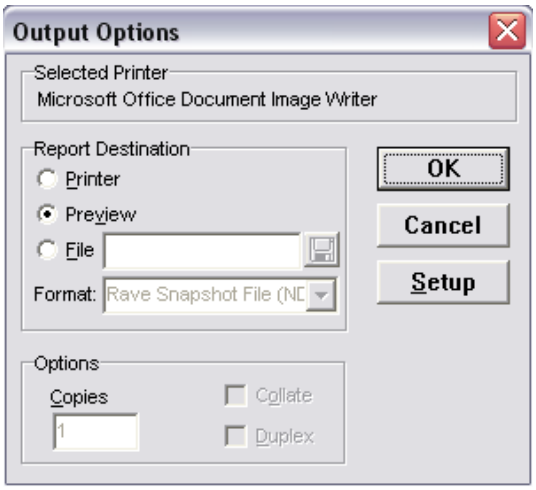


Рис.3.2.7. Отчет

Вывод по Главе III

В данной главе в разделе 3.1 приведено структура и описание программного обеспечения. Здесь приведены основные модули и некоторые процедуры программного обеспечения. А в разделе 3.2. приведена инструкция по использованию программного обеспечения. Здесь описаны этапы установки программы, а также инструкция к использованию данного программного обеспечения.

Заключение

В ходе выполнения работы была освоена СУБД Access и среда программирование Дельфи. Была также разработана программа, имитирующая часть работы учета материальных ценностей организации. С помощью СУБД позволяет получать данные о товарах, которые числится на балансе организации, о хранящихся товарах, о купленных товарах, поиск и сортировка товаров. В данном ВКР была, проанализирована предметная область и на основе этого анализа были реализованы постановка и алгоритмизация, а также машинная реализация задачи.

В проекте была решена задача автоматизации ведения и программирования работы учета материальных баз организации.

Для решения этих задач проектирование базы данных было разбито на несколько этапов и выполнены следующие:

- проведен анализ предметной области;
- разработана информационно-поисковая система для работ учета товаров;
- разработан удобный интерфейс программы;
- приведена инструкция по использованию данного программного обеспечения.

Список использованной литературы

1. Фаронов В.В. Delphi 6. Москва. «Нолидж».
2. Архангельский А.Я. Программирование в Delphi 7. Москва. «Бином». 2004 г.
3. Фленов Ф.М. Библия Delphi. Учебный курс.
4. Грехем И. Объектно-ориентированные методы. Принципы и практика. Вильямс. 879 стр, 2004 г.
5. Иванова Г.С. Объектно-ориентированное программирование. Учебник. МГТУ им Баумана. 320 стр, 2003 г.
6. Бекаревич Ю., Пушкина Н. Самоучитель Microsoft Access 2002,
7. БХВ Петербург 2003 г.
8. Пасько Виктор Microsoft Office для пользователя, БХВ Петербург 2001г.
9. MSDN - http://msdn.microsoft.com
10. www.citforum.ru/database

ПРИЛОЖЕНИЕ

Работа с программным обеспечением.

Цель программы: Регистрация товаров, и хранение информации о товарах. Быстрый поиск по базе.

