Міністерство освіти і науки України
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

Г. Г. Швачич, О. В. Овсянніков, В. В. Кузьменко, Н. І. Нєчаєва

СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ

Частина1

Дніпропетровськ НМетАУ 2007

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

Г. Г. Швачич, О. В. Овсяніков, В. В. Кузьменко, Н. І. Нєчаєва

СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ

Частина1

Затверджено на засіданні Вченої ради академії
як конспект лекцій

Дніпропетровськ НМетАУ 2007

УДК 004 (075.8)
Г.Г. Швачич, О.В. Овсянніков, В.В. Кузьменко, Н.І. Нєчаєва. Системи управління базами даних. Конспект лекцій. Частина1. – Дніпропетровськ: – НМетАУ, 2007. – 48 с.

Викладені основи створення інформаційно-логічних моделей та конструювання систем управління базами даних у середовищі розробки прикладного програмного забезпечення Delphi.

Призначений для студентів спеціальності 6.020100 – документознавство та інформаційна діяльність.

Іл. 31. Бібліогр.: 5 найм.

Відповідальний за випуск Г.Г. Швачич, канд. техн. наук, проф.

Рецензенти: Б. И. Мороз д-р техн. наук, проф. (Академія таможенної Службы Украины)
Т. И. Пашова канд. техн. наук, доц. (Дніпропетровсьгий державний аграрний університет)

ТЕМА 1 КОНЦЕПЦИИ ПОСТРОЕНИЯ И СТАДИИ ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Базы данных являются одной из основных составляющих информационных технологий предприятий. В настоящее время без применения баз данных трудно представить работу любого современного предприятия.

1.1 Основные понятия

Под базой данных понимается некоторая унифицированная совокупность данных, совместно используемая группой лиц, персоналом предприятия, отрасли, ведомства, населением региона, страны, мира. Задача базы данных состоит в хранении всех представляющих интерес данных в одном или нескольких местах, причем таким способом, который заведомо исключает возможную избыточность.

Под идеальной базой данных понимается база, в которой отсутствует избыточность и противоречивость данных. Наиболее приближенными к идеальной базе данных являются реляционные базы, которые в настоящее время получили наибольшее распространение.

Для создания реляционных баз данных в среде **Delphi** используется ядро баз данных **Borland Database Engine.**

В общем случае, базы данных можно разделить на два основных типа: локальные базы данных и серверные базы данных. Процесс проектирования баз данных единый для обеих архитектур баз данных и отличается лишь отдельными деталями.

Жизненный цикл базы данных, зависит от качества проектирования структуры базы данных и приложений доступа к данным. От того, насколько тщательно продумана структура базы, насколько четко определены связи между ее элементами, зависит производительность

системы, ее информационная насыщенность и модифицируемость, а значит – и время ее жизни.

Правильно спроектированное приложение управления базой данных должно удовлетворять следующим основным требованиям:

- Обеспечить необходимый пользователю информационный объем и содержание данных.
- Обеспечить легкое для восприятия структурирование информации и удобный пользовательский интерфейс.
- Гарантировать непротиворечивость данных.
- Минимизировать избыточность данных.
- Обеспечить максимальную производительность доступа к данным.

Перед проектированием базы необходимо провести комплекс исследований связанный с определением объема и вида хранимой информации, обеспечивающей возможность получения необходимой дополнительной (расчетной) информации из хранимой информации. Также необходимо:

- Проанализировать объекты и смоделировать их в базе данных.
- Сформировать из объектов *сущности* определить их характеристики.
- В соответствии сущностям разработать структуры таблиц и описать их поля.
- Определить атрибуты, которые будут являться идентификаторами объектов.
- Установить связи между объектами и выполнить нормализацию объектов.
- Выработать принципы управления данными, которые будут определять и поддерживать целостность данных.
- Обеспечивать надежность системы управления базой данных.

Помимо указанных мероприятий необходимо определить информацию, являющуюся конфиденциальной и выработать правила доступа к такой информации и вид ее хранения.

В результате проведенного исследования и выполненного анализа должна быть, во-первых, разработана структурная схема базы данных, во-

вторых, определена ее платформа, в-третьих, разработана функциональная схема управления, включающая функцию администрирования.

1.2 Концепции построения реляционных баз данных

Реляционная теория определяет несколько базовых понятий. Одним из основных понятий является понятие **отношения**. Математически отношение определяется следующим образом. Пусть заданы **n** множеств $\mathbf{D_1, D_2, ..., D_n}$. Тогда **R** есть отношение этих множеств, при условии, если **R** есть множество упорядоченных наборов вида: $\mathbf{d_1, d_2, ..., d_n}$, где $\mathbf{d_1}$ - элемент из $\mathbf{D_1}$, $\mathbf{d_2}$ - элемент из $\mathbf{D_2}$, ..., $\mathbf{d_n}$ – элемент из $\mathbf{D_n}$. При этом наборы вида: $\mathbf{d_1, d_2, ..., d_n}$ называются **кортежами**, а множества $\mathbf{D_1, D_2 ,..., D_n}$ – **доменами**. Каждый **кортеж** состоит из элементов, выбираемых из своих **доменов**. Эти элементы называются **атрибутами**, а их значения – **значениями атрибутов**. Графическое представление отношений показано на рис.1.

Графическое представление отношений

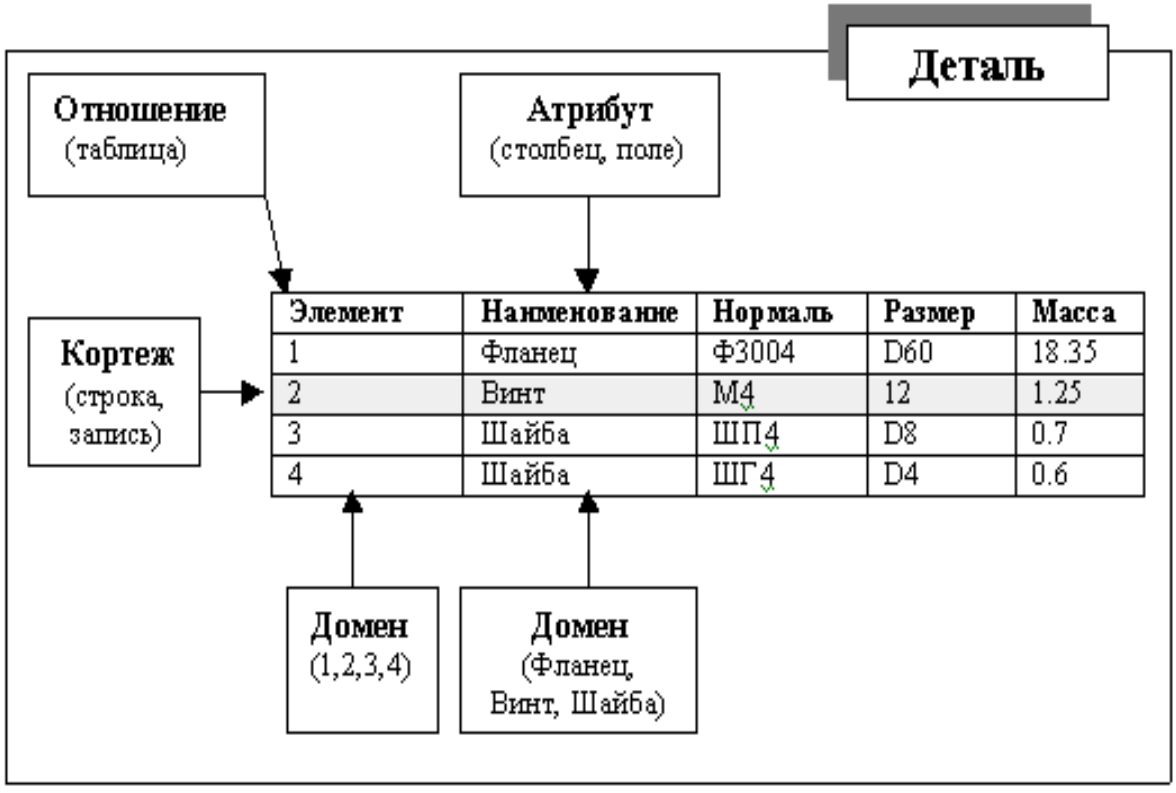


Рис.1

Необходимо обратить внимание на то, что отношение является отражением некоторой сущности реального мира. В данном примере сущностью является деталь. С точки зрения обработки данных сущность представляет собой таблицу. В локальных базах данных каждая таблица представляет собой отдельный файл. Поэтому с точки зрения размещения данных для локальных баз данных отношение можно отождествлять с файлом. Кортеж представляет собой строку в таблице, которая является записью. Атрибут – это столбец таблицы, который называется полем в конкретной записи. Домен представляет собой некий обобщенный образ, который может быть источником для типов полей в записи. Следовательно, следующие тройки терминов являются эквивалентными:

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

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

Атрибут, который может быть использован для однозначной идентификации конкретного кортежа, 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, который называется **индексированием**. Индекс представляет собой древовидный список, указывающий на истинное местоположение записи для каждого первичного ключа. В разных **СУБД** индексы реализованы по-разному. В локальных **СУБД** индексы реализованы в виде отдельных файлов.

Помимо первичного индексирования 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 выборки и сортировки данных.

Индексирование играет важную роль при восстановлении связей в базе данных. После удаления отдельных записей из таблицы, содержащих **blob** поля, само отношение становится не упорядоченным, потому, что в нем остаются не удаленные строки. Напротив индекс (индексный файл), остается упорядоченным, так как в нем удаляется соответствующая запись, и как следствие ссылка на удаленный кортеж исчезает, и целостность связей сохраняется. Для очистки базы данных от ненужной информации используется механизм переиндексирования, с помощью которого автоматически удаляется ненужные кортежи.

Для поддержания ссылочной целостности данных во многих **СУБД** имеется механизм **внешних ключей**. Смысл этого механизма состоит в том, что некоему атрибуту (или группе атрибутов) одного отношения назначается ссылка на первичный ключ другого отношения. В результате чего устанавливаются 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 нарушения связей между отношениями, а именно:

- если выполняется попытка вставить в подчиненную таблицу запись, для несуществующего внешнего ключа в главной таблице, **СУБД** сгенерирует ошибку;
- если выполнена попытка удаления из главной таблицы записи, на первичный ключ которой имеется хотя бы одна ссылка из подчиненной таблицы, **СУБД** сгенерирует ошибку;
- если производится попытка изменить первичный ключ записи главной таблицы, на которую имеется хотя бы одна ссылка из подчиненной таблицы, **СУБД** сгенерирует ошибку.

Администрирование, связанное с удалением записей в главной таблице может осуществляться двумя способами. Первый способ

запрещает пользователю удаление любой записи, а также изменение первичных ключей главной таблицы, на которые имеются ссылки подчиненной таблицы. Второй способ обеспечивает распространение всех изменений в первичном ключе главной таблицы на подчиненную таблицу, а именно:

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

1.3 Стадии проектирования реляционных баз данных

Проектирование любой базы данных начинается с определения информационного содержания базы данных. ***Первая стадия*** проектирования включает в себя опрос будущих пользователей с целью документирования их требований к предоставляемой информации. На этой стадии также определяется организационная структура базы данных и мероприятия по ее администрированию.

Вторая стадия проектирования включает в себя анализ объектов реального мира, которые необходимо смоделировать в базе данных. На этой стадии разрабатывается информационная структура базы данных.

Третья стадия проектирования предусматривает установление соответствий между сущностями и характеристиками предметной области, отношениями и атрибутами в выбранной **СУБД**. Исходя из того, что каждая сущность реального мира обладает собственными характеристиками, которые в совокупности образуют полную картину их проявлений, становиться возможным, выявить их соответствия, и построить набор отношений (таблиц) и их атрибутов (полей).

На ***четвертой стадии*** проектирования определяются атрибуты, которые уникальным образом идентифицируют каждый объект. Для того чтобы обеспечить единичный доступ к строке таблицы необходимо определить первичный ключ для каждого из отношений. Если нет возможности идентифицировать кортеж с помощью одного атрибута, то

первичный ключ может быть составным, т.е. содержать несколько атрибутов. Первичный ключ гарантирует, что в таблице не будет содержаться двух и более одинаковых строк. Во многих **СУБД** имеется возможность помимо первичного ключа определять еще ряд уникальных ключей. Отличие уникального ключа от первичного ключа состоит в том, что уникальный ключ не является главным идентификатором записи и на него не может ссылаться внешний ключ другой таблицы. Его главная задача состоит в гарантировании уникальности значения поля.

Пятая стадия проектирования предполагает разработку методов, которые должны определять и поддерживать целостность данных. В **СУБД** архитектуры клиент/сервер данные методы заведомо определены и поддерживаются автоматически сервером баз данных. В локальных **СУБД** указанные методы должна быть реализованы в пользовательском приложении.

Шестая стадия проектирования базы данных состоит в установлении связи между объектами (таблицами и столбцами) и исключении избыточности данных – нормализации таблиц. Каждый вид связей должен быть смоделирован в базе данных. Существуют три основных вида связей:

- связь «один к одному»;
- связь «один ко многим»;
- связь «многие ко многим».

Связь «один к одному» представляет собой простейший вид связи данных. Данный вид характеризуется тем, что первичный ключ главной таблицы является в то же время внешним ключом, ссылающимся на первичный ключ другой таблицы. Такую связь удобно устанавливать тогда, когда нет необходимости держать разные по типу или другим критериям данные в одной таблице. Например, можно выделить данные с общим описанием изделия в отдельную таблицу и установить связь «один к одному» с таблицей, содержащей описание элементов изделия.

Связь «один ко многим» отражает реальную взаимосвязь сущностей в предметной области. Эта связь реализуется уже описанной парой «внешний ключ – первичный ключ», т.е. когда определен внешний ключ

главной таблицы, который ссылается на первичный ключ других таблиц. Именно эта связь описывает широко распространенный механизм классификаторов. Для установки такой связи существует справочная таблица, содержащая названия, имена и другую информацию, и определенные коды. В такой схеме первичным ключом является код. В главной таблице, содержащей базовую информацию, определяется внешний ключ, который ссылается на первичный ключ классификатора. После этого в нее заносится не название из классификатора, а код. Такая система становится устойчивой от изменения названий в классификаторах. Существуют способы быстрой подмены в отображаемой таблице кодов на их названия, как на уровне сервера, 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. Один из наиболее распространенных способов заключается в создании дополнительной таблицы, строки которой состоят из внешних ключей, ссылающихся на первичные ключи других таблиц.

После определения количества таблиц, их структуры, полей, индексов и связей между таблицами следует проанализировать проектируемую базу данных в целом, с целью устранения логических ошибок. Для этого используются правила нормализации. Суть нормализации заключается в том, что структура каждой таблицы реляционной базы данных должна удовлетворять 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 только 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 к **нормальным формам.** 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 приведения таблиц к трем первым формам

Приведение к первой нормальной форме заключается в устранении повторяющихся групп. На стадии приведения ко второй нормальной форме производится удаление частично зависимых атрибутов. Удаление транзитивно зависимых атрибутов выполняется при приведении к третьей нормальной форме.

На заключительной стадии проектирования баз данных разрабатывается полный комплекс мероприятий администрирования. Он включает в себя:

- надежность системы и отдельных ее элементов;
- конфиденциальность информации и права доступа к ней;
- создание резервных копий базы данных, способы и места ее сохранения;
- нормативные материалы по обслуживанию и ведению базы данных и инструкции для пользователей.

ТЕМА 2 ПОДДЕРЖКА БАЗ ДАННЫХ В СРЕДЕ DELPHI

Разработка приложений баз данных является одной из ключевых функций среды Delphi. Помимо большого набора компонентов доступа и управления данными среда Delphi содержит собственный процессор управления базами данных Borland Database Engine, СУБД InterBase, а также необходимые в работе утилиты, позволяющие быстро разрабатывать базы данных различных структур и конфигураций.

2.1 Открытая архитектура средств поддержки баз данных

Поддержка разработки приложений баз данных в среде **Delphi** осуществляется при помощи:

- DBE (Borland Database Engine).
- СУБД InterBase (версии: 5.0 – 5.6).
- Утилиты Database Explorer.
- Утилиты Database Desktop.
- Мастера форм баз данных.
- Компонентов баз данных.

Среда **Delphi** предлагает открытую архитектуру средств поддержки баз данных (рис.2).

Основным связующим звеном между приложением и базой данных является компонент **TDataSet**

Приложения баз данных строятся на основе компонентов доступа к базам данных и компонентов управления базами данных. Эти компоненты могут быть связаны с локальными базами данных следующих форматов: **dBase, Paradox, ASCII, FoxPro** и **Access**. Кроме указанных форматов баз данных **DBE** используется для доступа к локальным и удаленным **SQL** серверам.

Для разработки баз данных, не имеющих непосредственного доступа к функциям **DBE**, предназначен компонент **TClientDataSet**, посредством которого можно обращаться к данным **OLE** сервера.

Открытая архитектура средств поддержки баз данных

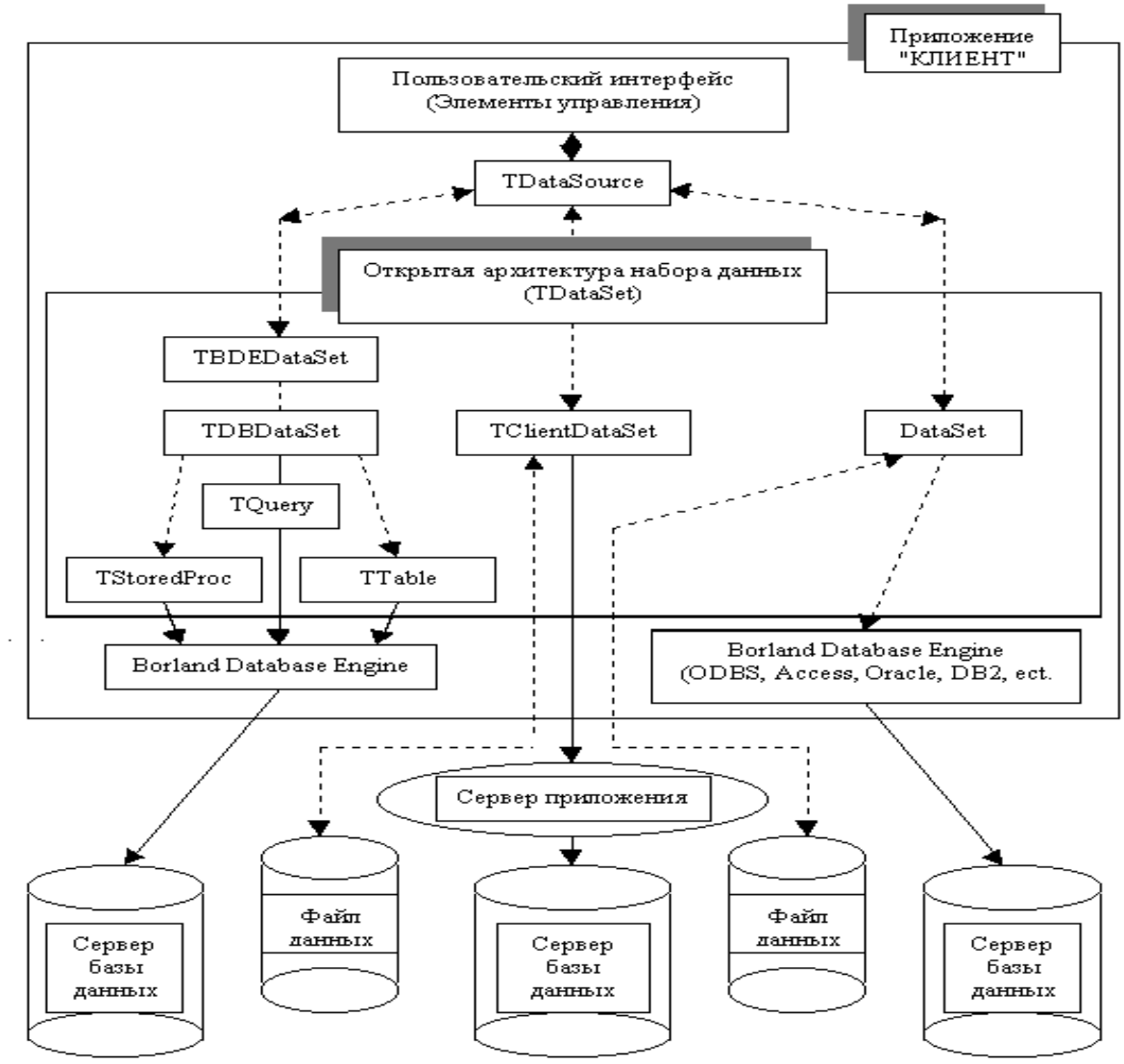


Рис.2

2.2 Утилита DBE Administrator

Конфигурирование **DBE** осуществляется посредством утилиты **DBE Administrator**, которая представлена в программной группе среды **Delphi**.Утилита **DBE Administrator** содержит две страницы **DataBases** (рис.3) и **Configuration** (рис.4). Все настройки **DBE** сохраняются в файле **IDAPI.CFG.**

Страница DataBases DBE Administrator

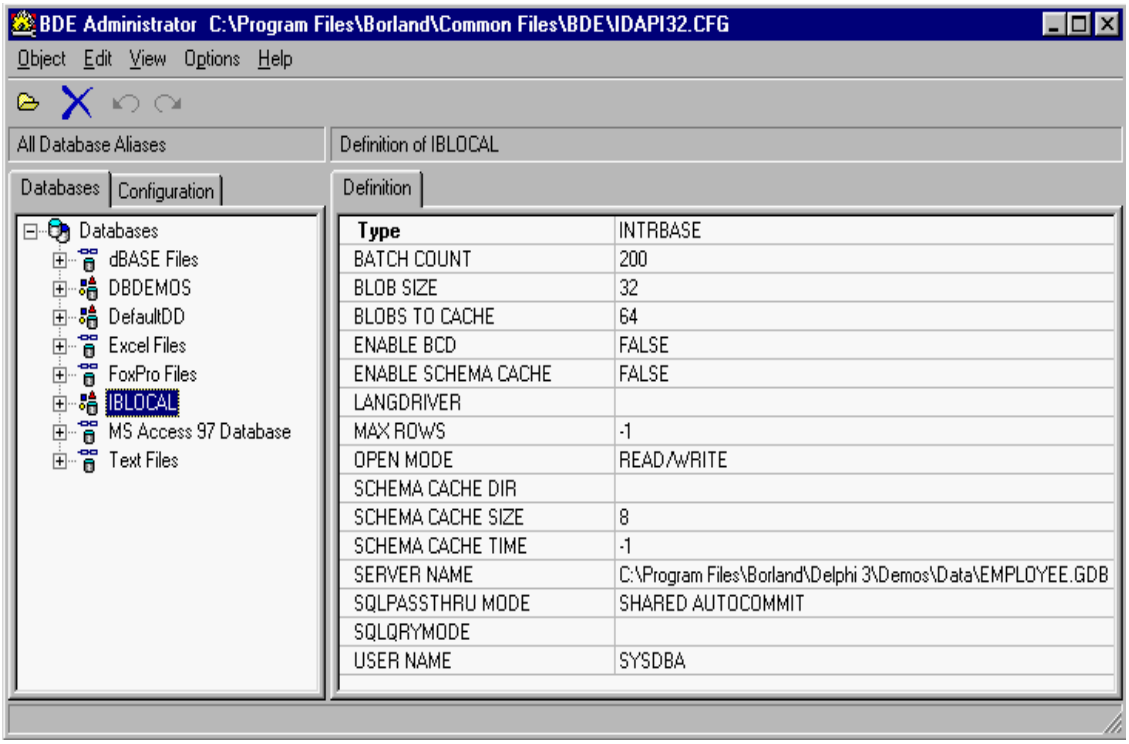


Рис.3

На странице **DataBases** представлены псевдонимы зарегистрированных баз данных. Имеющиеся в распоряжении пользователя псевдонимы можно редактировать, также можно создавать новые псевдонимы баз данных. Использование псевдонима позволяет обращаться к базе данных по имени, то есть без указания пути доступа к данным.

