МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

Методические указания по проведению практических занятий и самостоятельной работы

по дисциплине

Б1.В.ДВ.2.1 Комплексы проблемно-ориентированных программ

Код и направление подготовки	09.06.01 – Информатика и вычислительная техника
Наименование профиля / программы подготовки научно-педагогических кадров в аспирантуре	Математическое моделирование, численные методы и комплексы программ
Квалификация (степень) выпускника	***Исследователь. Преподаватель исследователь***

Краснодар 2014

Практическая работа № 1. Создание баз данных с помощью Database Desktop

1. Создание новой таблицы

Прежде, чем начать строить приложения, работающие с базами данных, надо иметь сами базы данных. C++Builder поставляется с примерами, имеющими немало баз данных, которыми можно воспользоваться для обучения. Вместе с BDE и C++Builder поставляется программа Database Desktop (файл DBD.EXE для 16-разрядных приложений, файл DBD32.EXE для 32-разрядных приложений, файл DBDLOCAL.EXE — файл конфигурирования), которая позволяет создавать таблицы баз данных некоторых СУБД, задавать и изменять их структуру.

Обычно вызов Database Desktop включен в главное меню C++Builder.

Вызовите Database Desktop. Вы увидите окно, показанное на рис. 1.

Создадим с помощью Database Desktop таблицу базы данных Pers СУБД Paradox 7. В Paradox 7 база данных — это каталог, в котором лежат таблицы — файлы с расширением **.db.** Поэтому прежде надо создать соответствующий каталог с помощью любой программы Windows, например, с помощью «Проводника».

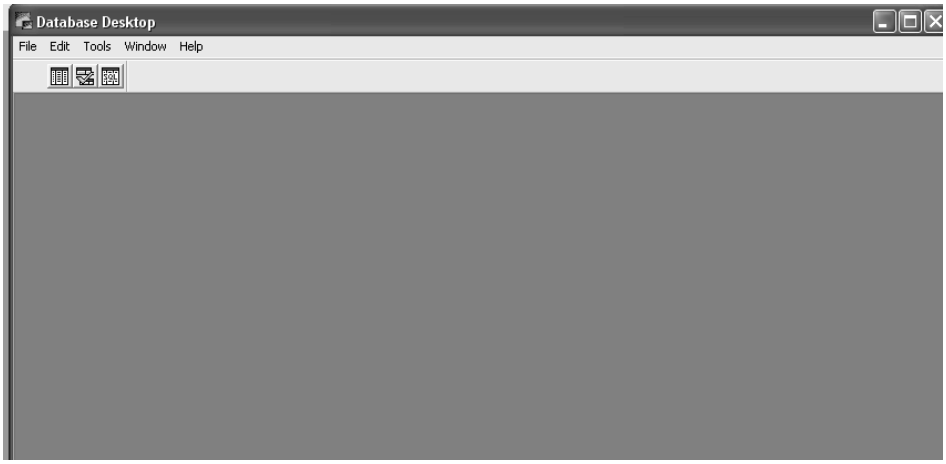


Рис. 1. Главное окно Database Desktop

Далее выполните команду Database Desktop File | New- Вам откроется подменю, содержащее три варианта:

QBE Query – Визуальный построитель запросов и запись их в файл;

SQL File – Создание запросов на SQL;

Table – Создание новой таблицы.

Выберите Table. Вам откроется небольшое диалоговое окно (рис 2). В нем из выпадающего списка вы можете выбрать СУБД, для которой хотите создать таблицу. Выберите Paradox 7.

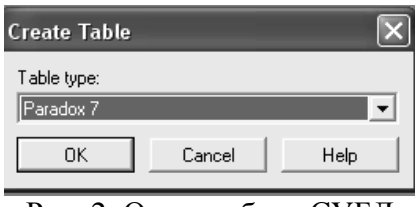


Рис. 2. Окно выбора СУБД

После этого вы увидите окно, представленное на рис. 3. В этом окне вы можете задать структуру таблицы (поля и их типы), создать вторичные индексы, ввести диапазоны допустимых значений полей, значения по умолчанию и ввести много иной полезной информации о создаваемой таблице.