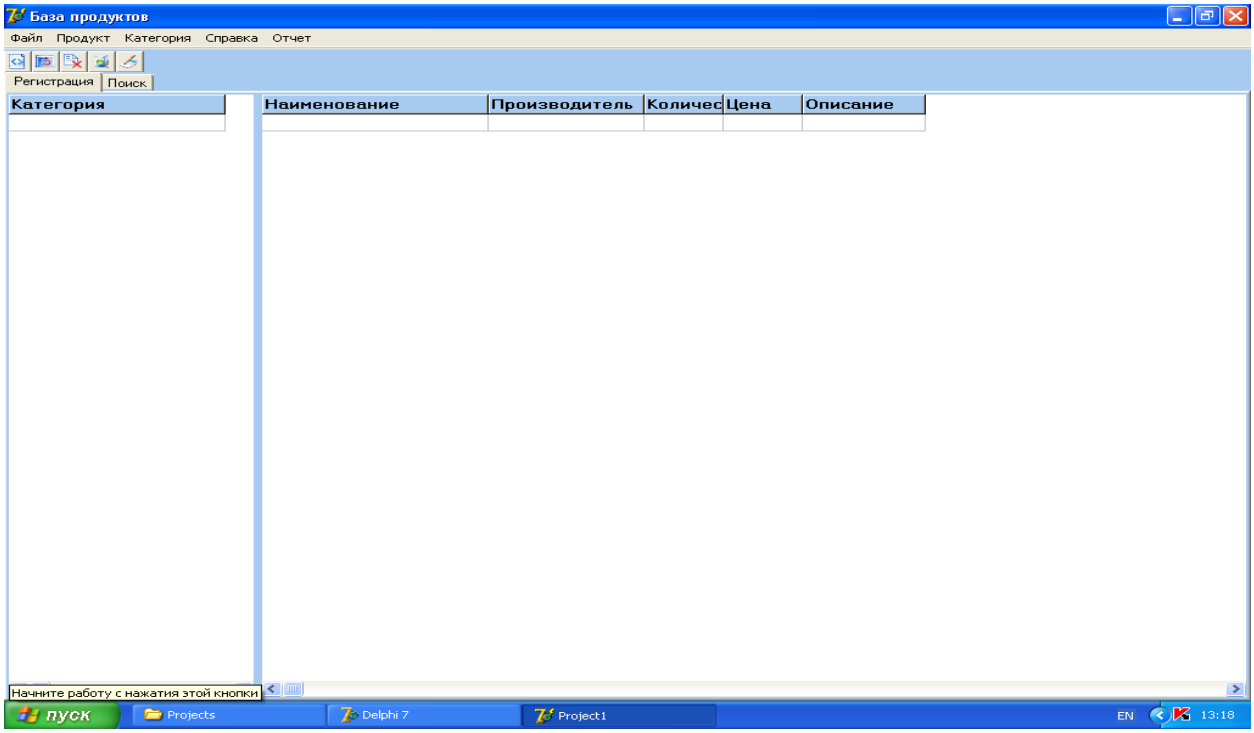


Рис. 3.2.2 Основная(Первоначальная) форма программы.

3. Информация как будет отражаться в форме пользователя.
    a. В левой стороне экрана будет список категории.
    b. Выбирает категорию, и в правой стране экрана появиться список товаров по этой категорий.
4. Регистрация нового товара или категории.
    a. Для того чтобы добавить новую категорию выбираем список

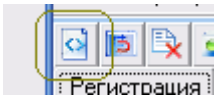
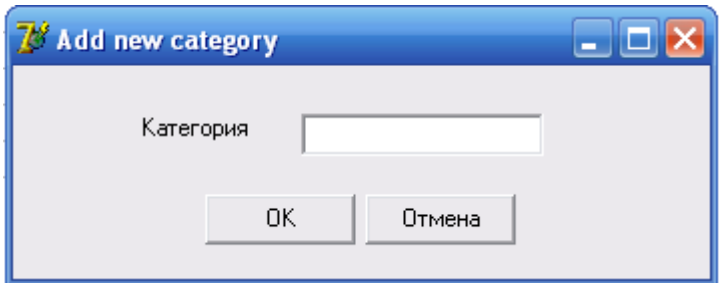


Рис.3.2.3. Икона для добавления новых категорий (наименование товаров)

    b. категории и нажимается кнопка добавить.