Для создания нового псевдонима необходимо выполнить следующие действия:

- Щелкните мышью на элементе **DataBases**.
- Выберите в меню **Object** или в контекстном меню команду **New**.
- Выберите в списке **DataBases Driver Name** диалогового окна **New DataBases Alias** необходимый драйвер.
- Введите в левой области окна **DBE Administrator** новый псевдоним.
- Щелкните на станице **Definition** в поле **Path** или в поле **Server Name** и затем нажмите на кнопку с многоточием. В открывшемся диалоговом окне **Select Directory** выберите путь для нового псевдонима или сервера.
- Введите путь непосредственно в поле **Path** или поле **Server Name.**

- При помощи правой кнопки мыши выберите в левой области окна **DBE Administrator** новый псевдоним и активизируете в контекстном меню команду **Apply**.

При выборе станицы **Configuration** на экран выводится список всех инсталлированных драйверов. На этой странице можно добавлять новые драйверы, а также сконфигурировать стандартные драйверы.

Страница Configuration DBE Administrator

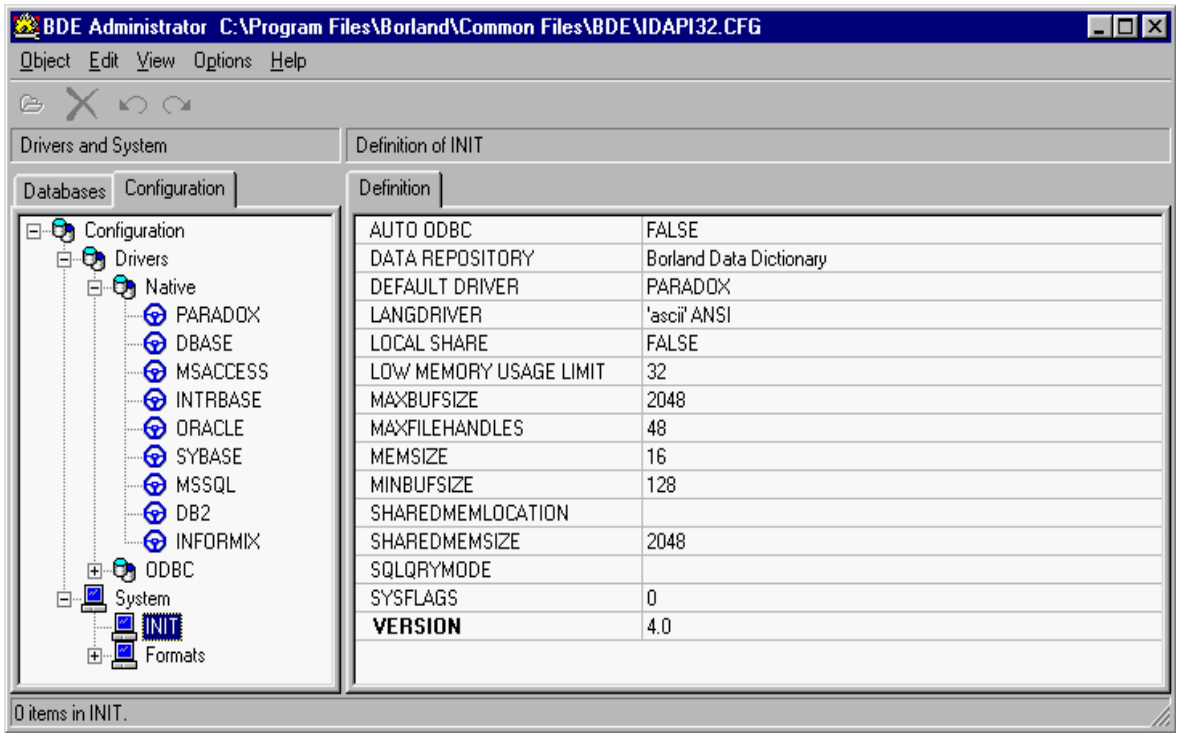


Рис.4

В окне **Definition** выводятся текущие установки выбранного драйвера. Щелкнув мышью на соответствующем поле можно модифицировать установки.

Посредством элемента **System/INIT** выбираются системные установки, которые сохраняются в файле **WINDOWS REGISTRY**. Модифицирование полей **VERSION** и **SYSFLAGS** недопустимо.

Поле **AUTO ODBS** может принимать значения **true** или **false**. Если это поле имеет значение **true**, то загружаются все **ODBS** псевдонимы из файла **ODBS.INI**. По умолчанию, указанное поле имеет значение **false**.

В поле **DATA REPOSITORY** указывается имя текущего словаря данных.

В поле **DEFAULT DRIVER** указывается драйвер, который используется при открытии базы данных.

В поле **LANGDRIVER** указывается используемый драйвер языка. Драйвер языка выбирается в открывшемся списке.

Поле **LOCAL SHARE** может принимать значения **true** или **false**. При одновременной обработке одних и тех же файлов **DBE** приложением и обычным приложением, значение этого поля следует должно быть установлено в **true**.

Поле **LOW MEMORY USAGE LIMIT** определяет объем нижней области памяти (до 640 КБ), которую использует **DBE**. Это значение по умолчанию установлено равным 32 КБ. Для операционных систем **Windows** это значение безразлично, так как используется **float memory model**, в которой границы между 640 КБ и остальной памятью исчезают.

Поле **MAXBUFSIZE** содержит максимальное значение КЭШа базы данных. Это значение должно быть больше значения указанного в поле **MINBUFSIZE**. По умолчанию, это значение равно 2048 КБ. Данное значение можно устанавливать кратным 128.

В поле **MAXFILEHANDLES** устанавливается максимальное число файлов, которые могут быть открыты в **DBE**. Это значение должно находится в промежутке между 5 и 4096. Чем больше данное значение, тем большими возможностями обладает **DBE**, однако это требует больших ресурсов.

В поле **MEMSIZE** указывается максимальный размер оперативной памяти, который может использоваться **DBE**.

В поле **MINBUFSIZE** указывается минимальный объем КЭШа для базы данных. Значение должно находится между 32КБ и 65535КБ.

В поле **SHAREDMEMSIZE** указывается максимальный размер оперативной памяти для совместного использования ресурсов. По умолчанию значение этого поля равно 2048КБ. Если приложение использует большое количество драйверов, таблиц, а также системных и клиентских объектов, значение этого поля следует увеличить.

Поле **SQLQRYMODE** определяет режим **SQL** запросов. Данное поле может принимать значения **nil**, **server** или **local**.

Посредством элемента **Formats/Date** определяется способ преобразования строковых значений в значения даты. Если поле **FOURDIGITYEAR** имеет значение **true**, то в этом случае для представления года используется четыре цифры (2004), а в противном случае – две (04). Если поле **FOURDIGITYEAR** имеет значение **false**, то в поле **YEARBIASED** можно указать, должно ли добавляться значение 2000 к двум цифрам для представления года.

Поля **LEADINGZEROM** и **LEADINGZEROD** определяют формат дня и месяца. Если эти поля имеют значения **false**, то в этом случае предшествующий нуль не добавляется к значениям дня и месяца (9.7.04).

Установка формата даты в окне DBE Administrator

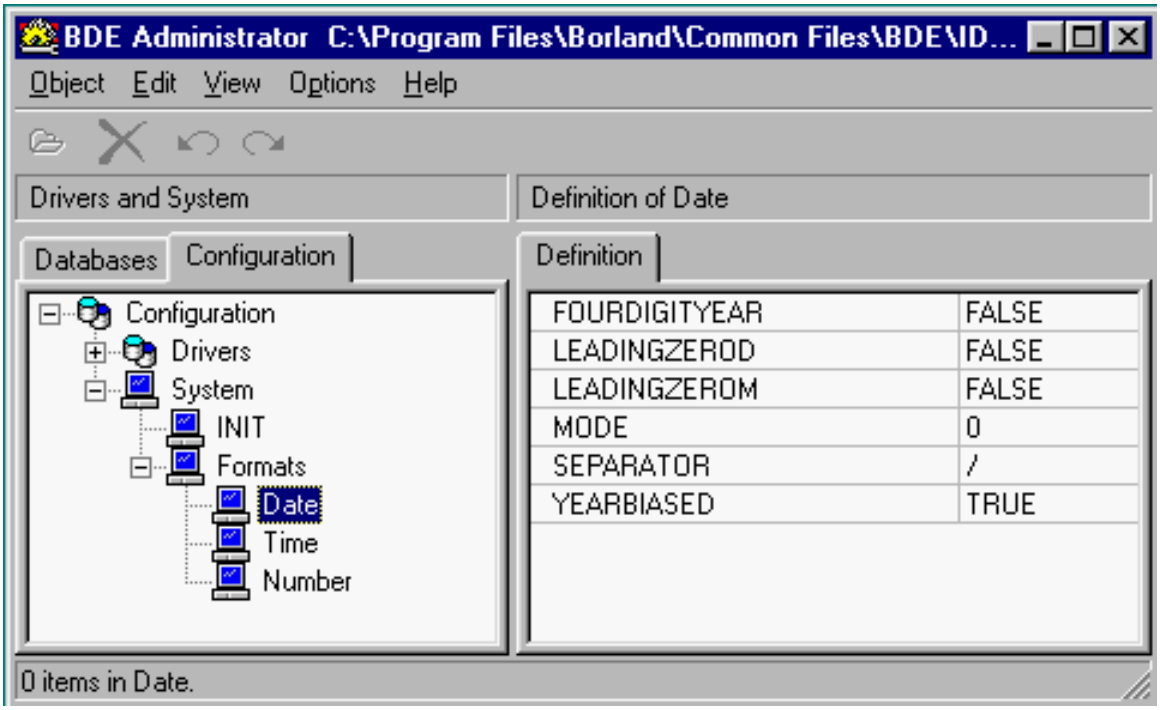


Рис.5

В поле MODE указывается, в какой последовательности должны выводится день, месяц и год. Это поле может принимать следующие значения:

- 0 – соответствует последовательности месяц, день, год;
- 1 – соответствует последовательности день, месяц, год;
- 2 - соответствует последовательности год месяц, день.

В поле **SEPARATOR** определяется разделитель между значениями дня, месяца и года.

Посредством элемента **Formats/Time** указывается способ преобразования строковых значений в значения системного времени. В том случае, если поле **TWELVEHOUR** имеет значение **true**, в полях **AMSTRING** и **PMSTRING** указывается, какие символы должны следовать за значением системного времени в интервале от 0 до 12 часов (**AMSTRING**) и соответственно от 12.01 до 23.59 (**PMSTRING**). По умолчанию принимаются символы **AM** и **PM**.

Поле **MILSECONDS** указывает, содержит ли системное время миллисекунды. Если данное поле имеет значение **true**, то формат вывода системного времени выглядит в следующем виде: 7:15:33:25.

Если поле **SECONDS** имеет значение **true**, то системное время содержит секунды.

Поле **TWELVEHOUR** определяет представление системного времени в виде двенадцати либо двадцати четырех часового формата. Если поле имеет значение true, то используется двенадцати часовый формат.

Посредством элемента **Format/Number** указывается, каким образом, строковые значения должны преобразовываться в числовые значения.

Установка формата времени в окне DBE Administrator

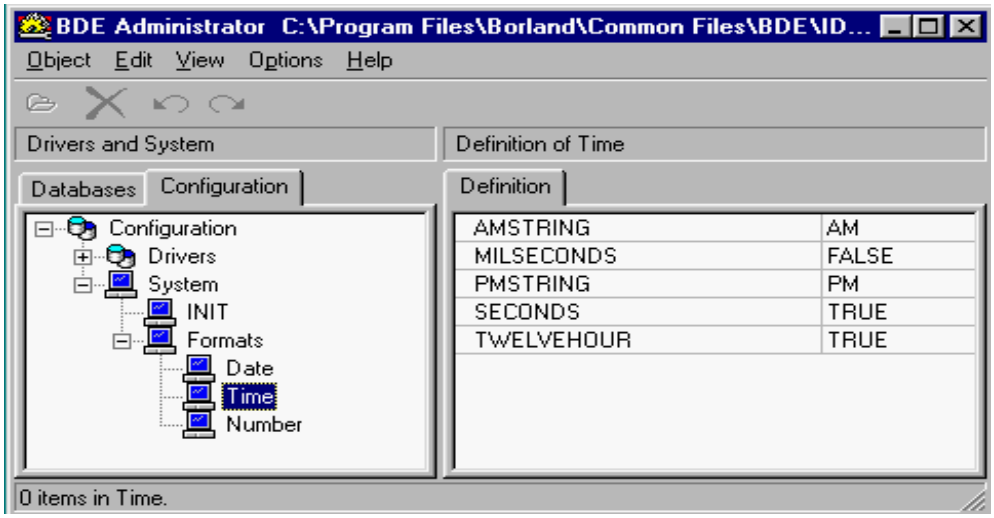


Рис.6

Поле **DECIMALDIGITS** определяет максимальное число десятичных разрядов, выводимых при преобразовании строки в числовое значение.

В поле **DECIMALSEPARATOR** определяется символ десятичного разделителя.

Установка формата чисел в окне DBE Administrator

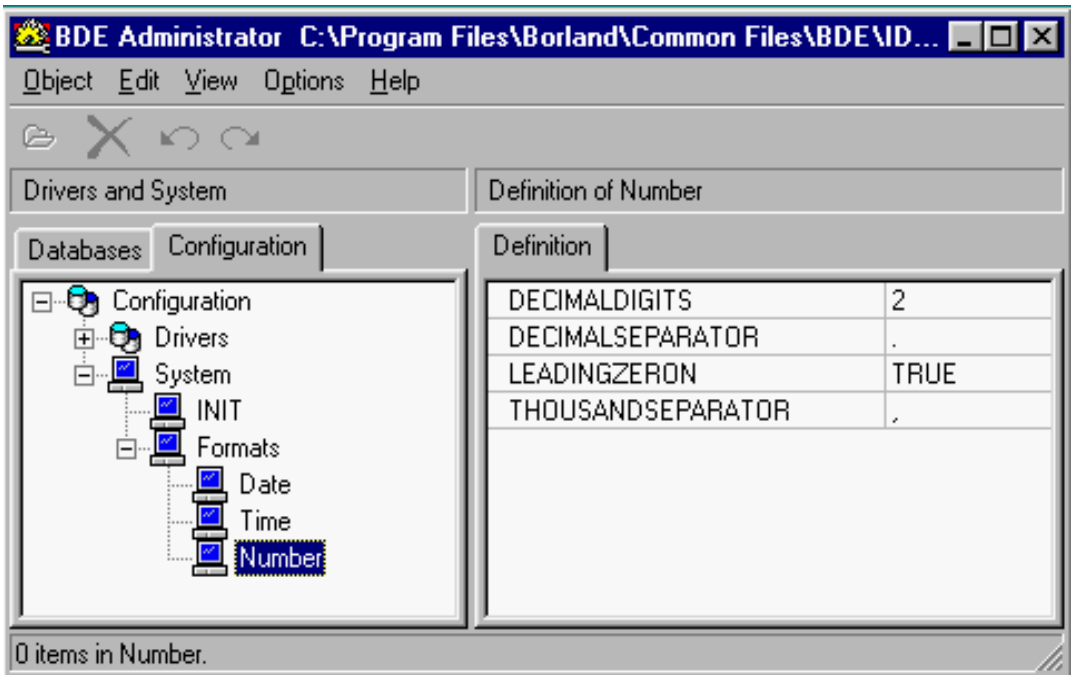


Рис.7

В поле **LEADINZERON** определяется, будет ли использоваться предшествующий нуль для значений в интервале от –1 до +1.

В поле **THOUSANDSEPARATOR** указывается символ разделителя разрядов.

2.3 Утилита Database Desktop

При помощи утилиты можно создавать и редактировать базы данных в формате **dBASE** и **Paradox**, а также выполнять **SQL** запросы. Данная утилита позволяет редактировать все поля данных, за исключением **BLOB** полей.

Утилита запускается из программной группы среды **Delphi**. При первом запуске программы следует указать рабочие каталоги. Определение рабочих каталогов выполняется посредством команд **File/Working Directory** и **File/Private Directory**.

Создание новой таблицы выполняется при помощи команды **File/New/Table**. После выполнения этой команды появится диалоговое

окно **Create Table**, в поле списка которого выбирается тип таблицы (рис.8).

Создание новой таблицы

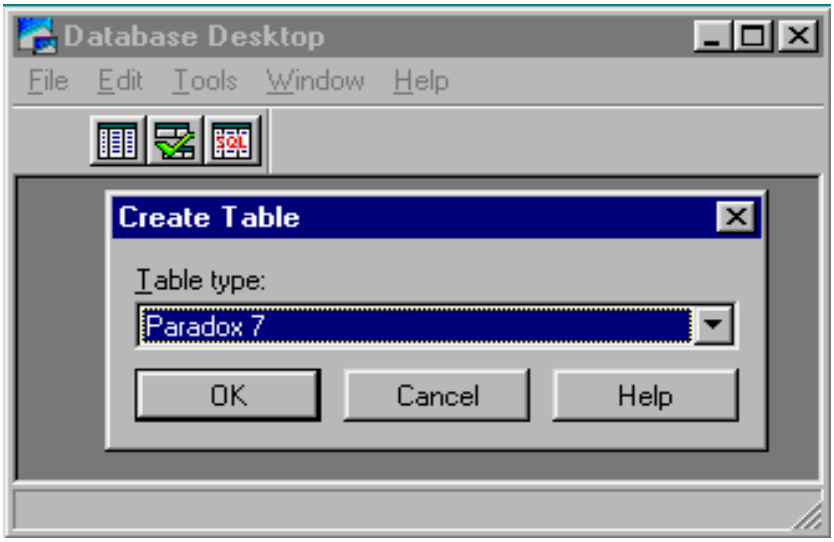


Рис.8

Поля новой таблицы определяются в области **Field Roster**, а именно в столбцах вводятся: **Field Name**, **Type**, **Size** и **Key**. Область **Table Properties** используется для выбора значений индексов и драйвера языка таблицы.

Структура таблицы Biolife

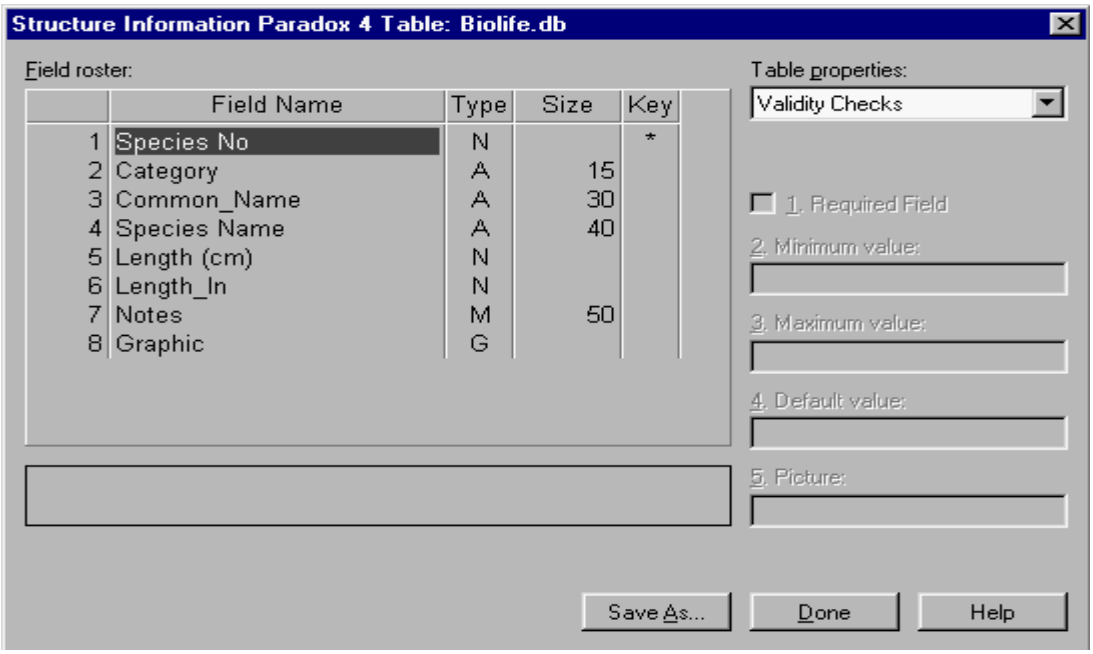


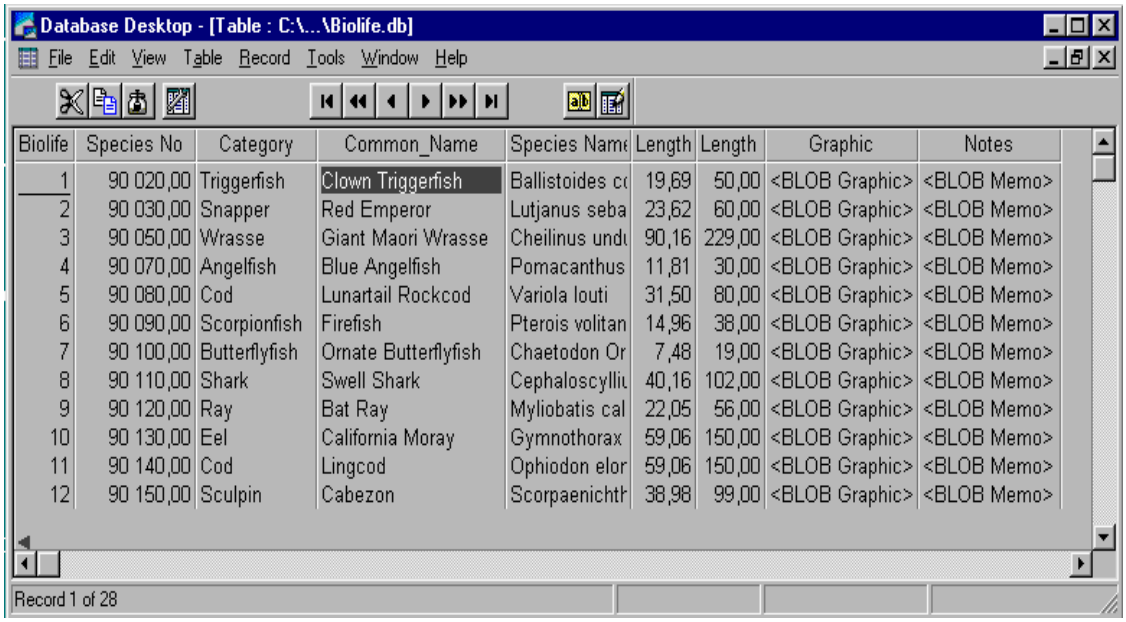
Рис.9

На рис.9 представлена структура таблицы **Biolife**, созданная в формате **Paradox 7**. Данная таблица содержит цифровые поля (**N**), строковые поля (**A**) и двоичные поля (**M**) и (**G**) преднозначенные для сохранения текста и изображения. Первое поле таблицы определено как автоинкрементное (*).