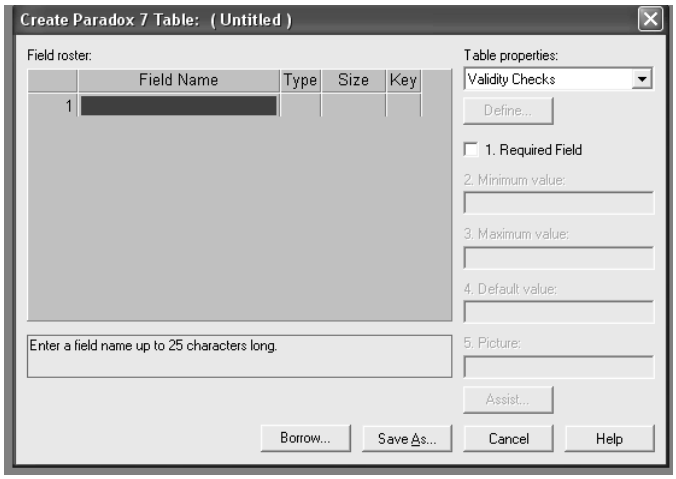


Рис. 3. Окно новой таблицы

2. Задание полей таблицы

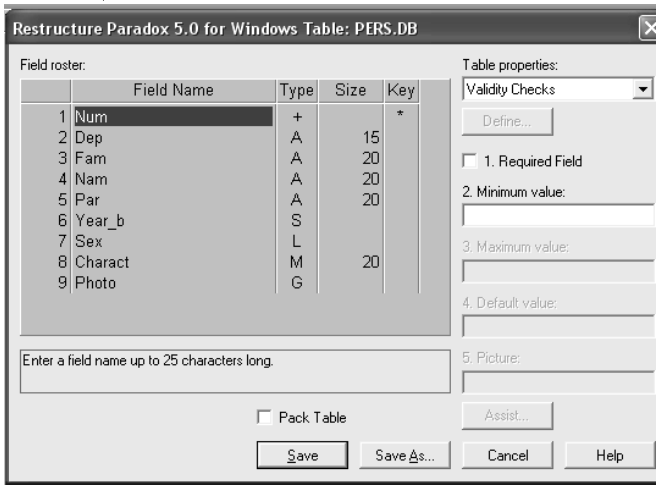


Рис. 4 .Пример создания структуры таблицы Paradox 7.

Для каждого поля создаваемой таблицы прежде всего указывается имя (Field Name) — идентификатор поля. Он может включать до 25 символов и не может начинаться с пробела (но внутри пробелы допускаются). Затем надо выбрать тип (Type) данных этого поля. Для этого перейдите в раздел Type поля и щелкните правой кнопкой мыши. Появится список доступных типов, из которого вы можете выбрать необходимый вам. Приведем пояснения типов данных, используемых в Paradox.

Таблица 1. Типы данных, используемые в Paradox и C++ Builder

Обозначение	Размер (Size)	Обозначение в списке	Пояснение
1	2	3	4
A	1-255	Alpha	Строковое поле, содержащее любые печатаемые ASCII символы. Размер – число символов.
N		Number	Действительные числа от -10^{307} до 10^{308} с 15 значащими разрядами. Для выбора формата надо использовать Paradox.
$		Money	Положительные или отрицательные числа, отличающиеся от Number формой представления и символом денежной единицы. Для выбора формата надо использовать Paradox.
S		Short	Короткие целые числа от -32 767 до 32 767
I		Long Integer	Длинные целые числа от -2 147 483 648 до 2 147 483 647
#	0-32	BCD	Числа в формате BCD (Binary Coded Decimal).