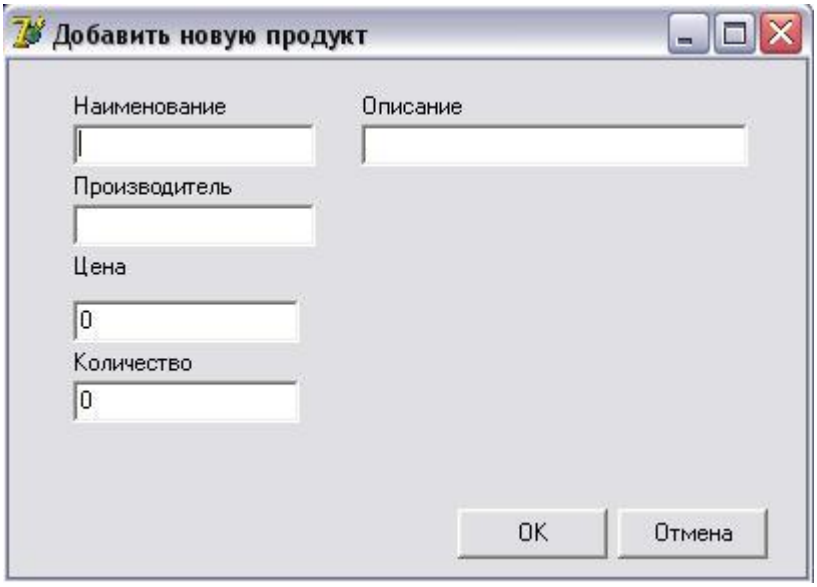
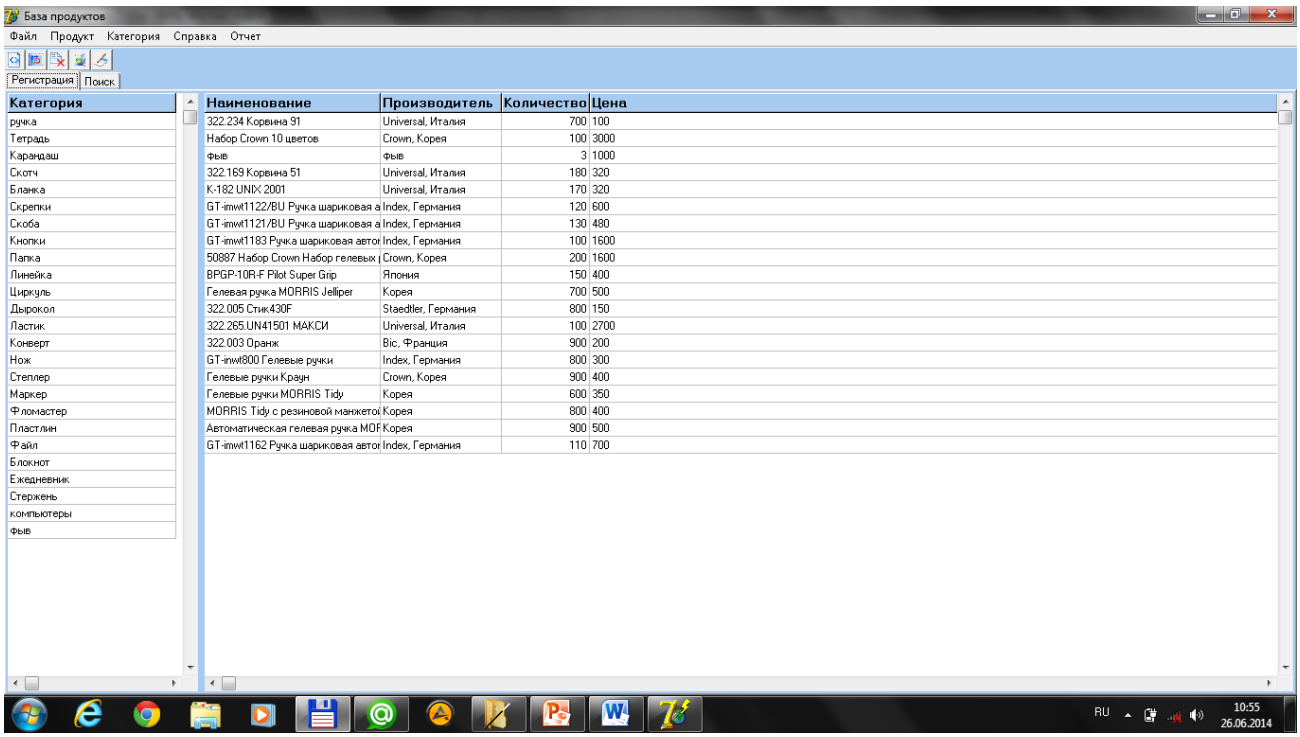
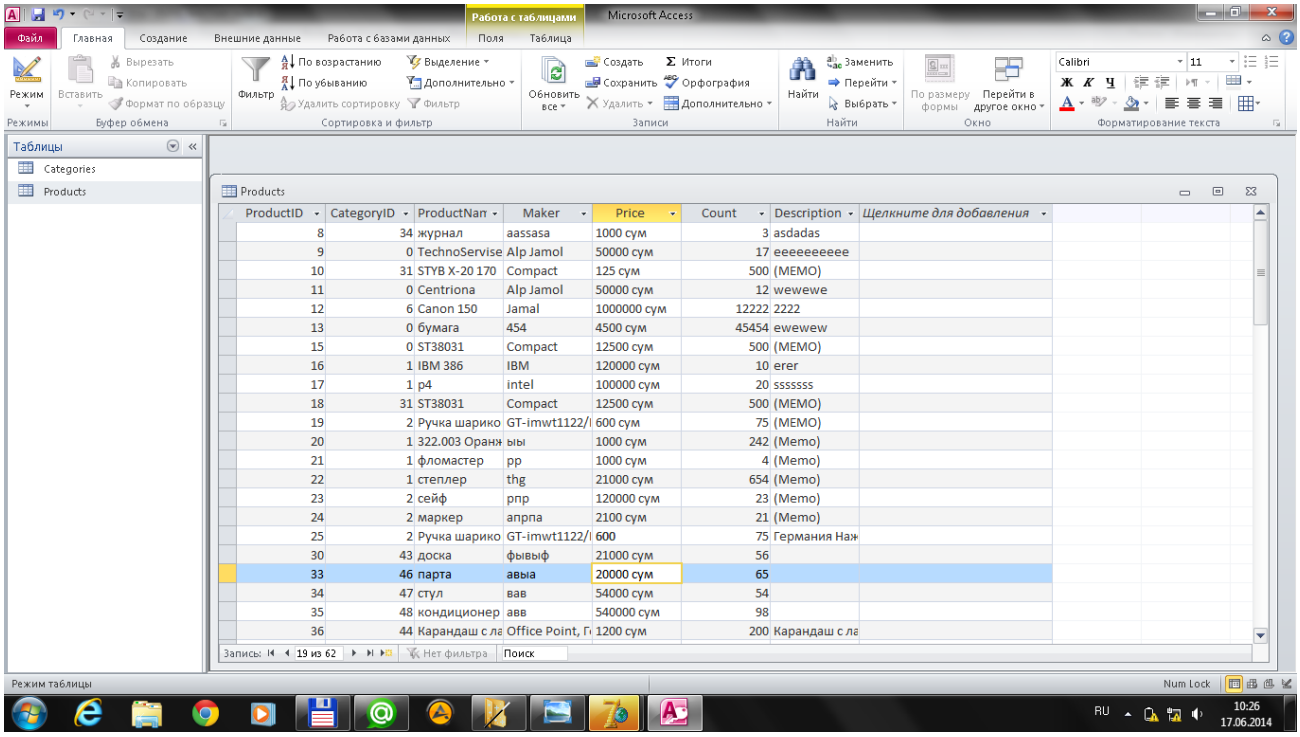