Для работы с новой таблицей, ее следует сохранить, используя команду **Save As**. Вывод на экран таблицы выполняется командой **File/Open/Table**.

На рис.10 представлена предварительно открытая заполненная таблица **Biolife**. Редактирование данных полей таблицы, за исключением **BLOB** полей, выполняется командами меню **Edit**.

Заполненные поля данных таблицы Biolife



Biolife	Species No	Category	Common_Name	Species Name	Length	Length	Graphic	Notes
1	90 020,00	Triggerfish	Clown Triggerfish	Ballistoides c	19,69	50,00	<BLOB Graphic>	<BLOB Memo>
2	90 030,00	Snapper	Red Emperor	Lutjanus seba	23,62	60,00	<BLOB Graphic>	<BLOB Memo>
3	90 050,00	Wrasse	Giant Maori Wrasse	Cheilinus undu	90,16	229,00	<BLOB Graphic>	<BLOB Memo>
4	90 070,00	Angelfish	Blue Angelfish	Pomacanthus	11,81	30,00	<BLOB Graphic>	<BLOB Memo>
5	90 080,00	Cod	Lunartail Rockcod	Variola louti	31,50	80,00	<BLOB Graphic>	<BLOB Memo>
6	90 090,00	Scorpionfish	Firefish	Pterois volitan	14,96	38,00	<BLOB Graphic>	<BLOB Memo>
7	90 100,00	Butterflyfish	Ornate Butterflyfish	Chaetodon Or	7,48	19,00	<BLOB Graphic>	<BLOB Memo>
8	90 110,00	Shark	Swell Shark	Cephaloscyllu	40,16	102,00	<BLOB Graphic>	<BLOB Memo>
9	90 120,00	Ray	Bat Ray	Myliobatis cal	22,05	56,00	<BLOB Graphic>	<BLOB Memo>
10	90 130,00	Eel	California Moray	Gymnothorax	59,06	150,00	<BLOB Graphic>	<BLOB Memo>
11	90 140,00	Cod	Lingcod	Ophiodon elor	59,06	150,00	<BLOB Graphic>	<BLOB Memo>
12	90 150,00	Sculpin	Cabezon	Scorpaenichth	38,98	99,00	<BLOB Graphic>	<BLOB Memo>

Рис.10

Создание таблиц в форматах **Paradox** и **dBASE** выполняются по установленным правилам.

Имя поля в таблице формата **Paradox** представляет собой строку, написание которой подчиняется следующим правилам:

- имя поля может содержать не более 25 символов;
- имя поля не должно начинаться с пробела, но может содержать пробелы;
- имя поля не должно содержать квадратные, круглые или фигурные скобки, тире, а также знаки больше и меньше;

- имя поля не должно быть только символом #, хотя этот символ может присутствовать в имени среди других символов;
- не рекомендуется в имени поля использовать точку (.), так как она зарезервирована в Delphi для других целей.

Имя поля в таблице формата **dBase** представляет собой строку, написание которой подчиняется правилам, отличным от **Paradox**:

- Имя должно быть не длиннее 10 символов;
- пробелы в имени недопустимы.

Имена полей в формате **dBase** подчиняются более строгим правилам, чем имена полей в формате **Paradox**. При совместном использовании платформ **dBase** и **Paradox** рекомендуется присваивать мена полей в формате **Paradox** по правилам формата **dBase**.

Поля таблиц формата **Paradox** могут иметь следующий тип:

- **Alpha** – строка длиной 1-255 байт, содержащая любые печатаемые символы.
- **Number** – числовое поле длиной 8 байт, значение которого может быть положительным и отрицательным. Диапазон чисел представляется от 10^{-308} до 10^{308} с 15 значащими цифрами.
- **$ (Money)** – числовое поле, значение которого может быть положительным и отрицательным. По умолчанию, данное поле форматировано для отображения десятичной точки и денежного знака.
- **Short** – числовое поле длиной 2 байта, которое может содержать только целые числа в диапазоне от -32768 до 32767.
- **Long Integer** – числовое поле длиной 4 байта, которое может содержать целые числа в диапазоне от -2147483648 до 2147483648.
- **# (BCD)** – числовое поле, содержащее данные в формате **BCD** (**Binary Coded Decimal**). Скорость вычислений значений в данном формате немного меньше, чем в других числовых форматах, однако, точность вычислений значительно выше. Поле может содержать 0-32 знака после десятичной точки.
- **Date** – поле даты длиной 4 байта, которое может содержать дату от 1 января 9999 г. до нашей эры – до 31 декабря 9999 г. нашей эры. Корректно обрабатывает високосные года и имеет встроенный механизм проверки правильности даты.

- **Time** – поле времени длиной 4 байта, содержит время в миллисекундах от полуночи и ограничено 24 часами.
- **@ (Timestamp)** – обобщенное поле даты длиной 8 байт - содержит и дату и время.
- **Memo** – поле для хранения текста. Может иметь любую длину. Размер, указываемый при создании таблицы, означает количество символов, сохраняемых в таблице (1-240) – остальные символы сохраняются в отдельном файле с расширением .**MB**.
- **Formatted Memo** – поле, аналогичное полю Memo, с добавлением возможности задавать шрифт текста. Также может иметь любую длину. При этом размер, указываемый при создании таблицы, означает количество символов, сохраняемых в таблице (0-240) – остальные символы сохраняются в отдельном файле с расширением .**MB**.
- **Graphic** – поле, содержащее графическую информацию. Может иметь любую длину. Смысл размера поля такой же, как и в **Formatted Memo**. **Database Desktop** позволяет создавать поля типа **Graphic**, однако заполнять их можно только в приложении.
- **OLE** – поле, содержащее **OLE (Object Linking and Embedding)** объекты: звук, видео, а также документы, которые для своей обработки вызывают создавшее их приложение. Данное поле может иметь любую длину. Смысл размера поля такой же, как и в **Formatted Memo**. **Database Desktop** позволяет создавать поля типа **OLE**, однако наполнять их можно только в приложении.
- **Logical** – поле длиной 1 байт, которое может содержать только два значения - **T** (true) или **F** (false). Допускаются строчные и прописные буквы.
- **(+) Autoincrement** – автоинкрементное поле длиной 4 байта, содержащее не редактируемое (read-only) значение типа: *long integer*. Значение этого поля для каждой новой записи автоматически увеличивается на единицу. Начальное значение поля соответствует 1. Применение этого поля удобно для создания уникального идентификатора записи.
- **Binary** – это поле, содержащее любую двоичную информацию. Может иметь произвольную длину. При этом размер, указываемый при

создании таблицы, означает количество символов, сохраняемых в таблице (0-240) – остальные символы сохраняются в отдельном файле с расширением .**MB**.

- **Bytes** – данное поле предназначено для хранения двоичной информации, представляет собой строку цифр длиной 1-255 байт.

Для ввода типа поля достаточно набрать только подчеркнутые символы.

В формате **dBase** поля таблиц могут иметь следующий тип:

- **Character (alpha)** – поле представляет собой строку длиной 1-254 байт, содержащую любые печатаемые символы.
- **Float (numeric)** – числовое поле размером 1-20 байт в формате с плавающей точкой, значение которого может быть положительным и отрицательным. Поле может содержать большие величины, однако следует иметь в виду ошибки округления, возникающие при работе с полем данного типа. Число цифр после десятичной точки (параметр **Dec**) должно быть по крайней мере на 2 меньше, чем размер всего поля, поскольку в общий размер включаются сама десятичная точка и знак.
- **Number (BCD)** – числовое поле размером 1-20 байт, содержащее данные в формате **BCD (Binary Coded Decimal**). Скорость вычислений значений данного поля немного меньше, чем скорость вычислений в других числовых форматах, при этом, точность вычислений значительно выше. Число цифр после десятичной точки (параметр **Dec**) также должно быть, по крайней мере, на 2 меньше чем размер всего поля, поскольку в общий размер включаются сама десятичная точка и знак.
- **Date** – поле даты длиной 8 байт. По умолчанию, используется формат короткой даты (**ShortDateFormat**).
- **Logical** – поле длиной 1 байт, которое может содержать только значения «**true**» или «**false**». Допускаются применение строчных и прописных букв. Также допускается применение букв «**Y**» и «**N**» (сокращение от **Yes** и **No**).
- **Memo** – поле для хранения символов, суммарная длина которых более 255 байт. Поле может иметь любую длину. Данное поле хранится в

отдельном файле. **Database Desktop** не обладает возможностью модифицировать данные в поле типа **Memo**.

- **OLE** – поле, содержащее **OLE** объекты (**Object Linking and Embedding**) – образы, звук, видео, документы – которые для своей обработки вызывают создавшее их приложение. Поле может иметь любую длину. Это поле также сохраняется в отдельном файле. **Database Desktop** обладает возможностью только создавать поля типа **OLE**, однако наполнять их можно только в приложении.
- **Binary** – поле, содержащее любую двоичную информацию. Может иметь любую длину. Данное поле сохраняется в отдельном файле с расширением .**DBT**.

Для ввода типа поля достаточно набрать только подчеркнутые символы.

Для таблиц в формате **Paradox** можно определить поля, составляющие первичный ключ, причем эти поля должны быть расположены в начале таблицы. Первое поле, входящее в ключ, должно быть первым полем в записи.

После создания таблицы, с ней можно связать некоторые свойства, перечень которых зависит от формата таблицы. Так, для таблиц формата **Paradox** можно задать:

- **Validity Checks** – это свойство проверяет минимальное и максимальное значение данных, а также значение по умолчанию. Кроме того, позволяет задать маску ввода.
- **Table Lookup** – данное свойство позволяет вводить значение в таблицу, используя уже существующее значение в другой таблице.
- **Secondary Indexes** – вторичные индексы. Создание вторичных индексов позволяет осуществлять доступ к данным в порядке, отличном от порядка, заданного первичным ключом.
- **Referential Integrity** – ссылочная целостность. Данное свойство позволяет задать связи между таблицами и поддерживать эти связи на уровне ядра базы данных. Как правило, **Referential Integrit** задается после создания всех таблиц в базе данных.
- **Password Security** – данное свойство позволяет закрыть таблицу паролем.

- **Table Language** – данное свойство предназначено для выбора языкового драйвера таблицы.

В таблицах формата **dBase** не существует первичных ключей. Однако, это обстоятельство можно преодолеть путем определения уникальных (**Unique**) и поддерживаемых (**Maintained**) индексов (**Indexes**). Кроме того, для таблиц формата **dBase** можно определить и язык таблицы (**Table Language**) т.е. установить языковый драйвер, управляющий сортировкой и отображением символьных данных.

Определения дополнительных свойств таблиц всех форматов доступны через кнопку «**Define**». Дополнительные свойства можно устанавливать не только при создании таблиц, но и для существующих таблиц. С этой целью в **Database Desktop** включены команды **Table|Restructure Table** (для открытой в данный момент таблицы) и **Utilities|Restructure** (для выбора таблицы). В том случае, если Вы попытаетесь изменить структуру или добавить новые свойства в таблицу, которая в данный момент уже используется другим приложением, **Database Desktop** выдаст сообщение об отказе, так как данная операция требует монопольного доступа к таблице. Тем не менее, все произведенные в структуре изменения сразу же начинают функционировать, если Вы определите ссылочную целостность для пары таблиц.

Database Desktop обладает возможностью создавать таблицу любого формата путем копирования структуры уже существующей таблицы. Для этого достаточно воспользоваться кнопкой «**Borrow**», которая расположена в левом нижнем углу окна формы. Появляющееся диалоговое окно позволит Вам выбрать существующую таблицу и включить/выключить дополнительные опции, совпадающие с уже перечисленными свойствами таблиц. Это наиболее легкий способ создания таблиц.

2.4 Утилита Database Explorer

Утилита **Database Explorer** (браузер баз данных) представляет собой вспомогательную программу, позволяющую выводить на экран структуру

базы данных и редактировать ее. При помощи данной утилиты можно также конфигурировать базы данных. Утилита **Database Explorer** может вызываться как из программной группы **Delphi**, так и из среды **Delphi**. Из среды **Delphi** утилита вызывается при помощи команды **Database/Explore** (рис.11).

Вид окна SQL Explorer

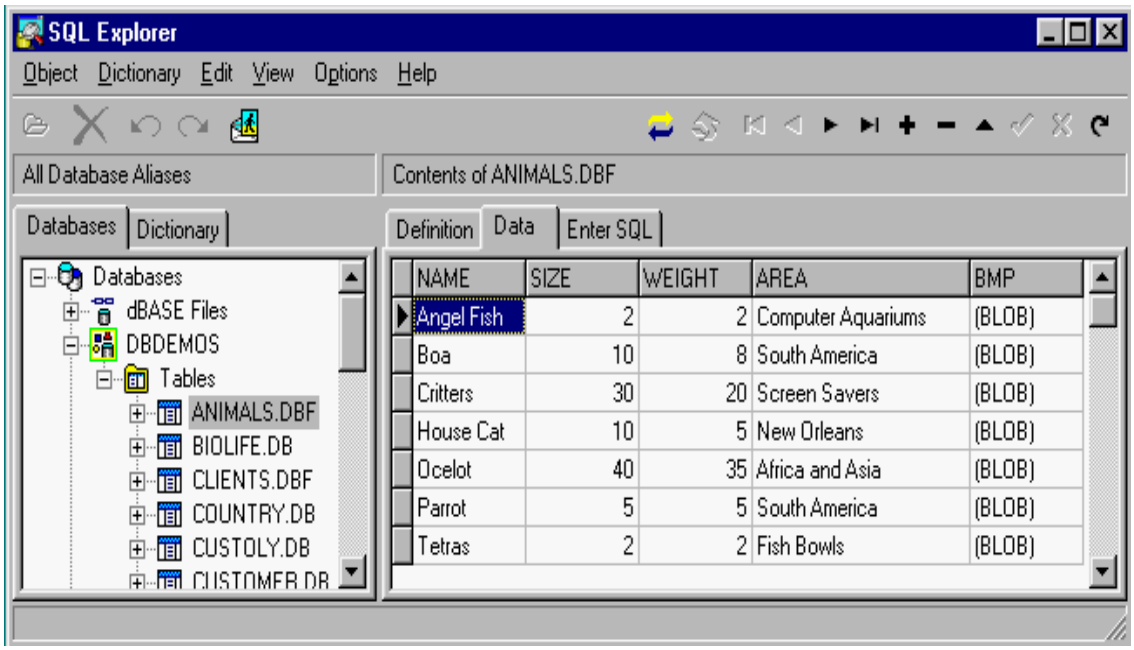


Рис.11

При помощи данной программы можно выводить и редактировать данные в таблицах баз данных, создавать псевдонимы баз данных и управлять ими, выполнять **SQL** запросы, а также создавать словари данных (**Data Dictionary**) и управлять ими. Утилита **Database Explorer** функционирует аналогично утилите **DBE Administrator**.

2.5 Мастер форм баз данных

С использованием мастера форм баз данных можно легко создавать формы обращения к таблицам внешних баз данных, таких как **InterBase**, **Paradox**, **dBASE** и **Oracle**. Мастер форм баз данных берет на себя задачи соединения компонентов формы с компонентами таблиц и запросов, а также определение последовательности активизации элементов управления.

Для создания простых приложений и приложений с архитектурой клиент/сервер необходимо выполнить определенную последовательность

действий. Для обоих видов приложений последовательность действий подобна и отличается незначительно. Наиболее полная последовательность действий выполняется при создании приложения с архитектурой клиент/сервер. Для создания такого приложения должен быть инсталлирован сервер **InterBase**.

Создание нового приложения с архитектурой клиент/сервер начинается с нового проекта в среде **Delphi**.

Далее при помощи команды **Data Bases/Form Wizard** запускается мастер форм баз данных.

В первом диалоговом окне мастера форм необходимо установить опции **Create a simple form** и **Create a form using TQuery object**s (рис.12). Последняя опция указывает, что обращение к базе данных будет выполняться при помощи **SQL** – запроса.

Первое диалоговое окно мастера форм баз данных

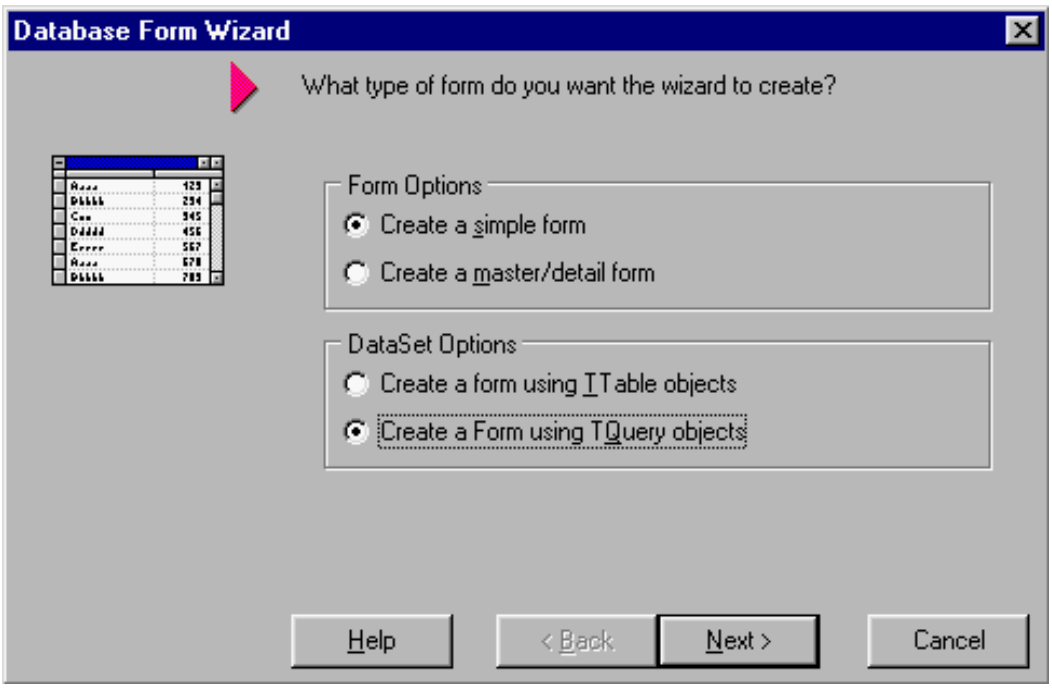


Рис.12

После нажатия на кнопку **Next** будет выведено второе диалоговое окно (рис. 12) в котором необходимо выбрать в поле списка **Drive or Alias Name** запись **IBLOCAL** для работы с сервером **InterBase**. В поле **Table Name** необходимо выбрать зарегистрированную таблицу. Если работа с сервером **InterBase** еще не производилась, то будет выведено диалоговое

окно **Database Login** (рис. 13). По умолчанию для доступа к базе данных используется имя пользователя **SYSDBA** и пароль **masterkey**.

Первое диалоговое окно мастера форм баз данных

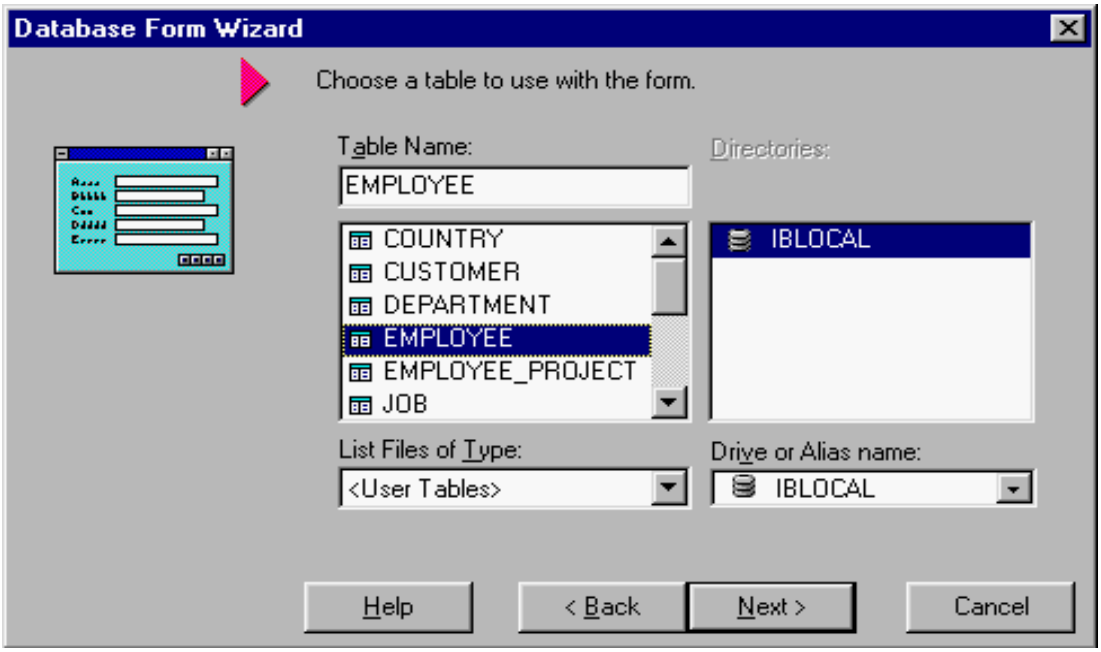


Рис.13

Диалоговое окно Database Login

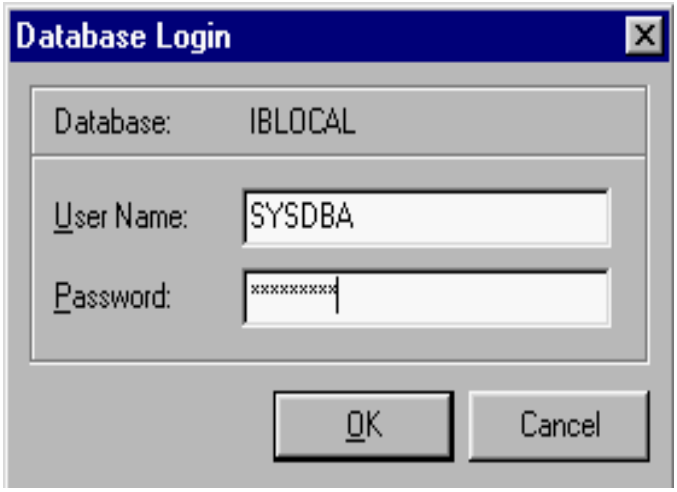


Рис.14

После нажатия на кнопку **Next** будет выведено третье диалоговое окно, в котором необходимо выбрать используемые поля таблицы. При

применении всех полей достаточно нажать на кнопку с двумя стрелками (рис.15).