			Вычисления с этими числами проводятся с повышенной точностью, но медленнее.
D		Date	Значения представляют собой даты. Для выбора формата надо использовать Paradox.
T		Time	Значения, представляющие собой время. Для выбора формата надо использовать Paradox.
@		Timestamp	Значение, хранящее время и дату. Для выбора формата надо использовать Paradox.
M	1-240	Memo	Поля для хранения текстов неограниченной длины. Тексты хранятся в отдельных файлах **mb.** Просмотр возможен в Paradox или в приложениях C++ Builder.
F	0-240	Formatted Memo	Поле для хранения форматированных текстов неограниченной длины. Тексты хранятся в отдельных файлах **mb.** Просмотр возможен в Paradox или в приложениях C++ Builder.
G		Graphic	Изображение из файлов в форматах **.bmp, .pcx, .tif, .gif** или **.eps.** Просмотр полей возможен в Paradox или в приложениях C++ Builder.
O		OLE	Данные типа OLE – изображения, звуки, документы. Database Desktop не поддерживает поля этого типа. Просмотр полей возможен в Paradox или в приложениях C++ Builder.
L		Logical	Логические поля. По умолчанию возможные значения – **true** и **false.** При вводе данных пользователь может ввести только первый символ из возможных значений.
+		Autoincre-ment	Автоматически увеличивающееся на 1 длинное целое. Только для чтения. При удалении записей значений полей в оставшихся записях не изменяются.
B		Binary	Данные, хранящиеся в отдельных двоичных файлах Просмотр возможен в Paradox или в приложениях **mb.,** которые Database Desktop не интерпретирует. В файлах могут храниться звуки и любые другие данные.
Y	1-255	Bytes	Данные, которые Database Desktop не интерпретирует. В отличие от полей Binary хранятся в таблице, а не во внешних файлах.

В нашем примере для поля Num целесообразно выбрать тип **Autoincrement,** обеспечивающий уникальность каждой записи. Для полей Dep, Fam, Nam и Par необходимо задать тип **Alpha,** для поля Year_b — тип **Short,** для поля Sex **— Logical,** для поля Charact — **Memo,** для поля Photo — **Binary.** В таблице **Dep** для поля Dep надо задать тип **Alpha, a** для поля Proisv **— Logical.**

Для некоторых типов необходимо задавать размер (Size). Например, для строкового типа **Alpha** размер — это число символов,

Ключевые поля должны быть отмечены символом «*» в последней колонке. Для того, чтобы поставить или удалить этот символ, надо или сделать двойной щелчок в соответствующей графе информации о поле, или выделить эту графу и нажать клавишу пробела. Если имеется несколько ключевых полей, то в таблицах Paradox они должны быть первыми. В нашем примере для таблицы Pers ключевым является поле Mum, а для таблицы **Dep** — поле Dep.

Для поля Year_b задать характеристики, соответствующие рис.5.

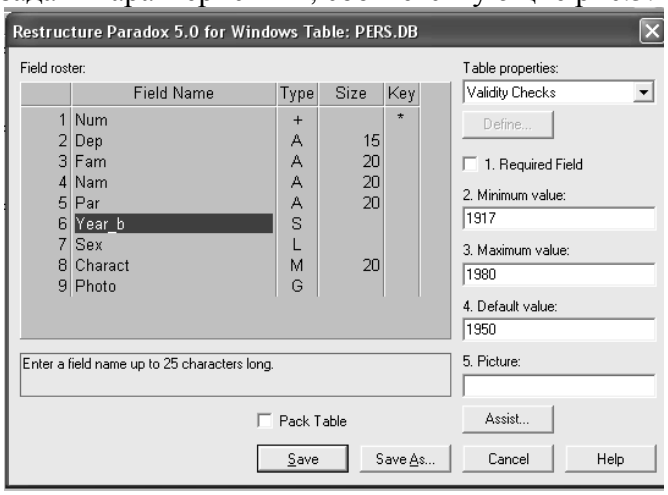


Рис. 5. Окно задания min и max значений

3. Задание свойств таблицы

Теперь обратите внимание на правую часть окна (рис.4). В нем задаются свойства таблицы (Table properties). Вверху имеется выпадающий список с рядом разделов.