Рис.3.2.4. Добавление новых категорий (наименование товаров)

c. После этого откроется форма для ввода новой категории
d. В поле «Категория» вводим новую категорию.
e. Регистрация нового продукта



Вид программы после заполненя категорий




```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2},
  DM in 'DM.pas' {DataModuleTest: TDataModule},
  Unit3 in 'Unit3.pas' {FProducts},
  Unit4 in '..\Unit4.pas' {Frame4: TFrame},
  Unit5 in 'Unit5.pas' {FChangeCategory},
  About in 'C:\Program Files\Borland\Delphi7\ObjRepos\ABOUT.pas'
{AboutBox},
  Unit6 in 'Unit6.pas' {AboutBox6};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TDataModuleTest, DataModuleTest);
  Application.CreateForm(TFProducts, FProducts);
  Application.CreateForm(TFChangeCategory, FChangeCategory);
  Application.CreateForm(TAboutBox6, AboutBox6);
  Application.Run;
end.
```

```
unit Unit3;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, ComCtrls, Unit1, Mask;
type
  TFProducts = class(TForm)
    txbProductName: TLabeledEdit;
    txbMaker: TLabeledEdit;
    btnOK: TButton;
    btnCancel: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    txbPrice: TEdit;
    txbCount: TEdit;
    txbDescription: TEdit;
    procedure FormShow(Sender: TObject);
    procedure btnCancelClick(Sender: TObject);
    procedure btnOKClick(Sender: TObject);
  private
    { Private declarations }
    add:Boolean;
  public
    { Public declarations }
  end;
var
  FProducts: TFProducts;

implementation

{$R *.dfm}

procedure TFProducts.FormShow(Sender: TObject);
begin
    if Form1.add = True then
        begin
         add := True;
         FProducts.Caption := 'Добавить новую продукт';
         txbProductName.Text := '';
         txbMaker.Text := '';
         txbPrice.Text := '0';
```

```
        txbCount.Text := '0';
        txbDescription.Text := '';
       end;
   if Form1.add = False then
       begin
        add := False;
        FProducts.Caption := 'Правка продукта';
        txbProductName.Text := Form1.Table2.FieldByName('ProductName').Text;
        txbMaker.Text := Form1.Table2.FieldByName('Maker').Text;
        txbPrice.Text := Form1.Table2.FieldByName('Price').Text;
        txbCount.Text := Form1.Table2.FieldByName('Count').Text;
        txbDescription.Text := Form1.Table2.FieldByName('Description').Text;
       end;
end;

procedure TFProducts.btnCancelClick(Sender: TObject);
begin
  FProducts.Close;
end;
procedure TFProducts.btnOKClick(Sender: TObject);
begin
  if add then
    begin
      Form1.Table2.Insert;
    end
  else
    begin
      Form1.Table2.Edit;
    end;
  Form1.Table2.FieldByName('CategoryID').AsInteger := Form1.CategoryID;
  Form1.Table2.FieldByName('ProductName').AsString := txbProductName.Text;
  Form1.Table2.FieldByName('Maker').AsString := txbMaker.Text;
  Form1.Table2.FieldByName('Price').AsString := txbPrice.Text;
  Form1.Table2.FieldByName('Count').AsString := txbCount.Text;
  Form1.Table2.FieldByName('Description').AsString := txbDescription.Text;
  Form1.Table2.Post;
  Form1.Table2.Refresh;
  FProducts.Close;
end;
end.
```