Третье диалоговое окно мастера форм баз данных

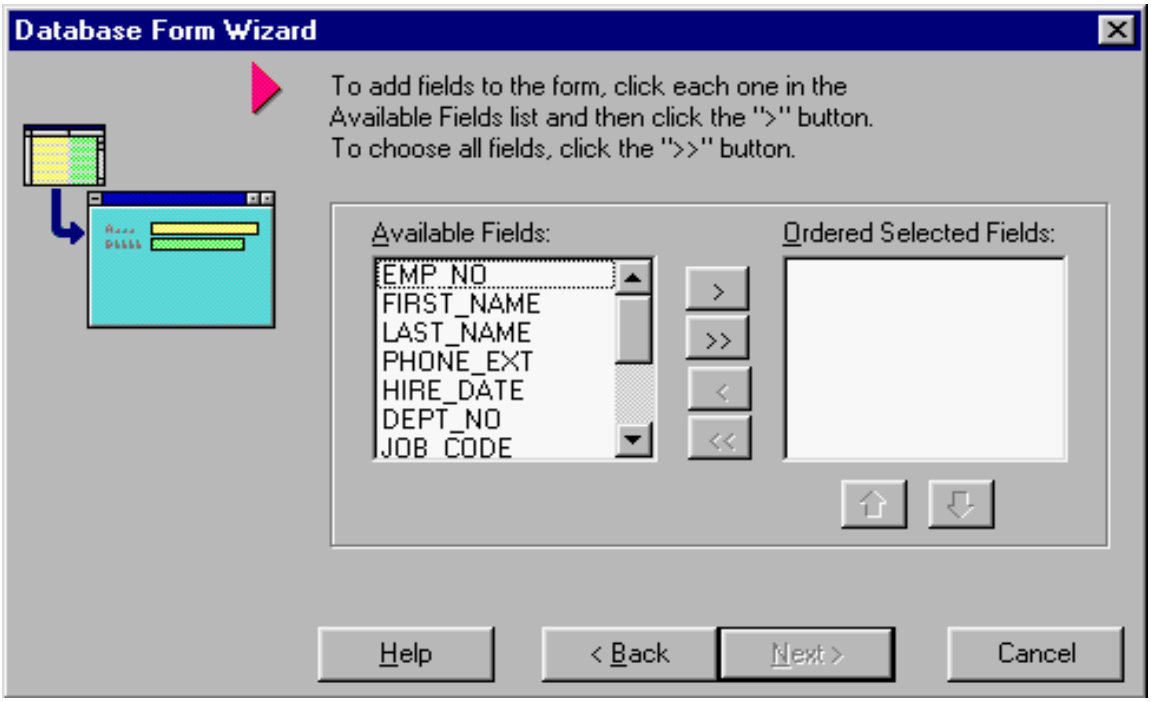


Рис.15

Выбор используемых полей

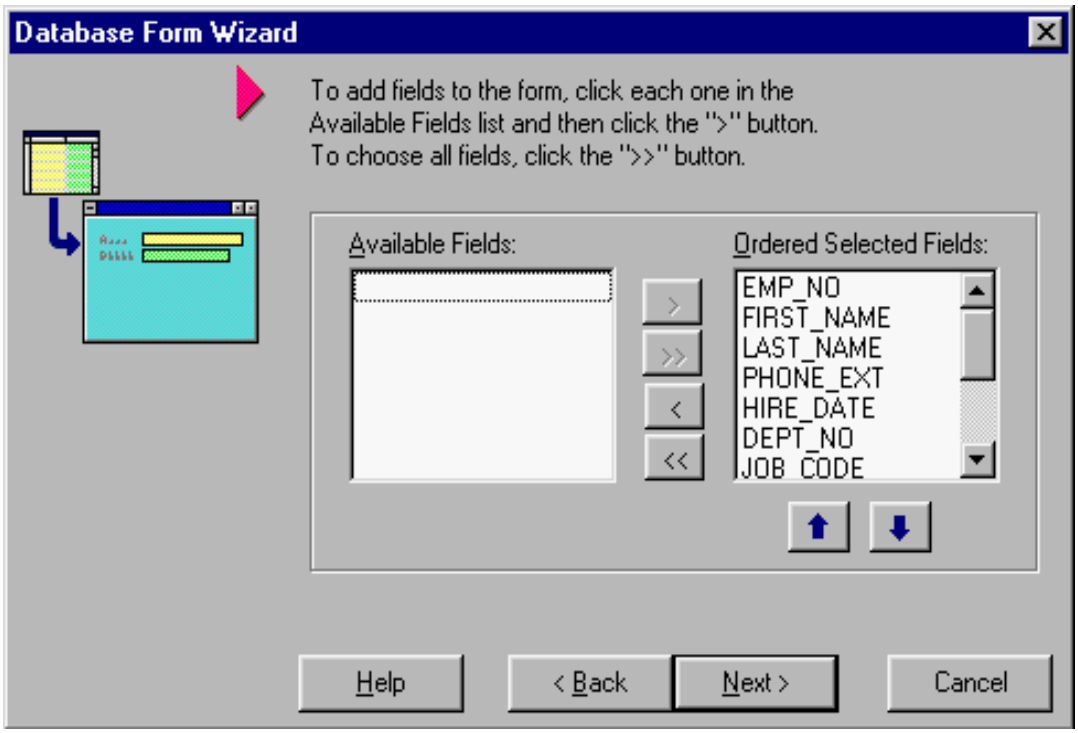


Рис.16

В четвертом диалоговом окне можно выбрать расположение компонентов управления данными в форме (рис.17).

Четвертое диалоговое окно мастера форм баз данных

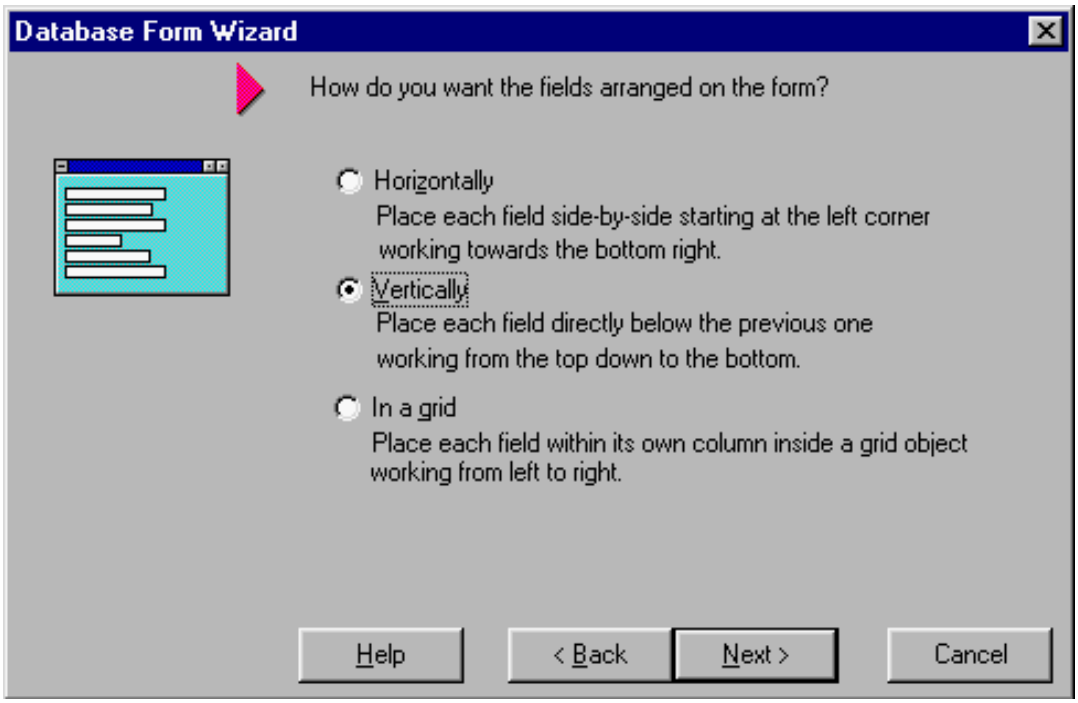


Рис.17

После нажатия на кнопку **Next** будет выведено диалоговое окно (рис.18), в котором задается расположение меток полей в форме.

Пятое диалоговое окно мастера форм баз данных

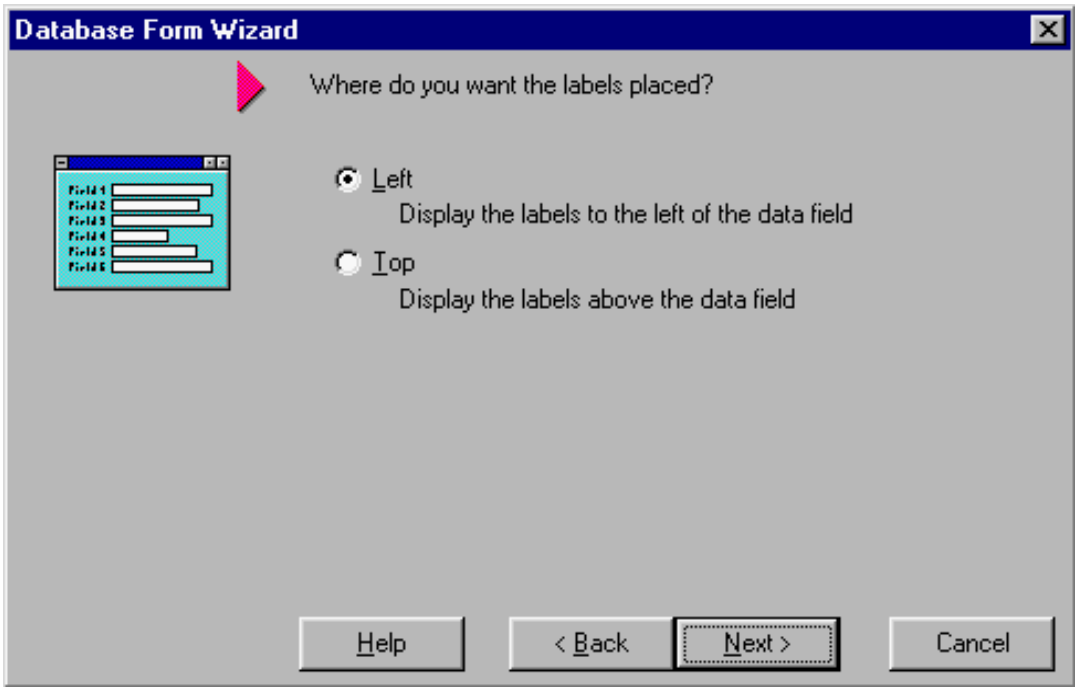


Рис.18

В завершении процесса создания формы в последнем диалоговом окне мастера форм определяется структура программы, которая может состоять раздельно из модуля формы и модуля данных или модуль данных будет включен в модуль формы (рис.19).

Шестое диалоговое окно мастера форм баз данных

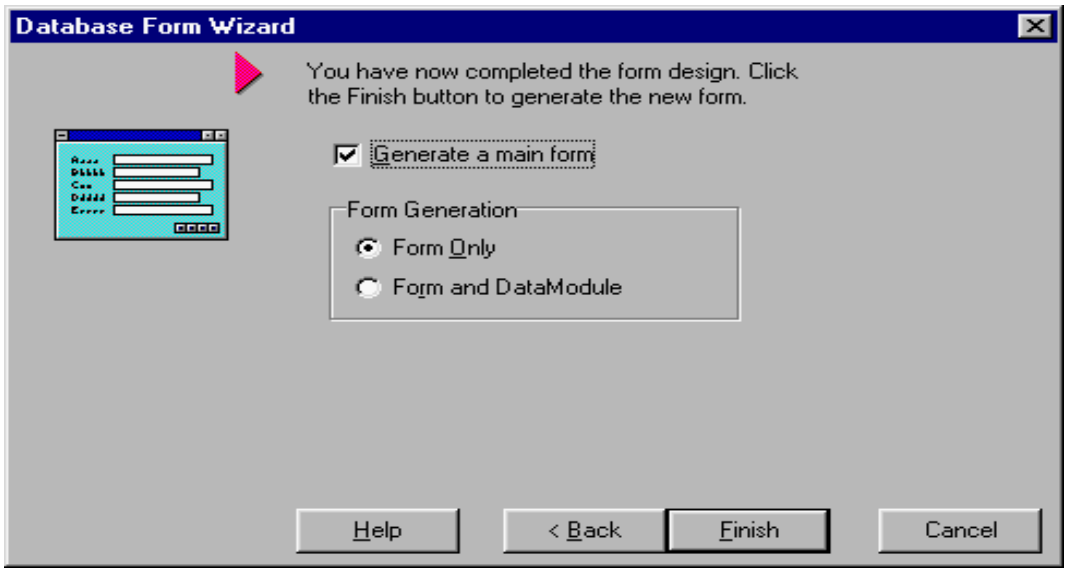


Рис.19

После нажатия на кнопку **Finish Delphi** создает новую форму приложения, которая имеет вид соответствующий примятым установкам (рис.20).

Форма созданная посредством программы Database Form Wizard

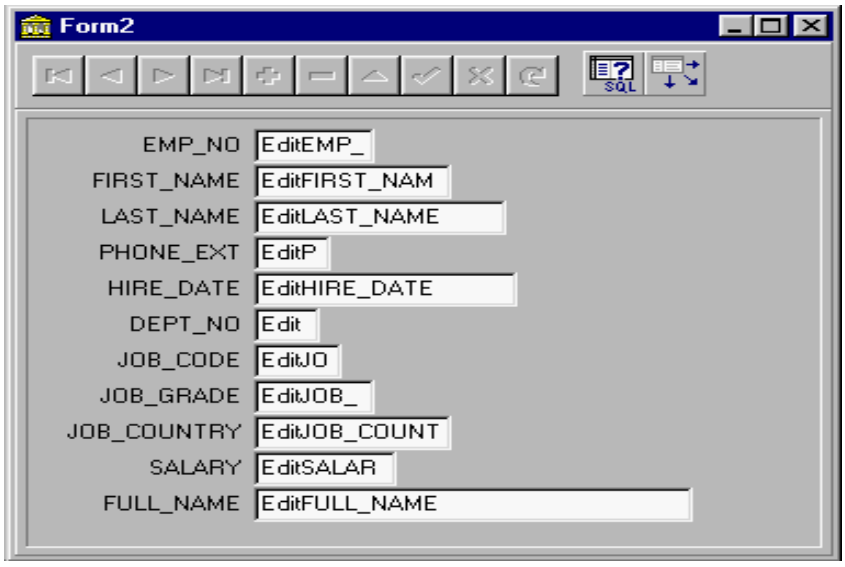


Рис.20

Дизайн формы можно изменить самостоятельно.

На рисунке 21 представлено запущенное приложение базы данных.

Из вышеизложенного видно, что мастер форм баз данных позволяет существенно сэкономить трудозатраты при разработке приложений баз данных.

Запущенное приложение, созданное при помощи Database Form Wizard

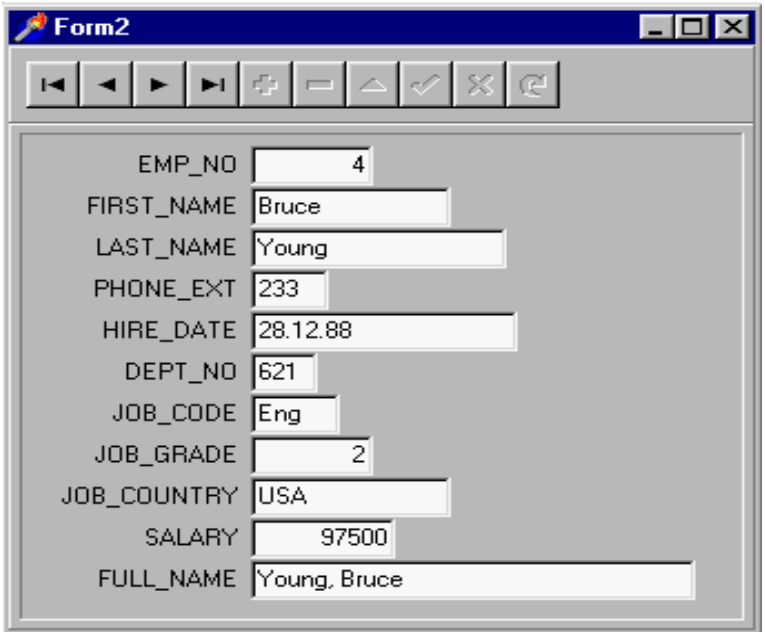


Рис.21

2.6 SQL сервер InterBase

SQL сервер InterBase представляет собой систему управления реляционными базами данных с использованием приложений архитектуры **клиент-сервер** произвольного масштаба от сетевой среды небольшой рабочей группы с сервером под управлением **Novell NetWare** или **Windows NT**, **Windows 2000** до информационных систем крупного предприятия на базе серверов **IBM**, **Hewlett-Packard, SUN** и других.

Основное различие между локальными системами управления базами данных, построенными на основе платформ **dBASE** и **Paradox**, и **SQL сервером InterBase** состоит в том, что с применением **InterBase** на одном персональном компьютере можно создавать полноценные приложения с многопользовательской архитектурой **клиент/сервер**. **InterBase** позволяет производить отладку приложений на одном персональном компьютере без использования полноценного **DBMS** сервера. Инсталляция соответствующей версии **InterBase** выполняется отдельно.

Работа с **InterBase** начинается с вызова **InterBase Server Manager**, из программной группы **InterBase**. Как известно, при разработке многопользовательского приложения типа клиент/сервер необходимо предусмотреть идентификации каждого пользователя. Для этого служит команда **Server Login** в меню **File** (рис.22).

Вид InterBase Saver Manager, после выполнения команды Server Login

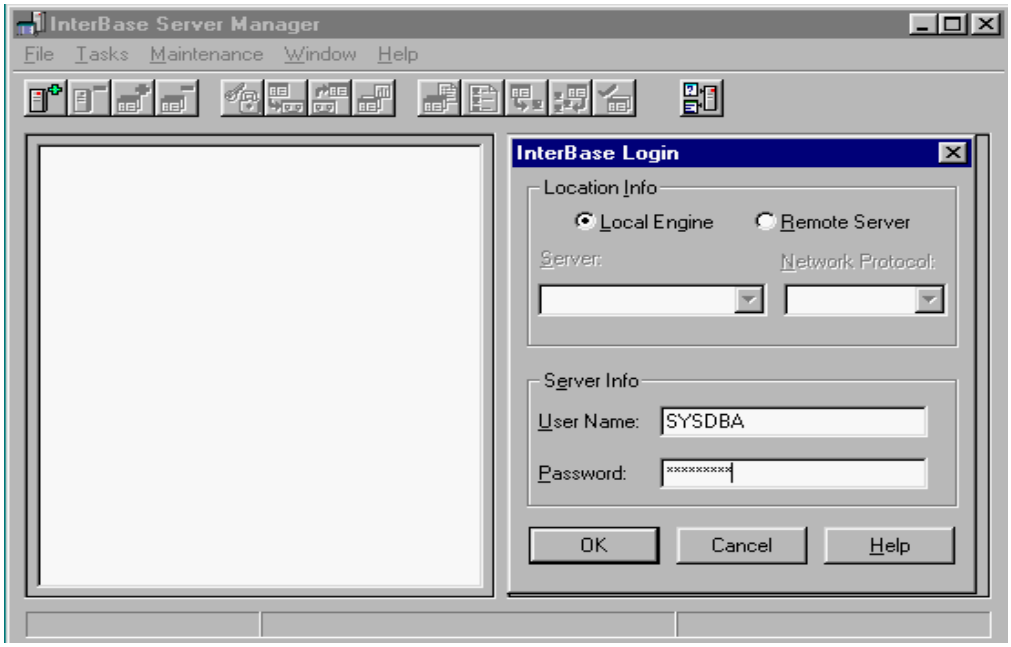


Рис.22

Процесс регистрации пользователя осуществляется следующим образом. При первом сеансе работы с **InterBase Server Manager** необходимо ввести в поле **User Name** стандартное имя **SYSDBA**, а в поле **Password** – стандартный пароль **masterkey**. Этот пароль в последующем можно будет изменить. После ввода указанных данных и выполнения щелчка по кнопке **OK**, будет выведено окно, показанное на рисунке 23.