Validity Checks — проверка правильности значений

Вид правой части окна при выборе этого раздела показан на рис.4 и может несколько изменяться в зависимости от того, какой тип у поля, выделенного курсором. Вы можете задать следующие характеристики поля:

Required Field	В этом индикаторе отмечаются те поля, значения которых обязательно должны содержаться в каждой записи. Для нашего примера такими полями, вероятно, должны быть поля Fam, Nam и Par.
Minimum	Минимальное значение. Это свойство полезно задавать для числовых полей. В нашем примере надо задать минимальное значение для поля Year_b.
Maximum	Максимально значение. Это свойство полезно задавать для числовых полей. В нашем примере надо задать максимальное значение для поля Year_b.
Default	Значение по умолчанию. Это свойство полезно задавать для числовых и логических полей. В нашем примере полезно задать значение по умолчанию для поля Year_b и обязательно надо задать значение по умолчанию для поля Sex (иначе у пользователя могут возникнуть проблемы при вводе информации).
Picture	Шаблон для ввода данных. Например, можно задать шаблон номера телефона и т.п. Подробнее о составлении шаблонов вы можете узнать во встроенной справке Database Desktop.
Assist	Эта кнопка вызывает диалоговое окно, помогающее создать шаблон Picture.

Table Lookup — таблица просмотра

Этот раздел позволяет связать с полем данной таблицы какое-то поле другой, просматриваемой таблицы, из которого будут браться допустимые значения. При выборе Table Lookup на экране появляется кнопка Define — определить. При ее нажатии открывается диалоговое окно, показанное на рис.5. В нем вы можете для данного поля задать таблицу просмотра (Lookup table). При этом вы можете воспользоваться выпадающим списком драйверов или псевдонимов (Drive or Alias) и кнопкой просмотра (Browse...). А затем кнопкой со стрелкой занести поле просматриваемой таблицы, из которого будут браться допустимые значения. В нашем примере пока все это сделать невозможно, поскольку еще не создана вторая таблица Dep. Однако, после того, как она будет создана, полезно вернуться к таблице Pers и для поля Dep задать таблицу Dep как просматриваемую и ее поле Dep как множество возможных значений. Это предотвратит ошибочное появление в таблице **Pers** каких-то значений подразделений, не содержащихся в таблице **Dep.**

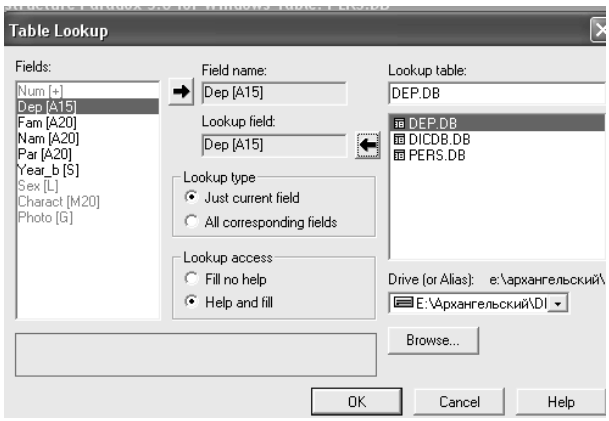


Рис. 6. Окно задания таблицы просмотра

Secondary Indexes — вторичные индексы

Этот раздел позволяет создать необходимые для дальнейшей работы вторичные индексы (первичный индекс создается по ключевым полям). Например, для дальнейшего использования нашей таблицы **Pers** полезны будут следующие индексы:

Имя индекса	**Поля**	**Пояснение**
fio	Fam, Nam, Par	**Упорядочивание таблицы сотрудников по алфавиту.**
depfio	Dep, Fam, Nam, Par	**Упорядочивание таблицы по подразделениям, а внутри каждого подразделения упорядочивание сотрудников по алфавиту.**
year	Year_b	**Упорядочивание таблицы по году рождения сотрудников.**

Чтобы создать новый вторичный индекс, нажмите кнопку Define — определить. Откроется диалоговое окно, представленное на рис. 7. В его левом окне Fields содержится список доступных полей, в правом окне Indexed fields вы можете подобрать и упорядочить список полей, включаемых в индекс. Для переноса поля из левого окна в правое надо выделить интересующее вас поле или группу полей и нажать кнопку со стрелкой вправо. Стрелками Change order (изменить последовательность) можно изменить порядок следования полей в индексе.