Окно InterBase Saver Manager, после выполнения первой регистрации

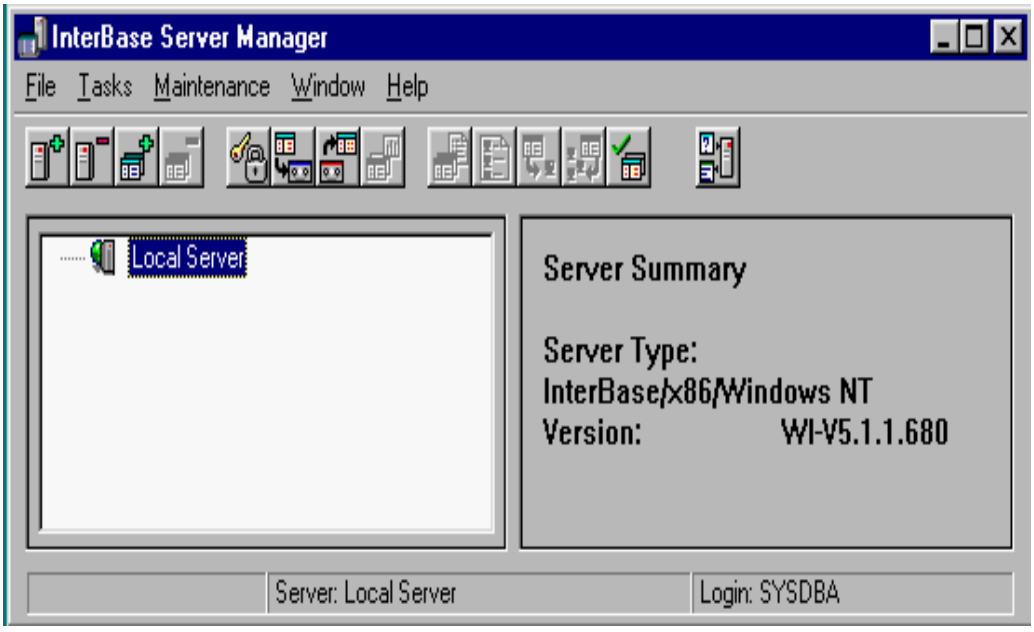


Рис.23

После выполнения первой регистрации, следует осуществить соединение с конкретной базой данных. Данное соединение выполняется посредством команды **File/Database Connect**, которая выводит диалоговое окно **Connect to Database** (рис. 24). Путь доступа к базе данных выбирается с поможью кнопки **Browse**. После установки соединения с базой данных недоступные ранее команды меню: **File**, **Tasks** и **Maintenance** становятся доступными (рис. 25).

Диалоговое окно Connect to Database

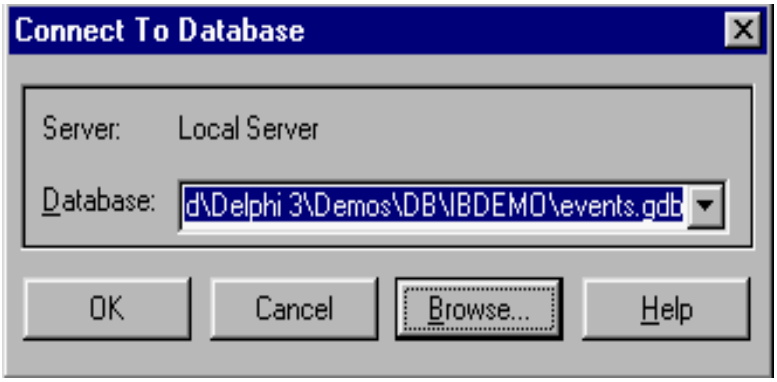


Рис.24

Окно InterBase Saver Manager, после установки соединения с базой данных

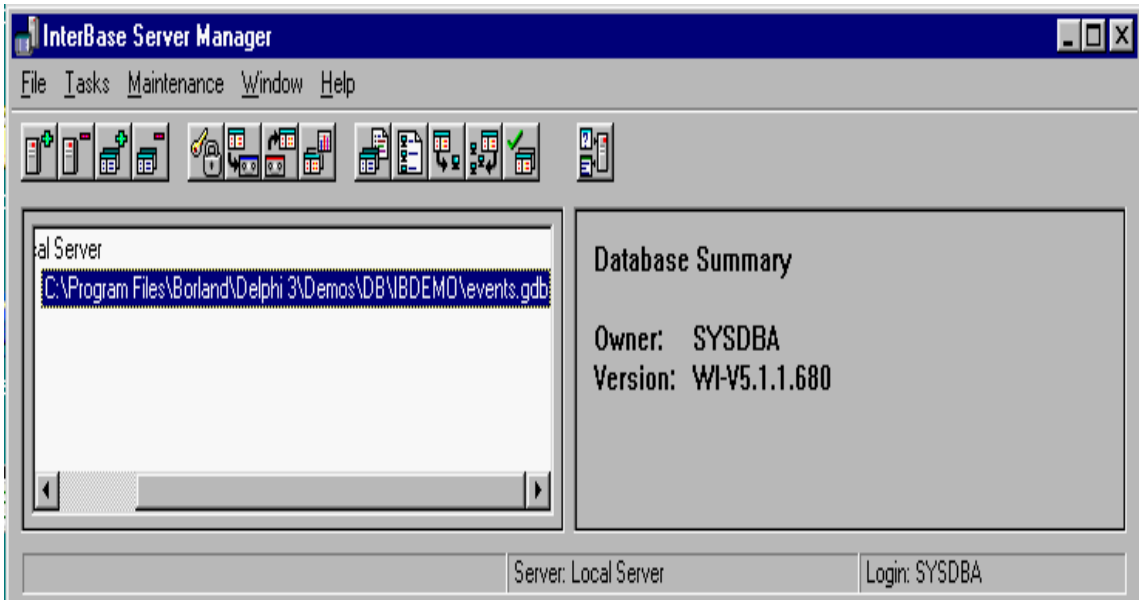


Рис.25

Команда **Database Disconnect** меню **File** предназначена для завержения работы с базой данных. В меню **Tasks** становится доступной опция **Database Statistics**, позволяющая получить сведения о текущей базе данных.

Для регистрации новых пользователей, модификации паролей и удаления регистрационных записей пользователей используется команда **Tasks/User Security**. В результате выполнения этой команды появится диалоговое окно **Interbase Security** (рис.26).

Диалоговое окно Interbase Security

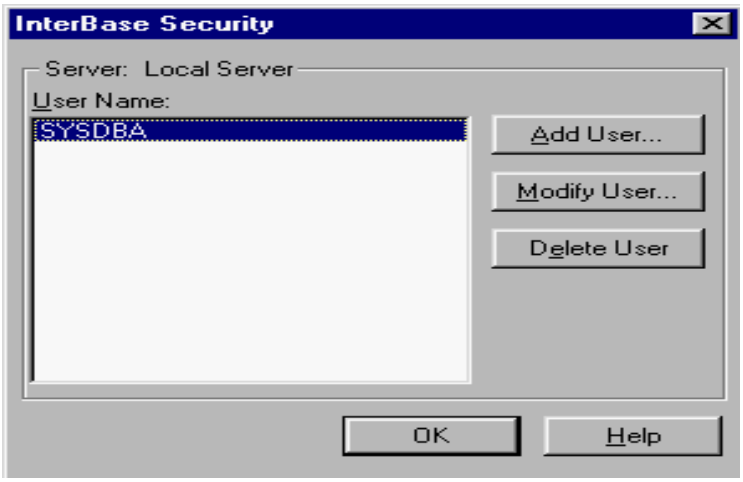


Рис.26

Регистрация, модификация и удаление пользователей выполняется выбором соответствующей кнопки в диалоговом окне **Interbase Security**. При выборе кнопки **Add User** будет выведено диалоговое окно **User Configuration** (рис. 15. 27), в котором необходимо заполнить соответствующие поля и нажать на кнопку **OK**.

Команды **Tasks/Backup** и **Tasks/Restore** позволяют создать резервную базу данных и восстановить исходную базу данных.

Команда **Task/Database Statistics**, открывает окно, в котором представлена подробная информация о текущей базе данных (рис.28).

Диалоговое окно User Configuration, используемое для регистрации новых пользователей

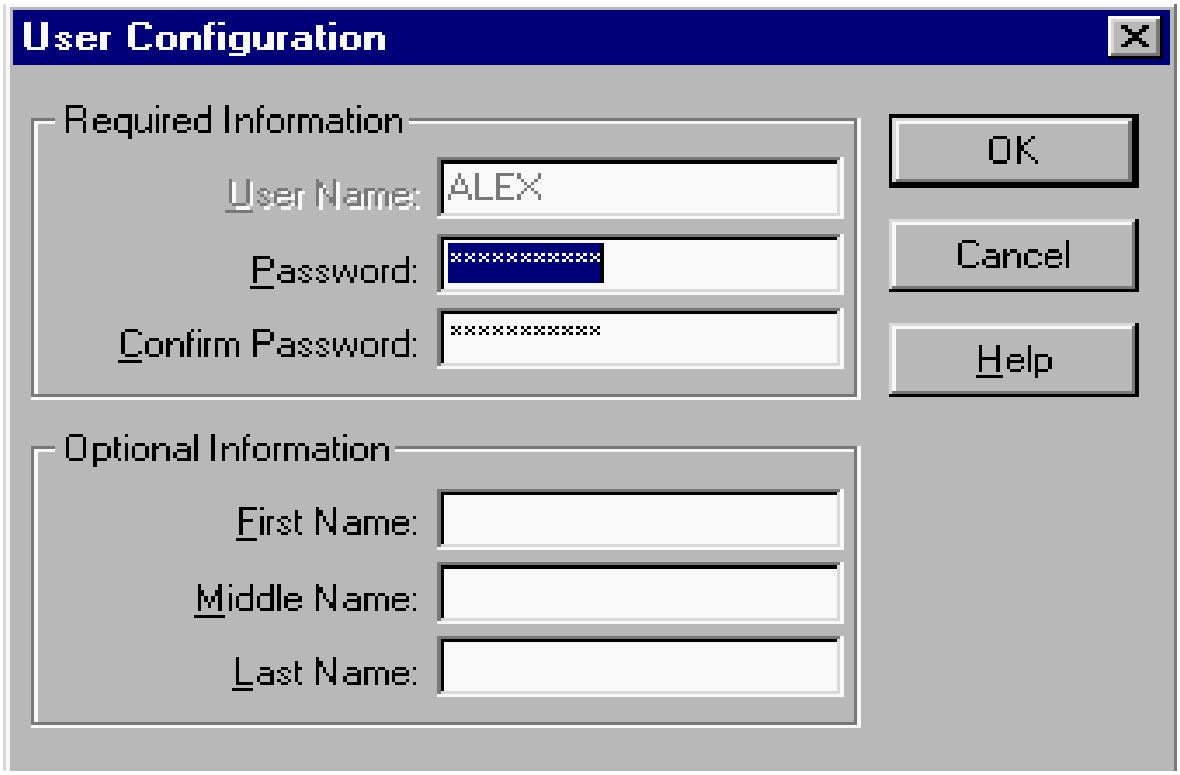


Рис.27

Окно Database Statistics

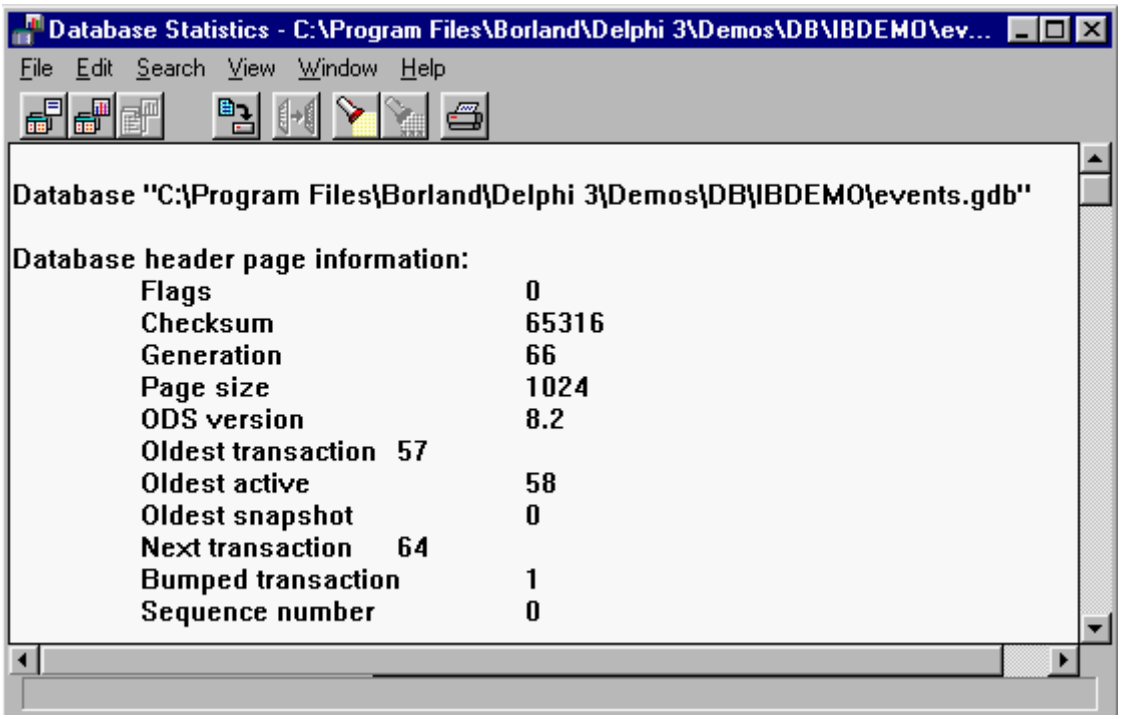


Рис.28

При помощи команды **Maintenance/Database Properties** выводится диалоговое окно, в котором указывается **Sweep Interval**. **Sweep Interval** является важным свойством базы данных. При управлении базой данных **Interbase** протоколирует все действия, предпринимаемые во время сессии. Это означает, что сохраняется несколько версий записи данных. Каждый раз, когда запись становится текущей или удаляется, сохраняются старая и новая версии записи. Вследствие этого, размер базы данных очень сильно увеличиваться. В результате выполнения операции **Sweep** все ненужные записи физически удаляются. После обновления, база данных сохраняется на жестком диске. Процесс обновления базы данных может выполняться автоматически. Для этого необходимо в поле **Sweep Interval** указать, после какого количества удаленных записей должна выполняться данная операция.

Используя **Interbase Server Manager** можно выполнять **SQL** команды для активной в данный момент базы данных. **SQL** команды вводятся в диалоговом окне **Interbase Interactive SQL**, которое вызывается посредством команды **Tasks/Interactive SQL**.

Структура таблиц в формате **Interbase** определяется строгими правилами:

- Имя поля должно быть не длиннее 31 символа.
- Имя поля должно начинаться с букв **A-Z, a-z**.
- Имя поля может содержать буквы **A – Z, a – z**, цифры, знак **$** и символ подчеркивания (_).
- Пробелы в именах таблиц и полей недопустимы.
- Для имен таблиц запрещается использовать зарезервированные слова **InterBase**.

Поля таблиц формата **InterBase** могут иметь следующий тип:

- **SHORT** – числовое поле длиной 2 байта, которое может содержать только целые числа в диапазоне от -32768 до 32767.
- **LONG** – числовое поле длиной 4 байта, которое может содержать целые числа в диапазоне от -2147483648 до 2147483648.
- **FLOAT** – числовое поле длиной 4 байта, значение которого может быть положительным и отрицательным. Диапазон чисел – от $3.4*10^{-38}$ до

$3.4*10^{38}$ с 7 значащими цифрами.

- **DOUBLE** – числовое поле длиной 8 байт, значение которого может быть положительным и отрицательным. Диапазон значений чисел представляется в пределах от $1.7*10^{-308}$ до $1.7*10^{308}$ и имеет 15 значащих разрядов.
- **CHAR** – строка символов фиксированной длины 0-32767 байт, содержащая любые печатаемые символы.
- **VARCHAR** – строка символов переменной длины 0-32767 байт, содержащая любые печатаемые символы.
- **DATE** – поле даты длиной 8 байт, значение которого может быть от 1 января 100 года до 11 декабря 5941 года.
- **BLOB** – поле, содержащее любую двоичную информацию. Может иметь любую длину.
- **ARRAY** – поле, содержащее массивы данных. **InterBase** позволяет определять массивы, имеющие размерность 16. Поле может иметь любую длину.
- **TEXT BLOB** – подтип **BLOB**-поля. **TEXT BLOB** содержит только текстовую информацию и может иметь любую длину.

СУБД **Interbase** представляет очень мощный и объемный инструмент, которому посвящен большой объем специальной литературы и полностью описать все возможности и способы работы в данной книге не представляется возможным.

Тема 3 Компоненты поддержки баз данных

Среда **Delphi** предоставляет в распоряжение пользователя компоненты, позволяющие получить доступ к базам данных и осуществить их редактирование. Компоненты, расположенные на станице **Data Access** политры компонентов предназначены для доступа к базам данных. Компоненты, расположенные на странице **Data Controls**, представляют собой элементы управления данными. Эти компоненты подобны компонентам расположенным на страницах **Standard** и **Additional**, однако

отличаются от них тем, что имеют свойства, обеспечивающие связь с полями таблицы базы данных.

Компоненты, обеспечивающие доступ к данным

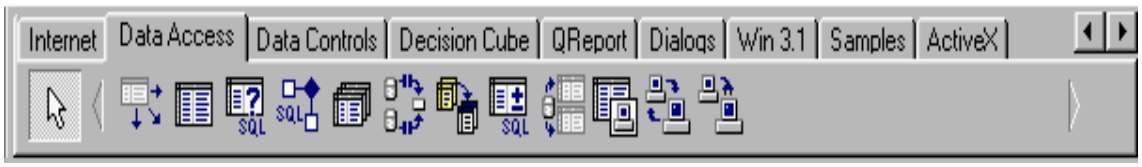


Рис.29

Компоненты, обеспечивающие управление данными

Рис.30

3.1 Компоненты доступа к данным

Соединение с базой данных обеспечивается посредством **DBE**. Связь между **DBE** и приложением осуществляется с помощью компонентов **TTable, TQuery**.

На рис.31 показаны установленные в форму компоненты, обеспечивающие доступ к данным. Эти компоненты являются невидимыми, т.е. не имеют свойств, описывающих внешний вид и положение в форме. Функцией компонентов доступа к данным является обеспечение работоспособности компонентов управления данными.

Компонент **TDataSource** обеспечивает интерфейс между компонентом набора данных и компонентами управления данными, т.е. данный компонент представляет собой программный код интерфейса осуществляющего связь между компонентами управления, такими как **TDBGrid**, **TDBNavigator**, **TDBText**, **TDBEdit** и др. и компонентами наборов данных **TTable**, **TQuery**, **TStoredProc**. В каждой форме должен размещаться, по меньшей мере, один компонент **TDataSource**, с которым связываются компоненты управления данными. В свойстве **DataSet** данного компонента должен указываться компонент набора данных, например, **TTable**. Свойство **AutoEdit** определяет, может ли компонент изменять данные. Свойство **Enabled** определяет, показываются ли данные в компонентах управления. Помимо

указанных свойств, компонент обладает свойством **State**, доступным только во время выполнения приложения и имеющим статус **Read Only**.

Компоненты доступа к данным

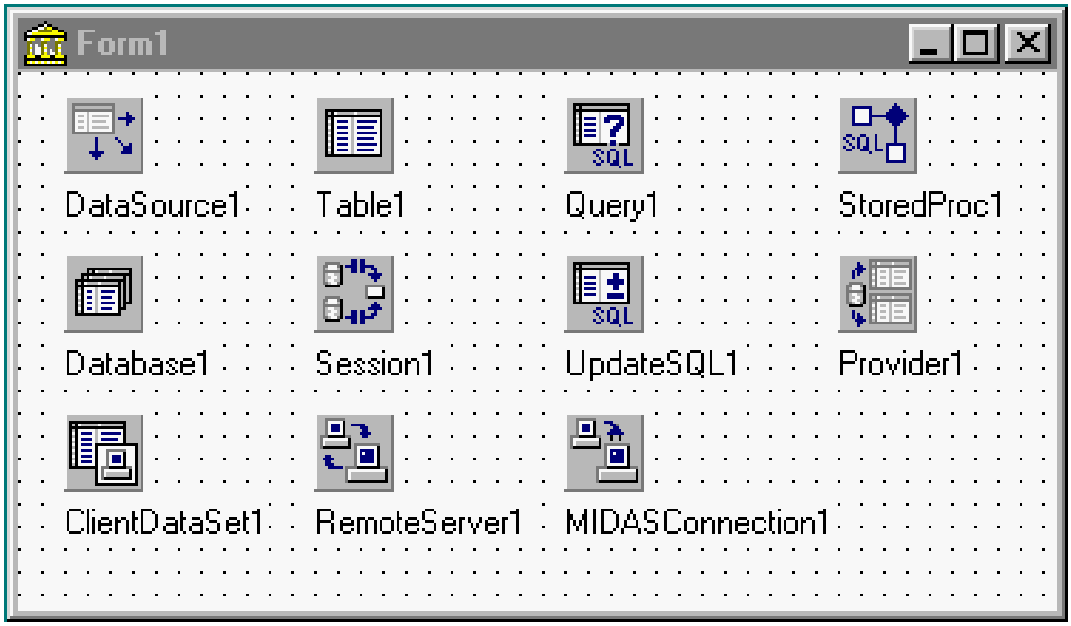


Рис.31

Компонент **TTable** представляет собой таблицу базы данных. Данный компонент функционирует как интерфейс между компонентом и **DBE**. Наиболее важными свойствами компонента являются:

- **Active** – данное свойство служит для получения доступа к таблице базы данных. Этому свойству присваивается значение true после установки нижеследующих свойств.
- **DataBaseName** – в данном свойстве указывается имя базы данных, доступ к таблице которой должен получить компонент **TTable**. Вместо имени базы данных можно указывать ее псевдоним или полный путь к каталогу, содержащему таблицы.
- **TableName** – в данном свойстве указывается конкретная таблица базы данных. При необходимости иметь в форме доступ к нескольким таблицам следует для каждой таблице определить свой компонент **TTable**.
- **Exclusive** – данное свойство определяет, могут ли несколько приложений обращаться одновременно к таблице базы данных. Если

необходимо заблокировать доступ к таблице со стороны другого приложения, то после открытия таблицы этому свойству надо присвоить значение **true**.

TableType – в данном свойстве указывается платформа базы данных. Обычно этому свойству присваивается значение **ttDefault**. Данное свойство не используется для таблиц удаленного **SQL** сервера.

Кроме **published** свойств, компонент **TTable** обладает свойствами, доступными только во время выполнения приложения. Основным из этих свойств является **FieldDefs**. Данное свойство содержит информацию о структуре таблицы, а именно имена типы и размеры полей таблицы. Это свойство не модифицируется. При необходимости обращения к полям набора данных следует использовать свойства **Fields**, **FieldsValues**, а также метод **FieldByName**.

Компонент **TTable** имеет также свойство **State**, которое совпадает с одноименным свойством **TDataSource**.

Компонент **TQuery** предназначен для доступа к базе данных посредством **SQL** запроса. Компонент **TQuery,** как и компонент **TTable** имеет свойство **DataBaseName**, но не имеет свойства **TableName**. Необходимая таблица создается автоматически при выполнении той или иной команды **SQL**. Свойству **SQL** присваивается текст одноименной команды, как при дизайне приложения, так и в процессе выполнения приложения. Для создания **SQL** команды в процессе дизайна приложения достаточно щелкнуть по кнопке расположенной рядом со свойством **SQL** в инспекторе объектов и в окне редактора команды ввести соответствующий текст **SQL** команды, например:

```
Select * from biolife
```

Данная команда указывает, что выбираются все поля таблицы **biolife**.

Эффективность компонентов **TTable** и **TQuery** зависит от используемой базы данных. Компоненты **TTable** обычно быстрее работают с локальными таблицами, а компоненты **TQuery** более эффективны при работе с **SQL** серверами. Компоненты **TQuery** рекомендуется использовать для создания сводной таблицы, включающих данные нескольких таблиц. Это

обусловлено тем, что компонент **TTable** всегда указывает на уже существующую таблицу базы данных, а компонент **TQuery** посредством **SQL** команды создает временную таблицу в результате выполнения селективного запроса.

Посредством компонента **TStoredProc** можно выполнять набор **SQL** команд на удаленном **SQL** сервере. Свойство **DataBaseName** должно содержать имя базы данных, а свойство **Params** предназначено для передачи необходимых параметров **SQL** команды.

Компонент **TDataBase** автоматически создается при выполнении приложения управления базами данных. Поэтому данный компонент не обязательно устанавливать в форму. Компонент используется для управления транзакциями, соединения с базой данных и доступом. Данный компонент преимущественно используется для соединения с удаленными базами данных в архитектуре клиент/сервер.

Управление процессом соединения с базой данных осуществляется посредством свойств **Connected** и **KeepConnection**. Оба эти свойства имеют тип **boolean**. Свойство **Connected** определяет, активно ли в данный момент соединение с базой данных, а свойство **KeepConnection** определяет, остается ли приложение соединенным с базой данных, если нет открытых таблиц.

В свойстве **DataBaseName** указываться имя базы данных. Значение свойства **AliasName** выбирается из списка зарегистрированных псевдонимов в **BDE Administrato**r.

Если свойству **LogionPrompt** присвоено значение true, пользователь при соединении с базой данных должен указывать имя пользователя и пароль.

Свойство **TransIsolation** определяет степень разграничения транзакций, т.е. изолирует одну транзакцию от воздействия другой. Транзакции выполняются через **BDE**. Для выполнения локальных транзакций с **Paradox** и **dBASE** свойству **TransIsolation** следует присвоить значение **tiDirtyRead**, что позволит избежать исключительной ситуации.

В свойстве **Params** указываются параметры для **BDE** псевдонима, например путь к каталогу базы данных. Если в свойстве **AliasName** указан

действительный псевдоним, то свойство **Params** будет содержать определенные для данного псевдонима параметры.

Компонент **TBatchMove** используется для многократного выполнения пакетов. Выполняемые пакетные операции определяются свойством **Mode**, которое может принимать следующие значения:

- **batAppend** – вставляет записи исходной таблицы **Source** в другую таблицу **Destination**;
- **batAppendUpdate** – заменяет соответствующие записи таблицы **Destination** на записи исходной таблицы **Source**. Если соответствующая запись отсутствует, то вставляется новая запись;
- **batCopy** – создает новую таблицу **Destination** с структурой исходной таблицы **Source**;
- **batDelete** – удаляет в таблице **Destination**, записи, соответствующие исходной таблице **Source**. Для выполнения этой операции таблица **Destination** должна быть проиндексирована;
- **batUpdate** – заменяет соответствующую запись в таблице **Destination** на запись из исходной таблицы **Source**. Для выполнения этой операции таблица **Destination** должна быть проиндексирована.

Свойство **Source** определяет, какая таблица является исходной для пакетной операции. В свойстве **Destination** указывается результирующая таблица.

В свойстве **Mapping** указываются названия полей, с которыми выполняется пакетная операция.

Остальные свойства влияют главным образом на действия, выполняемые с таблицей, и на обработку исключительных ситуаций.

Компонент **TSession** служит для глобального управления в приложении соединениями с базами данных. Свойство **Active** определяет, должно ли быть активным соединение с базой данных (сессия). Если данному свойству присваивается значение **true**, то данная сессия становится текущей, вызывается обработчик события **OnStartup** сессии, а также инициализируются свойства **NetFileDir**, **PrivateDir** и **ConfigMode**. В свойстве указывается каталог, в котором храниться управляющий сетевой

файл **BDE – PROXUSRS.NET**. Свойство **PrivateDir** предназначено для хранения имени каталога рабочих файлов **BDE**.

Посредством компонента **TUpdateSql** обеспечивается возможность выполнения **SQL** команд для различных операций обновления набора данных компонента **TQuery**. В свойствах **DeleteSQL**, **InsertSQL** и **ModifiSQL** можно определить необходимые команды **SQL Delete**, **Insert** и **Update** соответственно.

3.2 Компоненты управления данными

Компоненты управления данными подобны компонентам управления находящимся на страницах **Standard** и **Additional** инспектора объектов. Помимо одноименных свойств эти компоненты обладают важным свойством **DataSource**, посредством которого осуществляется связь с источником данных. Компоненты, предназначенные для отображения и редактирования полей таблиц, имеют свойство **DataField**, которое устанавливает связь с выбранным полем.

Компонент **TDBGrid** позволяет представить таблицу базы данных в виде похожем на электронную таблицу.

Компонент **TDBNavigator** представляет собой кнопочный переключатель, посредством которого можно перемещать курсор по записям таблицы и выполнять редактирование записей.

Компонент **TDBText** подобен компоненту **TLabel.**

Компонент **TDBEdit** выполняет ту же функцию, что и компонент **TEdit**.

Компоненты **TDBMemo**, **TDBRichEdit** и **TDBImage** предназначены для отображения и редактирования текстовой и графической информации, содержащейся в **Blob** полях базы данных.

Компонент **TDBListBox** выполняет ту же функцию, что и **TListBox**, а именно осуществляет отображение списка, в котором пользователь может выбрать требуемое значение. Содержимое компонента выводится только в том случае, если в свойстве **Items** был задан список возможных значений.

Компонент **TDBComboBox** аналогичен компоненту **TComboBox**. Посредством этого компонента пользователь получает возможность выбрать из списка необходимое значение или ввести его в поле ввода.

Компонент **TDBCheckBox** выполняет ту же функцию, что и компонент **TCheckBox**, однако источником данных для этого компонента является поле таблицы базы данных.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

Г. Г. Швачич, О. В. Овсяніков, В. В. Кузьменко, Н. І. Нєчаєва

СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ

Частина2

Затверджено на засіданні Вченої ради академії
як конспект лекцій

Дніпропетровськ НМетАУ 2007

УДК 004 (075.8)
Г.Г. Швачич, О.В. Овсянніков, В.В. Кузьменко, Н.І. Нєчаєва. Системи управління базами даних. Конспект лекцій. Частина2. – Дніпропетровськ: – НМетАУ, 2007. – 48 с.

Викладені основи створення інформаційно-логічних моделей та конструювання систем управління базами даних у середовищі робзобки прикладниого програмного забезпечення Delphi.

Призначений для студентів спеціальності 6.020100 – документознавство та інформаційна діяльність.

Іл. 31. Бібліогр.: 5 найм.

Відповідальний за випуск Г.Г. Швачич, канд. техн. наук, проф.

Рецензенти: Б. И. Мороз д-р техн. наук, проф. (Академія таможенної Службы Украины)

Т. И. Пашова канд. техн. наук, доц. (Дніпропетровсьгий державний аграрний університет)

ТЕМА 1 КЛАСС TDATASET – ОСНОВНОЙ КЛАСС ДОСТУПА К ДАННЫМ

Прежде чем приступить к разработке приложений баз данных необходимо познакомиться с механизмами доступа к данным и изучить свойства и методы базовых компонентов.

1.1 Класс TDataSet

Основным классом, определяющим доступ к данным, является класс **TDataSet**, который содержит абстрактные методы непосредственного управления данными. Данный класс является предком всех компонентов доступа к данным. При разработке приложений баз данных пользователь никогда не будет создавать объект типа **TDataSet**. Вместо этого пользователь будет использовать объекты **TTable**, **TQuery** или другие потомки класса **TDataSet.** Объект **TDataSet** представляет собой набор записей, содержащий определенное количество полей и указатель на текущую запись (рис. 1).

В большинстве случаев **TDataSet** имеет прямое соответствие (один к одному) с физической таблицей, существующей на диске. Тем не менее, **TDataSet** обладает методами, позволяющими получить любое подмножество записей одной таблицы или объединения нескольких таблиц. Однако, во многих случаях можно исполнять запрос или выполнять другое действие, возвращающие **DataSet**, содержащий любое подмножество записей одной таблицы, либо объединение (**join**) между несколькими таблицами. Далее в тексте будут иногда использоваться термины **DataSet** и **TTable** как синонимы.

Схема доступа к данным

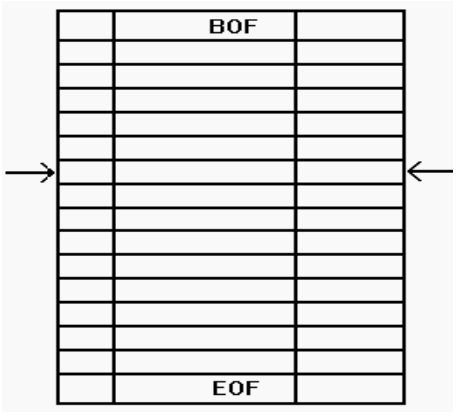


Рис.1

1.2 Открытие и закрытие DataSet

При использовании объекта **TTable** для доступа к таблице необходимо определить некоторые его свойства. Для изучения объекта необходимо поместить его во время дизайна в форму, и указать с какой таблицей будет установлена связь. Связь с таблицей устанавливается с помощью свойств **DatabaseName** и **TableName**. Для свойства **DatabaseName** достаточно указать директорий, в которой расположены таблицы в форматах **dBase** или **Paradox** например: **C:\DELPHI\DEMOS\DATA**. Также можно выбрать из списка псевдоним базы данных **DBDEMOS** или любой другой псевдоним. Псевдоним базы данных (**Alias**) определяется в утилите **Database Engine Configuration**. В поле **TableName** необходимо указать имя таблицы. Если свойство **Active** установлено в состояние **True**, то при запуске приложения таблица будет открываться автоматически.

Существует два способа открытия таблицы во время выполнения программы. Первый способ соответствует записи: **Table.Open;**

а второй способ заключается в установке свойства **Active** таблицы в состояние **True**: **Table.Active := True;**

Большого отличия между первым и вторым способом нет, так как метод **Open** заканчивается установкой свойства **Active** в состояние **True**. Поэтому второй способ несколько более эффективен.

Также имеется и два способа закрыть таблицу. Первый способ определяется вызовом метода **Close**: **Table.Close;**

Второй способ осуществляется установкой свойства **Active** в состояние **False**: **Table.Active := False;**

1.3 Навигация

Для перемещения по записям внутри таблицы объект **TDataSet** обладает следующими методами и свойствами:

- **procedure** First.
- **procedure** Last.
- **procedure** Next.
- **procedure** Prior.

- **property** BOF: Boolean **read** FBOF.
- **property** EOF: Boolean **read** FEOF.
- **procedure** MoveBy(Distance: Integer).

Метод **Table.First** перемещает указатель к первой записи в таблице. Метод **Table.Last** перемещает указатель к последней записи в таблице. Методы **Table.Next** и **Table.Prior** перемещают указатели на одну запись вперед и назад соответственно. Свойства **BOF** и **EOF** указывают на начало и конец таблицы. Метод **MoveBy** перемещает указатель на определенное число записей к началу или концу таблицы. Метод **Table.MoveBy(1)** аналогичен методу **Table.Next**, а метод **Table.MoveBy(-1)** аналогичен методу **Table.Prior.**

Рассмотрим на примере описанные методы и свойства объекта **TDataSet**. Необходимо поместить в форму следующие компоненты: **DataSource**, **Table**, **DBGrid**, **DBNavigator**, 4 компонента **Button** и компонент **SpinEdit**. Кнопкам **Button** необходимо присвоить следующие имена: **NextBtn**, **PriorBtn**, **FirstBtn**, **LastBtn**, **ReturnBtn**, **GoEndBtn** и **MoveBtn**. Посредством свойства **DataSource** требуется установить связь компонентов **DBGrid1** и **DBNavigator1** с компонентом **DataSource1**. Используя свойство **DataSet** компонента **DataSource1**, необходимо установить связь с компонентом **Table1**. Достаточно выбирать из списков **DataBaseName** псевдоним **DBDEMOS** и **TableName** имя таблицы **CUSTOMER** и установить свойство **Active** таблицы в состояние **True**. После выполнения данной операции в компоненте **DBGrid1** появятся данные таблицы **CUSTOMER** (рис.2).

Доступ к данным в период дизайна

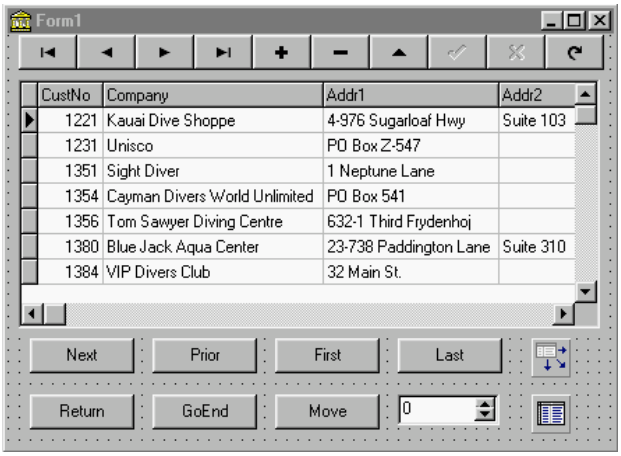


Рис.2

Теперь можно запустить приложение на выполнение и проследить перемещение по полям таблицы посредством нажатия соответствующих кнопок компонента **DBNavigator1** и полос прокруток компонента **DBGrid1**.

Компонент **DBNavigator** обладает практически всеми необходимыми свойствами и методами управления таблицей. Однако нашей задачей является рассмотрение вопроса программного управления объектом **TDataSet** без использования универсального элемента управления **DBNavigator**. Поэтому, для обработчиков событий **OnClick** кнопок необходимо написать строки, приведенные в примере 1.

Пример 1

```
procedure TForm1.NextBtnClick(Sender: TObject);
begin
  Table1.Next;
end;
procedure TForm1.PriorBtnClick(Sender: TObject);
begin
 Table1.Prior;
end;
procedure TForm1.FirstBtnClick(Sender: TObject);
begin
 Table1.First;
end;
procedure TForm1.LastBtnClick(Sender: TObject);
begin
  Table1.Last;
end;
```

Теперь можно запустить приложение и проверить функционирование кнопок.

При работе с базами данных часто возникает необходимость изменять значения полей для всей таблицы. С этой целью удобно использовать свойства **TDataSet.BOF** и **TDataSet.EOF**.

TDataSet.BOF – **read-only Boolean** свойство, используется для проверки, находитесь ли Вы в начале таблицы. Свойства **BOF** возвращает **true** в трех случаях:

- После того, когда пользователь открыл файл.
- После того, когда пользователь вызывал **TDataSet.First**.
- После того, когда вызов **TDataSet.Prior** не выполняется.

Первые два пункта – очевидны. Когда пользователь открывает таблицу, выполняется помещение на первую запись. Когда вызывается метод **First**, также происходит перемещение указателя в начало таблицы. Метод **Prior** требует небольшого пояснения. После того, когда пользователь многократно вызывает метод **Prior**, возникает ситуация достижения начала таблицы, и следующий вызов **Prior** будет неудачным. При этом **BOF** возвратит значение **True**. Используя данное свойство, можно определить достижение начала таблицы, как показано в примере 2.

Пример 2

```
procedure TForm1.ReturnBtnClick(Sender: TObject);
begin
  Table1.Last;
  while not Table1.Bof do
  begin
  { В данном месте должен находиться текст модификации
    полей  таблицы.}
  Table1.Prior;
  end;
end;
```

Аналогично свойству **BOF** применяется свойство **EOF** для проверки достижения конца таблицы. **EOF** возвращает **True** в следующих трех случаях:

- После того, когда пользователь открыл пустой файл.
- После того, когда пользователь вызывал **TDataSet.Last**.
- После того, когда вызов **TDataSet.Next** не выполняется.

Пример 3 демонстрирует простой способ перемещения по всем записям таблицы, начиная с первой записи.

Пример 3

```
procedure TForm1.GoEndBtnClick(Sender: TObject);
begin
Table1.First;
  while not Table1.EOF do
  begin
{В данном месте должен находится текст модификации полей таблицы.}
 Table1.Next;
  end;
end;
```

Для перемещения на заданное число записей в любом напрвлении используется метод **MoveBy**. Пример 4 демонстрирует указанный метод.

Пример 4

```
procedure TForm1.MoveBtnClick(Sender: TObject);
begin
   Table1.MoveBy(SpinEdit1.Value);
end;
```

1.4 Поля таблицы

Для получения доступа к полям таблицы объект **TDataSet** обладает рядом методов и свойств, основными из которых являются:

- **property** Fields[Index: Integer];
- **function** FieldByName(**const** FieldName: string): TField;
- **property** FieldCount.

Свойство **FieldCount** возвращает число полей в текущей структуре записи. Если необходимо программным путем прочитать имена полей, то для доступа к ним следует применить свойство **Fields** (пример 5).

Пример 5

```
var
S: String;
begin
S := Table.Fields[0].FieldName;
end;
```

В приведенном выше примере 5 переменной S присваивается имя первого поля таблицы, индекс которого соответствует нулю. Для доступа к имени последнего поля следует указать индекс равный **Table.FieldCount–1 (**Пример 6).

Пример 6

```
var
S: String;
begin
 S := Table.Fields[Table.FieldCount - 1].FieldName;
end;
```

Если необходимо определить текущее содержание выбранного поля конкретной записи, то рекомендуется использовать свойство **Fields** или метод **FieldsByName**. Для этого достаточно найти значение i-го поля записи как i - й элемент массива **Fields** (пример 7).

Пример 7

```
var
S: String;
i: Integer;
begin
i := 3;
S := Table.Fields[i].AsString;
end;
```

Предположим, что указанное поле в записи содержит номер записи, тогда код, приведенный выше, возвратил бы строку типа «10», «1012» или «1024». Если требуется получить доступ к этой переменный, как к числовой величине, тогда необходимо использовать тип **AsInteger** вместо типа **AsString**. Аналогично, указываются и другие типы данных: **AsBoolean**, **AsFloat** и **AsDate**.

В примере 8 показано, что для доступа к данным можно применять функцию **FieldsByName** вместо свойства **Fields**.

Пример 8

```
var
S: String;
begin
S := Table.FieldsByName('Nuncode').AsString;
  end;
```

В приведенных примерах 5 – 8 показано, что метод **FieldsByName**, и свойство **Fields** возвращают те же самые данные. Два различных синтаксиса используются для того, чтобы обеспечить программистов гибким и удобным набором инструментов для доступа к содержимому **DataSet**.

Рассмотрим пример, как можно использовать доступ к полям таблицы во время выполнения программы. Для этого необходимо поместить в форму объект **Table**, два объекта **ListBox** и две кнопки **Button**, и присвоить кнопкам имена **Fields** и **Values** (рис. 3.). Далее требуется установить связь объекта **Table1** с базой данных **DBDEMOS** посредством свойства **DatabaseName** и выбрать таблицу **CUSTOMER** в списке свойства **TableName**. Для доступа к данным в период дизайна требуется установить свойство **Active** таблицы в состояние **True**.

Расположение элементов управления в форме

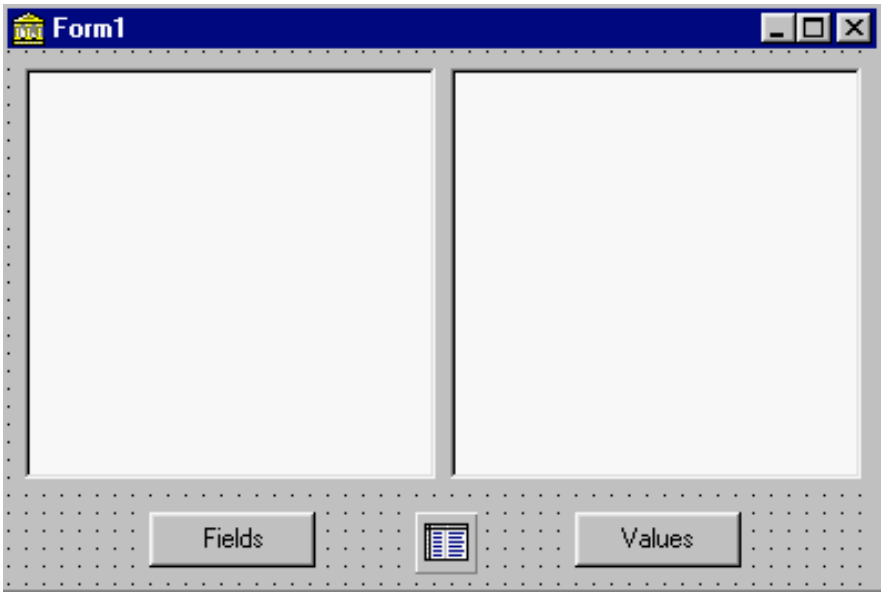


Рис.3

Далее необходимо написать обработчик события для нажатой кнопки **Fields** (пример 9).

Пример 9

```
procedure TForm1.FieldsClick(Sender: TObject);
var
i: Integer;
begin
ListBox1.Clear;
for i := 0 to Table1.FieldCount - 1 do
ListBox1.Items.Add(Table1.Fields[i].FieldName);
end;
```

Приведенная процедура (пример 9) функционирует следующим образом. Первая программная строка очищает компонент **ListBox1**. Затем в цикле последовательно добавляются имена полей таблицы. Обратите внимание на то, что счет в цикле начинается с 0, и заканчивается **Table1.FieldCount – 1**. Если не вычесть единицу из значения **FieldCount**, то произойдет исключительная ситуация «**List Index Out of Bounds**», обусловленная попыткой прочесть имя поля которое не существует.

Для получения доступа к содержимому полей обработчик события для кнопки **Values** будет иметь вид, приведенный в примере 10.

Пример 10

```
procedure TForm1.ValuesClick(Sender: TObject);
var
i: Integer;
begin
ListBox2.Clear;
for i := 0 to Table1.FieldCount - 1 do
ListBox2.Items.Add(Table1.Fields[i].AsString);
end;
```

На рисунке 4 приведено функционирующее приложение, демонстрирующее выполнение процедур приведенных в примерах 9 – 10.