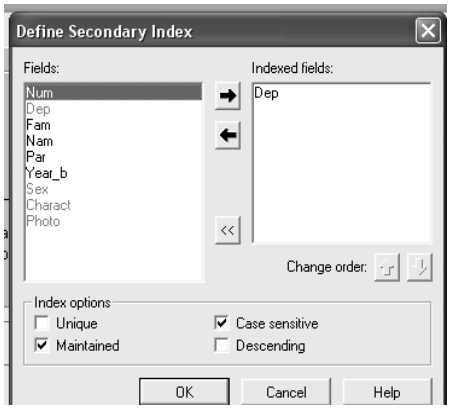


Рис. 7.Окно задания вторичного индекса

Панель радиокнопок Index Options (опции индекса) позволяют установить следующие характеристики: :

Unique — **Установка этой опции не позволяет индексировать таблицу, если в ней находятся дубликаты совокупности включенных в индекс полей. Например, установка этой опции для индекса fio не допустила бы наличия в таблице сотрудников с совпадающими фамилией, именем и отчеством.**

Descending — **При установке этой опции таблица будет упорядочиваться по степени убывания значений (по умолчанию упорядочивание производится по степени нарастания значений).**

Case Sensitive — **При установке этой опции будет приниматься во внимание регистр, в котором введены символы.**

Maintained — **Если эта опция установлена, то индекс обновляется при каждом изменении в таблице. В противном случае индекс обновляется только в момент связывания с таблицей или передачи в нее запроса. Это несколько замедляет обработку запросов. Поэтому полезно включать эту опцию для обновляемых таблиц. Если таблица используется только для чтения, эту опцию лучше не включать.**

После того, как индекс сформирован, щелкаете на кнопке OK. Открывается окно в котором вы задаете имя индекса.

Referential Integrity — целостность на уровне ссылок

Речь идет о способах, позволяющих обеспечить постоянные связи между данными отдельных таблиц. Если устанавливается целостность на уровне ссылок между двумя таблицами, одна из которых — головная (родительская), а другая — вспомогательная (дочерняя), то во вспомогательной таблице указывается поле (или группа полей), которые могут брать свои значения только из ключевого поля (или полей) головной таблицы. Подобные связи допустимы не для всех типов таблиц, но в Paradox они предусмотрены. Прежде, чем создавать Referential Integrity, надо иметь обе связываемые таблицы — родительскую и дочернюю. Если бы мы уже имели в нашем примере обе таблицы — **Pers** и Dep, мы могли бы задать целостность, связав поле Dep таблиц **Pers c** ключевым полем Dep головной таблицы **Dep.** Чтобы ввести подобную связь надо сначала установить в качестве рабочего каталог, содержащий обе таблицы (это делается командой File-> Working Directory...). Затем надо открыть дочернюю таблицу Pers (команда File | Open), войти в режим ее реструктуризации (команда Table Restructure...) и в окне Table properties выбрать раздел Referential Integrity. Затем щелкнуть на кнопке Define..., и перед вами откроется диалоговое окно, представленное на рис. 8.

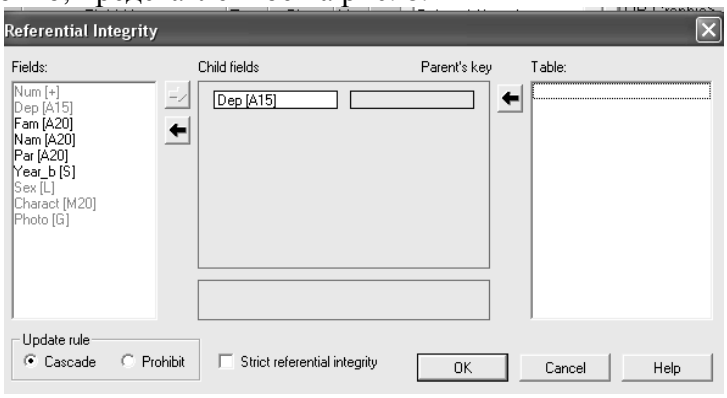


Рис.8. Окно установления .целостности на уровне ссылок

На его левой панели Fields вы можете выбрать поле или группу полей, связываемых с ключевыми полями головной таблицы, и кнопкой со стрелкой перенести их в список дочерних полей Child fields. Затем на правой панели Table вы можете указать головную таблицу (если ее там нет, значит вы неверно установили рабочий каталог) и кнопкой со стрелкой перенести в список ключей родительской таблицы Parent's key. Группа радиокнопок Update rule определяет, что будет, если в головной таблице вы удалите или измените значение ключевого поля, с которым связаны какие-то записи во вспомогательной таблице. Если вы установили опцию Prohibit, то Database Desktop просто не разрешит подобную операцию. Если же вы установили опцию Cascade, то при смене значения ключевого поля в головной таблице аналогичные изменения автоматически произойдут в записях дочерней таблицы. А если вы удалите запись в головной таблице, содержащую некоторое значение ключевого поля, то во вспомогательной таблице автоматически удалятся все записи, связанные с этим значением ключевого поля

Установка индикатора Strict Referential Integrity в нижней части диалогового окна не позволит ранним версиям Paradox (в частности, версиям Paradox для DOS) открыть и испортить таблицы, в которых введена целостность на уровне ссылок.

Когда вы провели все необходимые операции, щелкните на кнопке OK и в открывшемся окне введите имя созданной ссылки.

Password Security — пароли доступа

Paradox позволяет задать для таблицы пароли и для каждого из них определить разрешенные операции как для таблицы в целом, так и для отдельных ее полей. Щелчок на копке Define... откроет вам окно, показанное на рис. 9. В нем вы можете ввести главный пароль (окно Master password), подтвердить его (окно Verify master password), после чего щелчком на копке Auxiliary Passwords (вспомогательные пароли) открыть

новое диалоговое окно (рис. 10), позволяющее ввести вспомогательные пароли и определить правила доступа по ним.

Рис. 9. Ввод главного пароля

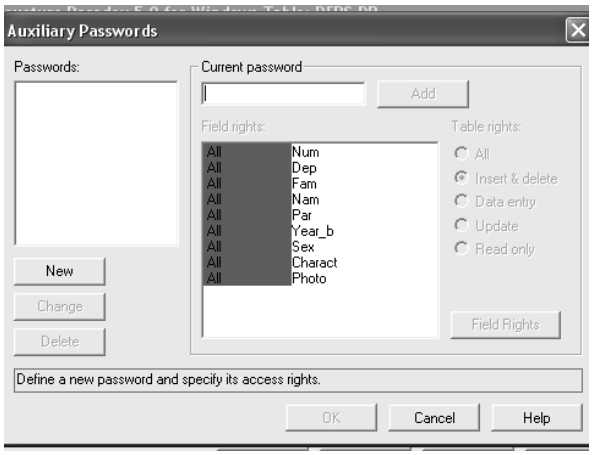


Рис. 10. Ввод вспомогательного пароля

В окне Current Password (текущий пароль) вы указываете пароль (он совершенно не обязательно должен совпадать с тем, под которым вы вошли в это окно), для которого намереваетесь сформировать правила доступа. В группе радиокнопок Table Rights (права доступа к таблице) вы можете определить общий уровень доступа к таблице:

All	**Допускаются любые операции, вплоть до изменения ее структуры, удаления таблицы, изменения и удаления паролей.**
Insert & Delete	**Допускаются любые операции с записями (редактирование, вставка, удаление), но не разрешается изменение структуры таблицы и ее удаление.**
Data Entry	**Допускается редактирование данных и вставка записей, но запрещено удаление записей и не разрешается изменение структуры таблицы и ее удаление.**
Update	**Допускается только просмотр таблицы и изменение неключевых полей.**
Read Only	**Допускается только просмотр таблицы.**

В окне Field Rights (права доступа к полю) вы можете определить дополнительные права доступа к каждому полю, но не превышающие заданный уровень доступа к таблице:

All	**Дает все права доступа к полю, предусмотренные заданными правами доступа к таблице.**
Read Only	**Позволяет только читать данные этого поля.**
None	**Не позволяет ни наблюдать, ни редактировать данное поле.**

После того, как вы установили все права доступа для данного вспомогательного пароля, щелкните на кнопке Add и пароль занесется в окно списка паролей Passwords. Далее кнопкой New вы можете начать задание нового вспомогательного пароля. Кнопкой Change вы можете изменить выделенный в списке ранее введенный вспомогательный пароль или удалить его, щелкнув после кнопки Change на кнопке Delete.