Реализация процедур

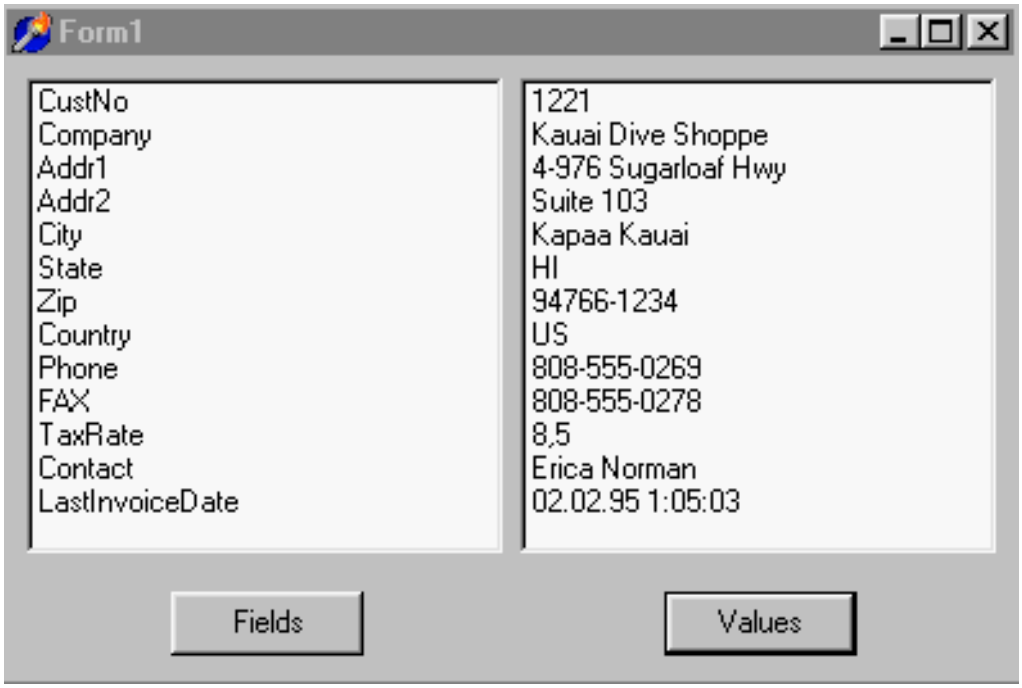


Рис.4

Напомним, что класс **TField** обладает следующими свойствами:

- **property** AsBoolean;
- **property** AsFloat;
- **property** AsInteger;
- **property** AsString;
- **property** AsDateTime.

Если возникает необходимость, то можно преобразовывать тип поля **Boolean** к **Integer** или к **Float**, тип поля **Integer** к **String** или тип поля **Float** к **Integer**. Однако преобразовывать тип поля **String** к **Integer** невозможно. Если необходимо работать с полями **Date** или **DateTime**, то можно использовать преобразования **AsString** и **AsFloat** для доступа к ним.

1.5 Работа с данными

Для работы с данными объект TTable имеет следующие методы:

- **procedure** Append;
- **procedure** Insert;
- **procedure** Cancel;
- **procedure** Delete;
- **procedure** Edit;
- **procedure** Post;

Все указанные методы являются методами **TDataSet**. Они унаследованы и используются объектами **TTable** и **TQuery**. Всякий раз, когда пользователь желает изменить данные, он должен сначала перевести **DataSet** в моду редактирования. Большинство элементов управления базами данных переводят **DataSet** в моду редактирования автоматически. Однако, если необходимо изменить **TTable** программно, пользователю придется использовать вышеуказанные функции.

Для изменения содержимого поля в текущей записи необходимо выполнить последовательность программных строк (пример 11).

Пример 11

```
Table1.Edit;
Table1.FieldByName('CustName').AsString := 'Client 1024';
Table1.Post;
```

Первая строка программы переводит **DataSet** в режим редактирования. Вторая строка присваивает значение полю '**CustName**'. Третья строка обеспечивает запись данных на диск, путем вызова метода **Post**.

Другим способом сохранения записи на диске является перемещение к следующей записи после ее редактирования (пример 12).

Пример 12

```
Table1.Edit;
Table1.FieldByName('CustNo').AsInteger := 1024;
Table1.Next;
```

На основании вышеизложенного, следует вывод, что всякий раз, когда выполняется перемещение с текущей записи на другую в режиме

редактирования выполняется запись на диск. Это означает, что вызовы методов **First**, **Next**, **Prior** и **Last** завершаются методом **Post**, при условии, что объект **DataSet** нахотится в состоянии редактирования.

Для добавления и вставки новых записей в таблицу используются методы **Append** и **Insert** соответственно. Рассмотрим данные методы на следующем примере. Для демонстрации указанных методов необходимо разместить в форме компоненты **Table**, **DataSource**, **DBGrid** и три кнопки **Button**. Присвоим кнопкам соответствующие имена: **InsertBtn**, **AppendBtn** и **DeleteBtn**. Далее необходимо установить соединения между объектами **DataSource1**, **Table1**, **DBGrid1,** и связать объект **Table1** с таблицей **COUNTRY** базы данных **DBDEMOS** (рис. 5). Для доступа к данным в период дизайна требуется установить свойство **Active** объекта **Table1** в состояние **true**.

Расположение элементов управления в форме

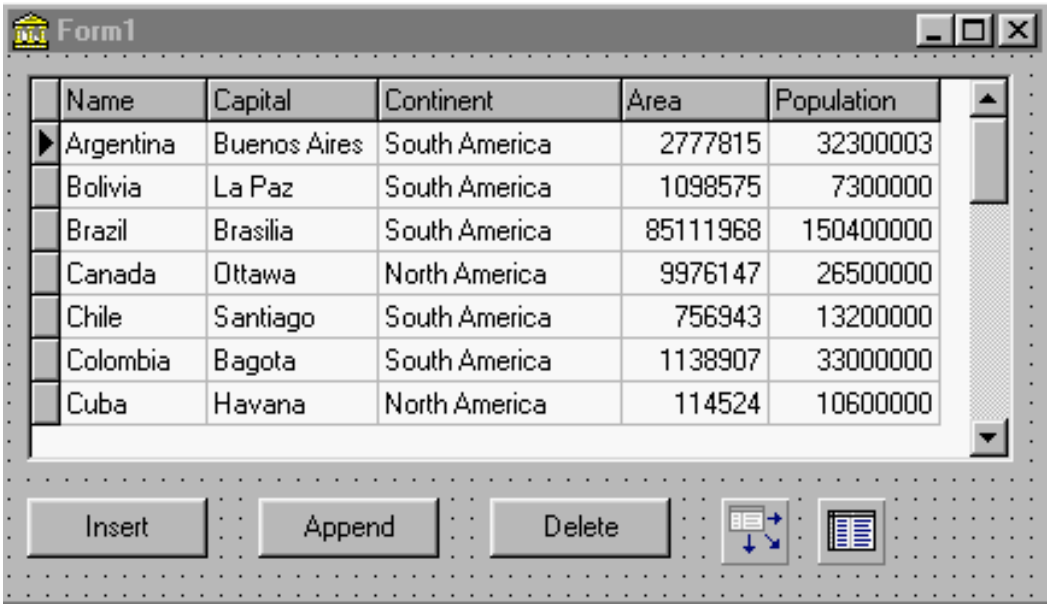


Рис.5

Процедуры обработчиков событий **OnClick** кнопок, в нашем случае, будут иметь вид приведенный в примере 13.

Пример 13

```
procedure TForm1.InsertBtnClick(Sender: TObject);
begin
```

```
  Table1.Insert;
  Table1.FieldByName('Name').AsString := 'Ukraine';
  Table1.FieldByName('Capital').AsString := 'Kiev';
  Table1.Post;
end;
procedure TForm1.AppendBtnClick(Sender: TObject);
begin
  Table1.Append;
  Table1.FieldByName('Name').AsString := 'Ukraine';
  Table1.FieldByName('Capital').AsString := 'Kiev';
  Table1.Post;
end;
procedure TForm1.DeleteBtnClick(Sender: TObject);
begin
   Table1.Delete;
end;
```

Процедура **TForm1.InsertBtnClick(Sender: TObject)** переводит таблицу в режим вставки, то есть, создается новая запись с пустыми полями, которая вставляется в текущую позицию **dataset**. После вставки пустой записи, выполняется присвоение значений одному или нескольким полям, а затем вызывается метод **Post**. Аналогично функционирует обработчик события **AppendBtnClick**.

Существует несколько различных способов присвоения значений полям таблицы. В рассматриваемой программе пользователь мог бы просто ввести информацию в новую запись, используя возможности **DBGrid**. Также пользователь мог разместить в форме стандартное элемент ввода **TEdit** и присвоить каждому полю значение, которое он ввел в элемент ввода:

```
Table1.FieldByName('Name').AsString := Edit1.Text;
```

Интересным моментом в рассматриваемом примере является то, что нажатие кнопки **Insert** дважды подряд автоматически вызывает исключительную ситуацию '**Key Violation**', так как первое поле таблицы

COUNTRY опредедено как певичный ключ. Таким образом, **DataSet** защищает базу данных от повторного присвоения одинакового имени.

Если после вызова методов **Insert** или **Append**, необходимо отказаться от вставки или добавления новой записи, то надо вызвать метод **Cancel** до метода **Post**.

1.6 Использование SetKey для поиска записей в таблице

С целью нахождения необходимых значений полей таблицы, используются две процедуры **SetKey** и **GotoKey**. Обе эти процедуры предполагают, что поле, по которому производится поиск индексировано.

Для того, чтобы создать программу поиска данных, требуется поместить в форму компоненты **Table**, **DataSource**, **DBGrid**, **Button**, **Label** и **Edit**, и расположить их как показано на (рис. 5). Далее необходимо изменить имя кнопки **Button** на имя **SearhBtn**, и затем соединить компоненты управления базой данных так, чтобы пользователь открыл в **DBGrid1** таблицу **Customer**.

Расположение компонентов в форме

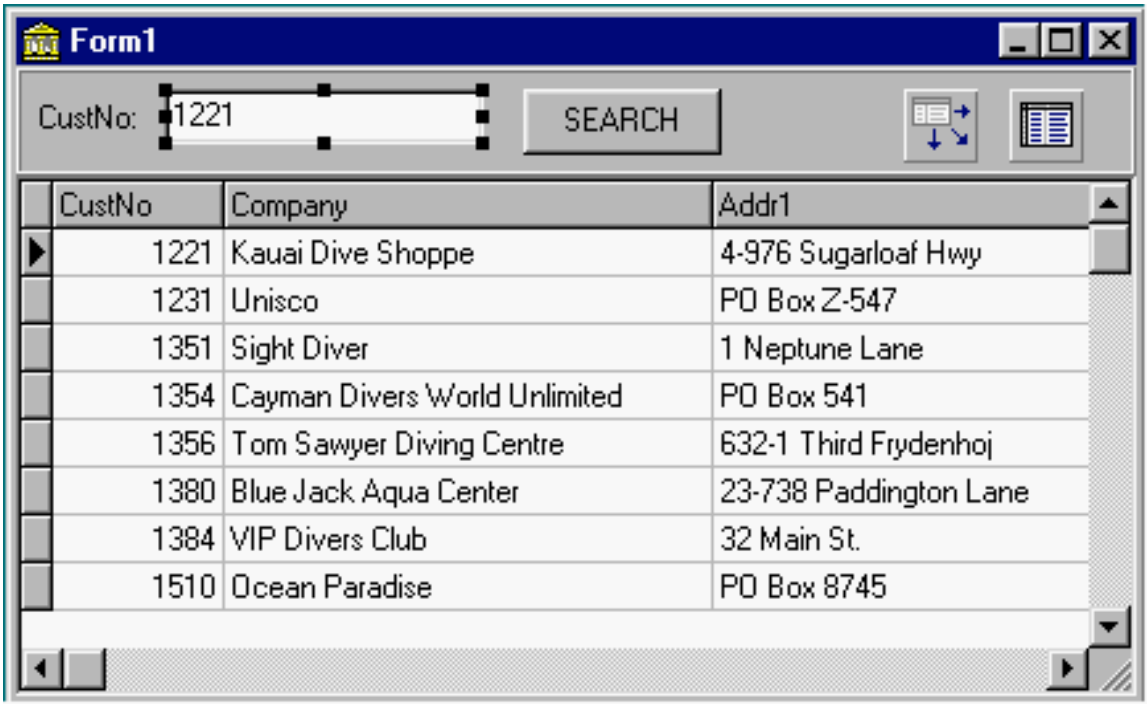


Рис.6

Вся функциональность данной программы скрыта в единственном методе, который является обработчиком события **OnClick** кнопки **SearchBtn**. Данный обработчик считывает строку, введенную в окно редактора, и ищет ее значение в поле **CustNo**, и помещает фокус на найденной записи. В простейшем варианте, код обработчика события выглядит так (пример 14):

Пример 14

```
procedure TForm1.SearhBtnClick(Sender: TObject);
begin
Table1.SetKey;
Table1.FieldByName('CustNo').AsString := Edit1.Text;
Table1.GotoKey;
end;
```

Первый вызов в этой процедуре установит **Table1** в режим поиска. Далее, свойству **Fields** присваивается значение, которое пользователь ввел в элемент ввода. Для фактического выполнения поиска нужно вызывать метод **Table1.GotoKey**.

Если необходимо выполнить поиск не по первичному индексу файла, тогда пользователю потребуется определить имя индекса, который будет использоваться свойством IndexName. Например, если таблица Customer имеет вторичный индекс по полю City, тогда пользователь должен установить свойство IndexName равным имени индекса. Реализация описанного метода приведена в примере 15.

Пример 15

```
procedure TForm1.SearhBtnClick(Sender: TObject);
begin
Table1.IndexName := 'CityIndex';
Table1.Active := True;   Table1.SetKey;
Table1.FieldByName('City').AsString := Edit1.Text;  Table1.GotoKey;
end;
```

Обращаем внимание на тот факт, что поиск не будет выполняться, если пользователь не назначит правильно индекс (свойство **IndexName**). Также, пользователь должен помнить, что **IndexName** - это свойство

объекта **Ttable**, и не присутствует в других прямых потомках **TDataSet** или **TDBDataSet**.

Когда пользователь производит поиск некоторого значения, всегда существует вероятность того, что поиск окажется неудачным. В таком случае будет автоматически возникать исключительная ситуация (**exception**). Если возникает необходимость обработать ошибку самостоятельно, то пользователь мог бы написать примерно такую процедуру (пример 16).

Пример 16

```
procedure TForm1.SearhBtnClick(Sender: TObject);
begin
Cust.SetKey;
Cust.FieldByName('CustNo').AsString:= CustNoEdit.Text;
if not Cust.GotoKey then
raise Exception.CreateFmt('Cannot find CustNo %g', [CustNo]);
end;
```

В приведенном примере 16, неверное присвоение номера, или неудача поиска автоматически приведут к сообщению об ошибке:

‘Cannot find CustNo %g’.

Иногда требуется найти не точно совпадающее значение поля, а близкое к его значению. Для этого следует вместо **GotoKey** пользоваться методом **GotoNearest**.

1.7 Использование фильтров для ограничения числа записей

Процедура **ApplyRange** позволяет установить фильтр, который ограничивает диапазон просматриваемых записей. Например, в таблице **Customers**, поле **CustNo** имеет диапазон от 1,000 до 10,000. Если пользователь желает получить доступ к идентификаторам в диапазоне 2000 – 3000, то он должен использовать метод **ApplyRange**, и еще два связанных с ним метода. Данные методы работают только с индексированным полем.

Приведенные ниже процедуры наиболее часто используются для установки фильтров:

- **procedure** SetRangeStart;
- **procedure** SetRangeEnd;
- **procedure** ApplyRange;
- **procedure** CancelRange.
-

Кроме того, объект **TTable** содержит дополнительные методы для управления фильтрами:

- **procedure** EditRangeStart;
- **procedure** EditRangeEnd;
- **procedure** SetRange.

Для создания фильтра с использованием указанных процедур необходимо:

- Вызвать метод **SetRangeStart** и посредством свойства **Fields** определит начало диапазона.
- Вызвать метод **SetRangeEnd** и посредством свойства **Fields** указать конец диапазона.
- Для активизации фильтра достаточно вызвать метод **ApplyRange**.
- Чтобы отказаться от действия фильтра необходимо вызвать метод **CancelRange**.

Рассмотрим пример использования приведенных методов для создания фильтра. Для этого необходимо поместить компоненты **Table**, **DataSource** и **DbGrid** в форму и соединить их поля таким образом, чтобы обеспечить доступ к таблице **CUSTOMERS** базы данных **DEMOS**. Затем поместить в форму два объекта **Label** и назовить их **Start Range** и **End Range**. Затем включить в форму два объекта **Edit** и две кнопки **ApplyRange** и **CancelRange** (рис.7).

Расположение компонентов в форме

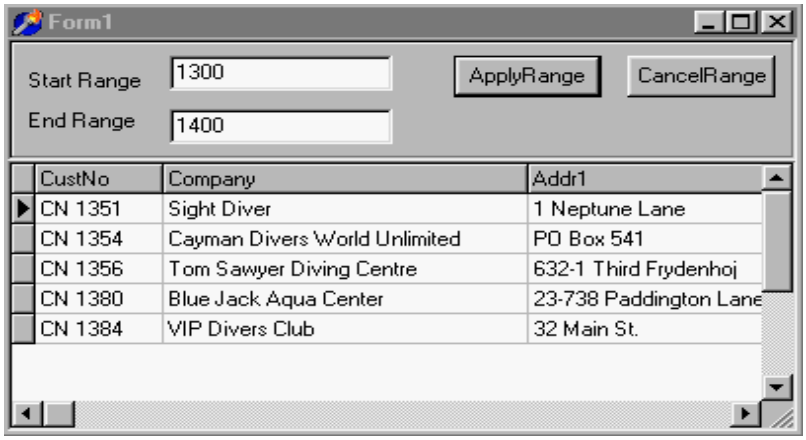


Рис.7

В примере 17 приведены процедуры, реализующие фильтр указанными методами.

Пример 17

```
procedure TForm1.ApplyRangeClick(Sender: TObject);
begin
 Table1.SetRangeStart;
 if Edit1.Text <> '' then
 Table1. Fields[0].AsString := Edit1.Text;
 Table1.SetRangeEnd;
 if Edit2.Text <> '' then
 Table1.Fields[0].AsString := Edit2.Text;
 Table1.ApplyRange;
end;
procedure TForm1.CancelRangeClick(Sender: TObject);
begin
  Table1.CancelRange;
end;
```

1.8 Обновление данных

Известно, что любая таблица базы данных, с которой работает пользователь, подвержена изменению другим пользователем. То есть, пользователь всегда должны расценить таблицу как изменяющуюся сущность. Даже если лицо является единственным пользователем, всегда существует возможность того, что приложение управления базой данных, может иметь два различных пути изменения данных в таблице. Поэтому, пользователь должен регулярно обновлять представление данных таблицы на экране.

Для обновления данных **DataSet** располагает функцией **Refresh**.Функция **Refresh** связана с функцией **Open**, таким образом, что она считывает данные, или некоторую часть данных, связанных с данной таблицей. Например, когда пользователь открывает таблицу, **DataSet** считывает данные непосредственно из файла базы данных. Аналогично, когда пользователь регенерирует таблицу, **DataSet** считывает данные

напрямую из таблицы. Вследствие чего, всегда можно использовать эту функцию, чтобы обновить таблицу. Быстрее и эффективнее, вызывать метод **Refresh**, чем метод **Close** и затем метод **Open**.

Обращаем внимание на то, что обновление **TTable** может иногда привести к неожиданным результатам. Например, если один пользователь рассматривает запись, которая уже была удалена другим пользователем, то она исчезнет с экрана в тот момент, когда будет вызван **Refresh**. Аналогично, если другой пользователь редактировал данные, то вызов **Refresh** приведет к динамическому изменению данных. Конечно, маловероятно, что один пользователь будет изменять или удалять запись в то время, как другой просматривает ее, но все же это возможно.

1.9 Закладки

Часто бывает полезно отметить текущее местоположение в таблице так, чтобы можно имелась возможность возвратиться к этому месту в дальнейшем. **DataSet** обеспечивает эту возможность посредством трех методов:

- **function** GetBookmark: TBookmark – устанавливает закладку в таблице.
- **procedure** GotoBookmark(Bookmark: TBookmark) – переводит указатель на закладку.
- **procedure** FreeBookmark(Bookmark: TBookmark) – освобождает память.

Как видно, вызов метода **GetBookmark** возвращает переменную типа **Tbookmark**. Метод **TBookmark** содержит достаточное количество информации, чтобы **DataSet** мог найти местоположение, к которому относится этот **TBookmark**. Поэтому пользователь может передавать **Tbookmark** функции **GotoBookmark**, которая немедленно возвратит указатель к местоположению, связанному с закладкой. Обращаем внимание на то, что вызов метода **GetBookmark** выделяет память для **TBookmark**, так что пользователю необходимо вызывать метод **FreeBookmark** до завершения работы программы, или перед каждой попыткой повторного использования метода **GetBookMark**.

1.10 Создание связанных курсоров

Связанные курсоры позволяют определить отношение один ко многим (**one-to-many relationship**). Например, иногда полезно связать таблицы, например CUSTOMER и ORDERS так, чтобы каждый раз, когда пользователь выбирает имя заказчика, то он видит список заказов связанных с этим заказчиком. Иначе говоря, когда пользователь выбирает запись о заказчике, то он может просматривать только заказы, сделанные этим заказчиком.

Рассмотрим пример программы, которая использует связанные курсоры. Дизайн программы сводится к установке в форму двух экземпляров объектов TTable, двух экземпляров TDataSource и двух экземпляров элементов управления TDBGrid. Далее необходимо присоединить первый набор к таблице CUSTOMER, а второй к таблице ORDERS. Программа в этой стадии имеет вид, приведенный на рис 8.

Дизайн программы

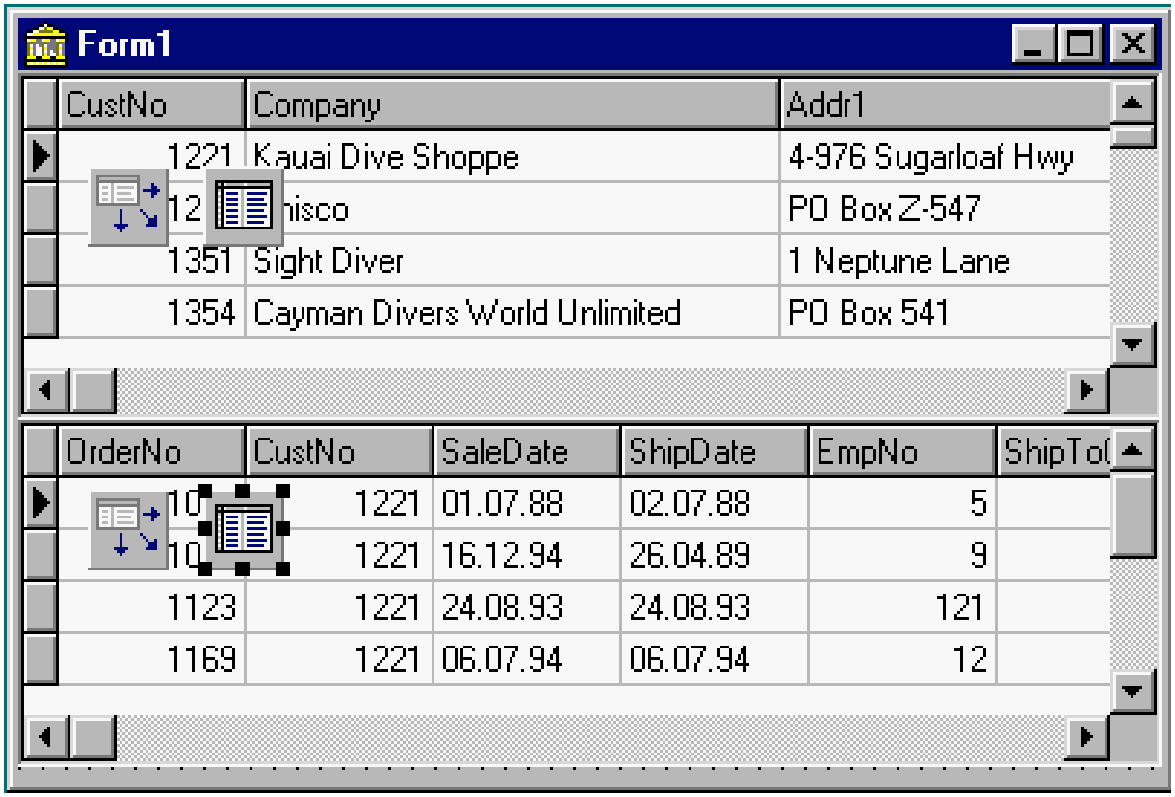


Рис.8

После выполнения дизайна следует связать таблицу ORDERS с таблицей CUSTOMER так, чтобы были видимы только те заказы, которые связанные с текущей записью в таблице заказчиков. В первой таблице заказчик однозначно идентифицируется своим номером – поле **CustNo**. Во второй таблице принадлежность заказа определяется также номером заказчика в поле **CustNo**. Следовательно, таблицы нужно связывать по полю **CustNo.** Следует заметить, что в обеих таблицах поля могут иметь различное название, но должны быть совместимы по типу. Для этого, чтобы установить связь необходимо выполнить три действия.

- Установить свойство Table2.MasterSource = DataSource1.
- Установить свойство Table2.MasterField = CustNo.
- Установить свойство Table2.IndexName = CustNo.

Второе действие выполняется посредством редактора связей (рис. 9).

Общий вид редактора связей

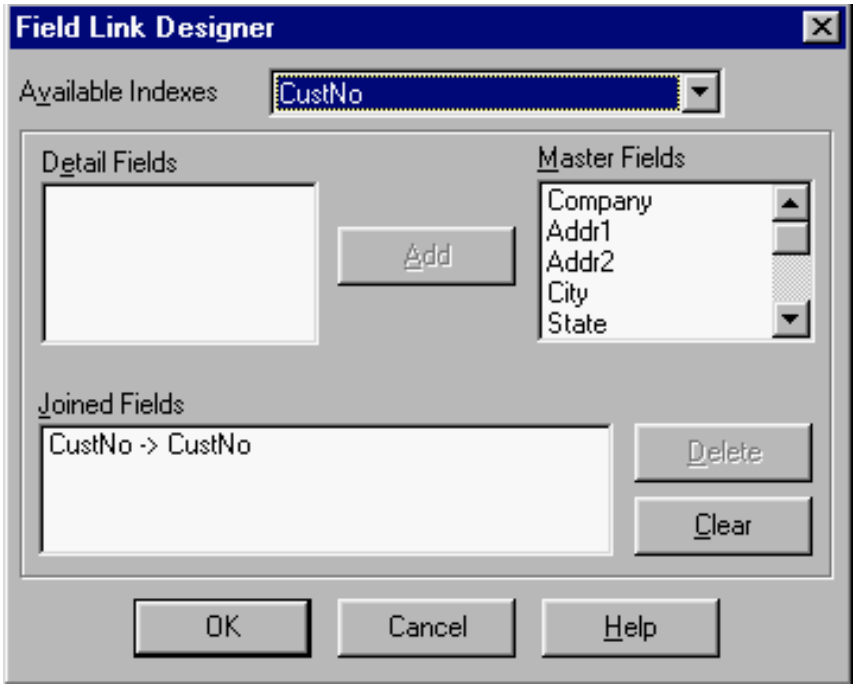


Рис.9

В период дизайна и после запуска приложения видно, что обе таблицы связаны вместе, и всякий раз, когда пользователь выполнит перемещение на новую запись в таблице CUSTOMER, в таблице ORDERS

будут отображаться только те записи, которые принадлежат соответствующему заказчику.

Связанные курсоры позволяют также определить отношение многие ко многим (**mamy-to-many relationship**). Для реализации такого отношения связанные таблицы должны содержать комплексные индексы, состоящие из нескольких полей. Причем, поле входящее в состав комплексного индекса может иметь и собственный индекс.

ТЕМА 2 ПРИМЕР КОНСТРУИРОВАНИЯ БАЗЫ ДАННЫХ В СРЕДЕ DELPHI

Сформулируем основную концепцию разработки базы данных:

- Справочник должен представлять локальную базу данных, в которой каждая таблица должна соответствовать тематическому разделу справочника.
- Каждая таблица должна иметь одинаковую структуру.
- Таблица должна содержать поисковое и описательное поле.
- Приложение управления базой данных должно обеспечивать просмотр и запись данных.
- В процессе создания базы данных приложение должно обеспечить возможность загрузки ранее подготовленных файлов примеров.
- В процессе работы со справочником должна быть обеспечена защита от случайного повреждения данных.
- С целью обеспечения возможности перемещения на другие компьютеры таблицы базы данных должны размещаться в папке приложения, т.е. без изменения конфигурации ядра базы данных.

На основании сформулированной концепции можно приступить к созданию базы данных и разработке приложения управления базой данных. Работу предлагается выполнять в следующем порядке:

1. Используя утилиту **DataBaseDesctop,** создайте в формате **Paradox** таблицы **Funct1**, **Funct2**, … **Funct(n)** соответственно количеству

разделов. Структура всех таблиц должна быть одинаковой. Каждая таблица должна иметь содержать поля, приведенные в таблице 1.

Таблица 1

	Field Name	**Type**	**Size**	**Key**
	FunctName	A	20	*
	Description	F	240	

2. Создайте новую папку будущего проекта и сохраните в ней созданные таблицы. Используя команду **Alias Manager** меню **Tools** утилиты **DataBaseDesctop**, создайте временный псевдоним будущей базы данных, например **DelphiFunction**.
3. В среде **Delphi** создайте новый проект и сохраните его с уникальным именем в папку, содержащую таблицы базы данных.
4. Выполните дизайн приложения управления базой данных согласно рисунку 10 и таблице 2.

Расположение компонентов в форме

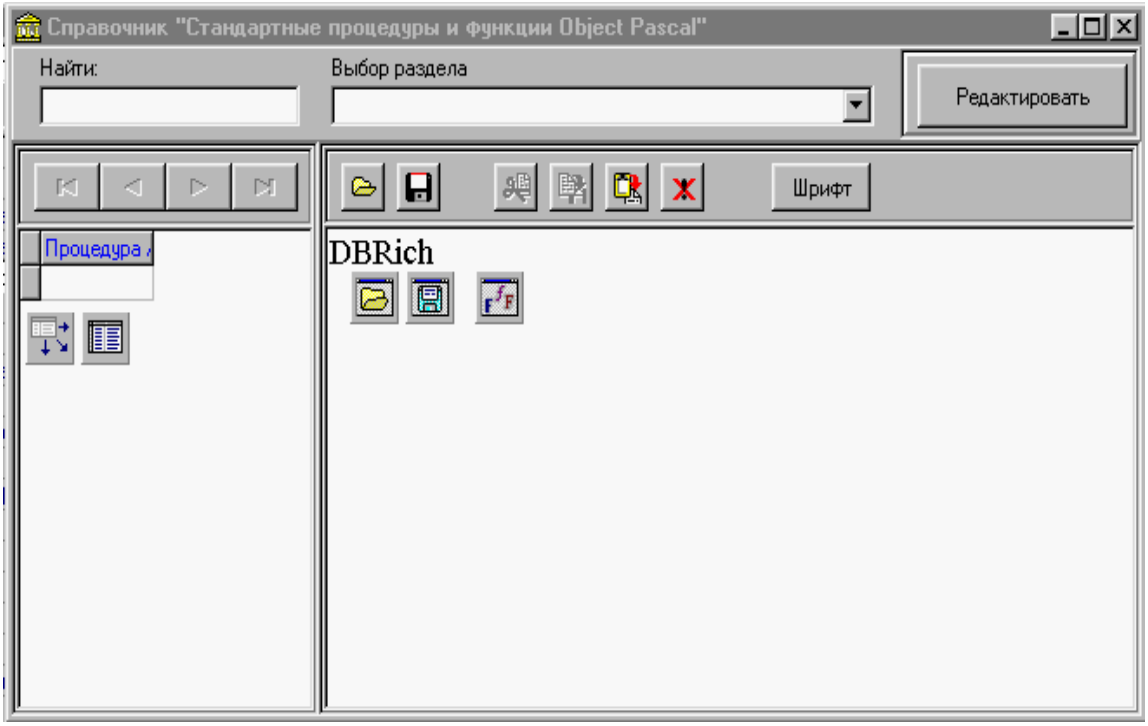


Рис.10

Таблица 2

№ п/п	Компонент	Имя	Расположение
1	TEdit	SearchEdit	Panel1
2	TComboBox	DbBox	Panel1
3	TSpeedButton	EditButton	Panel2, Panel1
4	TDBGrid	DBGrid	Panel3
5	TDBNavigator	DBNavigator	Panel4, Panel3
6	TDBRichEdit	DBRich	Panel5
7	TSpeedButton	OpenButton	Panel6, Panel5
8	TSpeedButton	SaveButton	Panel6, Panel5
9	TSpeedButton	CutButton	Panel6, Panel5
10	TSpeedButton	CopyButton	Panel6, Panel5
11	TSpeedButton	PasteButton	Panel6, Panel5
12	TSpeedButton	DelButton	Panel6, Panel5
13	TSpeedButton	FontButton	Panel6, Panel5
14	TTable	FTable	
15	TDataSource	DataSource	
16	TOpenDialog	OpenDialog	
17	TsaveDialog	SaveDialog	
18	TFontDialog	FontDialog	

5. Присвойте имена элементам управления соответственно именам, приведенным в таблице 2.

6. Выберете в списке свойств объекта **FTable** свойство **DataBaseName** и присвойте ему псевдоним Вашей базы данных. Также из списка свойства **TableName** выберете нужную таблицу, например **Funct1**.

7. Свяжите объект **DataSource** с объектом **FTable** посредством свойства **DataSet**.

8. Используя свойство **DataSource** элементов управления **DBGrid**, **DBNavigator**, **DBRich** установите связь с объектом **DataSource**. Также используя свойство **DataField** компонента **DBRich** установите его связь с полем **Description** таблицы базы данных.

9. Посредством свойства **VisibleButtons** установите видемыми соответствуюние кнопки навигатора (рис. 16, 10).
10. Откройте выбранную таблицу базы данных установив свойство **Active** объекта **FTable** в состояние **True** и настройте компонент **DBGrid** в соответствии рисунку 16.10. посредством редактора свойства **Columns**.
11. Закройте выбранную таблицу базы данных путем установки свойства **Active** в состояние **False.**
12. Занесите названия всех разделов справочника в поле списка DbBox.

Прежде чем приступить к разработке программы сформулируем задачи управления. Итак, приложение должно обеспечить два режима функционирования, а именно режим создания базы данных и режим просмотра и поиска записей. В режиме создания и редактирования базы данных, приложение должно обеспечить возможность загрузки информации (описания функций и примеров их применения) из ранее подготовленных файлов, а также возможность их последующего редактирования. В режиме просмотра и поиска записей, база данных должна быть защищена от возможной модификации в результате действий пользователя. Поиск записей по ключевому полю (FunctName) должен производится без учета регистра клавиатуры. Доступ к конкретной таблице базы данных должен выполнятся посредством выбора раздела поиска.

Итак, переключение режима функционирования приложения свяжем с кнопкой **EditButton**. Для этого настроим кнопку в режим «залипания», установив ее свойство **AllowAllUp** в **true**, и присвоив свойству **GroupIndex** значение равное единице. Переключение режимов будет состоять в изменении видимых кнопок навигатора, видимости панели инструментов текстового редактора, а также доступа к поисковому полю. Перед написанием процедуры следует установить свойство **Visible Panel6** в состояние **false** и свойство **ReadOnly DBRich** в **true**. Команда переключения режимов будет выглядеть как:

```
{***** Переключение режимов *****}
procedure TReferenceForm.EditButtonClick(Sender: TObject);
begin
 if EditButton.Down = false then
 begin
 DBNavigator.VisibleButtons  := [nbFirst,nbPrior,nbNext,nbLast];
 SearchEdit.Enabled := true;
 Panel6.Visible := false;
 DBRich.ReadOnly := true;
 end else
   begin
   DBNavigator.VisibleButtons  := [nbInsert,
   nbDelete,nbEdit,nbPost,nbCancel];
   SearchEdit.Text := ' ';
   SearchEdit.Enabled := false;
   Panel6.Visible := true;
   DBRich.ReadOnly := false;
   end;
 end;
```

Доступ к таблицам базы данных в нашей задаче должен соответствовать выбору раздела в списке разделов справочника, т.е. при выборе содержимого списка **DbBox** должна открываться соответствующая таблица. Тогда реализация команд доступа к таблицам будет иметь следующий вид:

```
{****** Выбор таблиц базы данных *****}
procedure TReferenceForm.DbBoxChange(Sender: TObject);
begin
 FTable.Close;
 FTable.DatabaseName := ExtractFilePath (Application.ExeName);
 if  DbBox.Text= 'Арифметические и математические' then
 FTable.TableName := 'Funct1';
```

```
  if  DbBox.Text= 'Операции над строками' then
  FTable.TableName := 'Funct2';
  FTable.Open;
  DBRich.DataField := 'Description';
end;
```

Таким образом, число конструкций **if** DbBox.Text= 'xxxxxx' **then** будет соответствовать количеству разделов спавочника и количеству таблиц в базе данных.

Обратите внимание на первые строки процедуры.

```
  FTable.Close;
  FTable.DatabaseName := ExtractFilePath (Application.ExeName);
```

В первой строке вытолняется закрытие таблицы с целью избежания исключительной ситуации при выборе следующей таблицы. Вторая строка процедуры указывает полный путь к базе данных, которая в нашем случае находится в папке исполняемого файла. Замена псевдонима на указанный путь к базе данных позволяет переносить таблицы на другой комьютер без изменения файла конфигурации ядра баз данных.

***Примечание**: При установке приложения на другой компьютер, не содержащий **среду Delphi**, необходимо установить на нем ядро управления базами данных.*

Для того, чтобы поиск по индексированному полю **FunctName** выполнялся без учета заглавных и прописных символов, обработчик события изменения данных поля ввода **SearchEdit** должен иметь следующий вид:

```
{***** Поиск записей *****}
procedure TReferenceForm.SearchEditChange(Sender: TObject);
begin
  FTable.SetKey;
```

```
  FTable.FieldByName('FunctName').AsString := SearchEdit.Text;
  FTable.GotoNearest;
end;
```

В заключение разработки наделите компонент **DBRich** необходимыми свойствами текстового редактора.

```
{***** Вырезание в буфер *****}
procedure TReferenceForm.CutButtonClick(Sender: TObject);
begin
  DBRich.CutToClipboard;
end;

{***** Копирование в буфер *****}
procedure TReferenceForm.CopyButtonClick(Sender: TObject);
begin
 DBRich.CopyToClipboard;
end;
{***** Вставка из буфера *****}
procedure TReferenceForm.PasteButtonClick(Sender: TObject);
begin
  DBRich.PasteFromClipboard;
end;

{***** Очистка окна редактора *****}
procedure TReferenceForm.DelButtonClick(Sender: TObject);
begin
  DbRich.SelectAll;
  DbRich.ClearSelection;
end;

{***** Выбор шрифта *****}
procedure TReferenceForm.FontButtonClick(Sender: TObject);
begin
```

```
    FontDialog.Font := DBRich.Font;
    if FontDialog.Execute then
    DBRich.SelAttributes.Assign(FontDialog.Font);
end;

{***** Загрузка файла в формате RTF *****}
procedure TReferenceForm.OpenButtonClick(Sender: TObject);
begin
  FTable.Edit;
  DbRich.Clear;
  if OpenDialog.Execute then
    DBRich.Lines.LoadFromFile(OpenDialog.FileName);
    FTable.Post;
end;

{***** Сохранение файла в формате RTF *****}
procedure TReferenceForm.SaveButtonClick(Sender: TObject);
begin
  if SaveDialog.Execute then
  DBRich.Lines.SaveToFile(SaveDialog.FileName);
end;

{***** Защита от возникновения исключительных ситуаций *****}
procedure TReferenceForm.DBRichSelectionChange(Sender: TObject);
begin
   CopyButton.Enabled := DBRich.SelLength > 0;
   CutButton.Enabled := CopyButton.Enabled;
end;
```

Запустите приложение на выполнение и устраните возможные ошибки. Создайте несколько пробных записей в таблицу и проверьте выполнение поиска нужной записи.

Вторая часть данного задания предполагает создание информационной части базы данных. Это большая коллективная работа, требующая значительного времени и усилий. Примеры по применению процедур и функций необходимо самостоятельно разработать в среде **Delphi** и оформить, используя встроенный редактор. Также можно применить текстовый процессор **Word** или редактор **WordPad** при этом, сохранять файлы примеров необходимо в формате **RTF**. После выполнения указанных работ занесение данного материала в базу банных не вызывает затруднений.

Пример созданной базы данных приведен на рисунке 11.

Поиск функции в справочнике

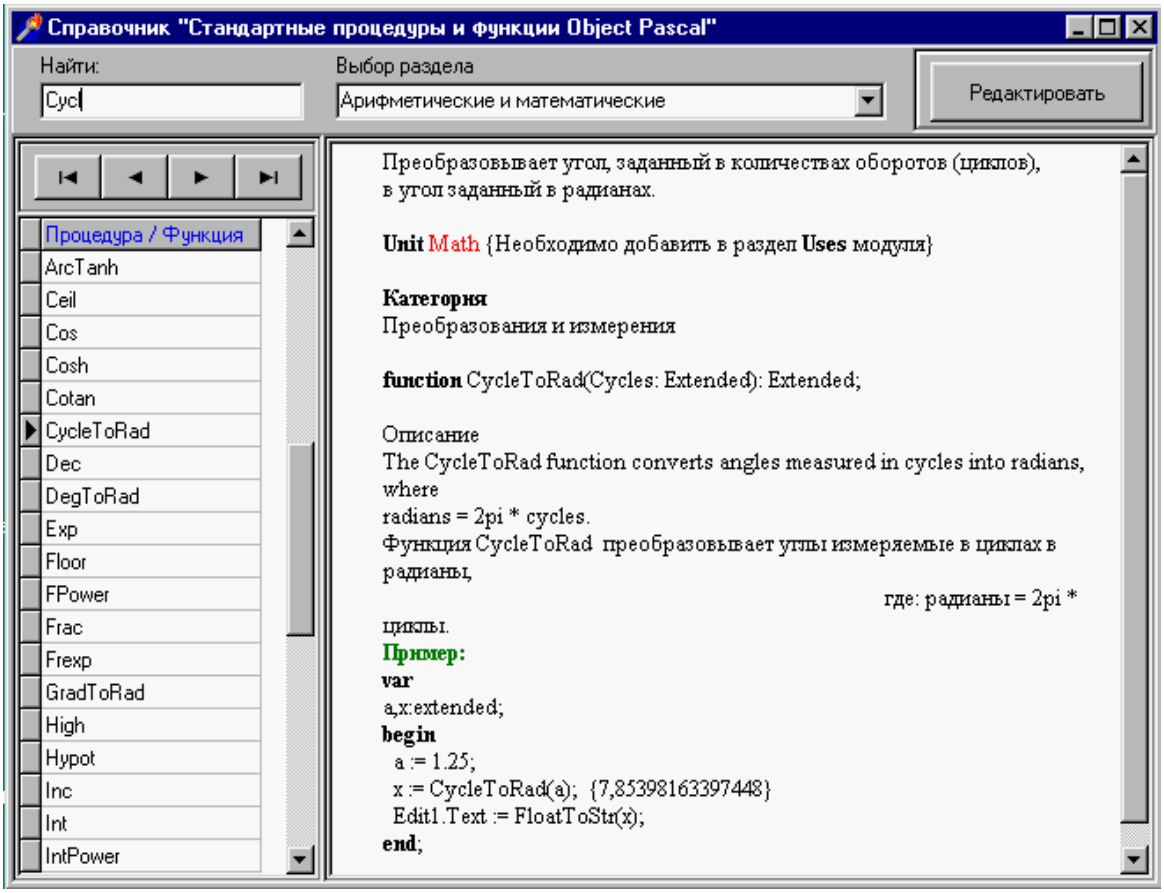


Рис.11

На основании разработанного примера базы данных можно сделать следующие выводы:

1. *На основе объектов доступа к данным TTable и TDataSource легко создавать различные приложения локальных баз данных.*
2. *Компоненты управления данными среды Delphi обладают большой гибкостью и не практически не требуют дополнительного программирования.*

Література

1. Р. Боас, М. Фервай, Х. Гюнтер, Delphi 4 Полное Руководство, – Киев.: BHV, 1998. – 448с.
2. Developer’s Guide for Delphi 3, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
3. Developer’s Guide for Delphi 5, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
4. Object Pascal Language Guide, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
5. Анталогия Delphi, http://www.Torry.ru