Table Language — язык таблицы

Этот раздел в выпадающем списке Table Properties позволяет задать (если он не задан) или переопределить (кнопкой Modify,-) язык таблицы, установленный по умолчанию в драйвере данной СУБД с помощью программы BDE Configuration Utility.

Dependent Tables — зависимые таблицы

Этот последний раздел в выпадающем списке Table Properties позволяет просмотреть список зависимых таблиц, связанных с данной целостностью на уровне ссылок Referential Integrity.

4. Завершение создания таблицы.

После того, как все необходимые данные о структуре таблицы внесены, щелкните на кнопке Save as... (сохранить как) и перед вами откроется окно, напоминающее обычный диалог сохранения в файле. От обычного это окно отличается выпадающим списком Alias. Этот список содержит псевдонимы различных баз данных (о них пойдет речь позднее), из которого вы можете выбрать базу данных, в которую будете сохранять свою таблицу. Если вам не надо сохранять таблицу в одной из существующих баз данных, то вы можете воспользоваться обычным списком «Сохранить» в верхней части окна. При этом вы с помощью обычной быстрой кнопки можете создать новую папку (каталог). Вспомните, что для **Paradox** база данных — это каталог, в котором сохраняется таблица.

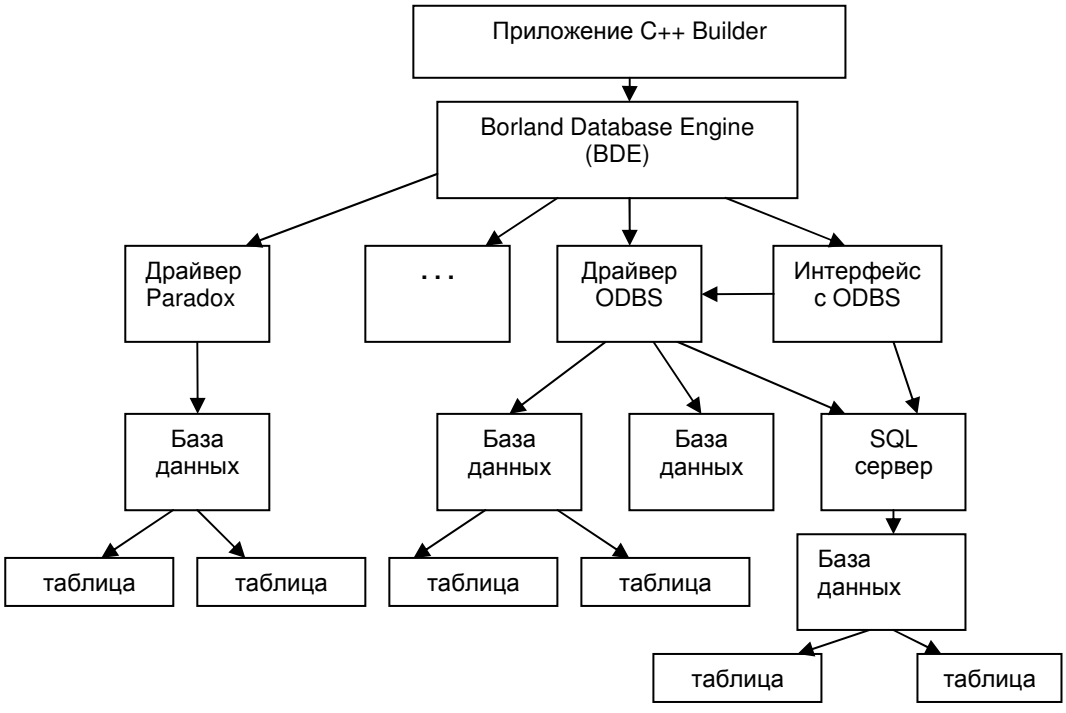
Задания для выполнения:

1. Создать таблицы Pers и Dep базы данных;
2. Задать им поля с соответствующими свойствами;
3. Задать таблицу просмотра (сооветств. примеру);
4. Задать вторичные индексы (соответств. примеру);
5. Задать целостность на уровне ссылок;
6. Задать главный и вспомогательные пароли доступа к таблице;
7. Отменить пароли в присутствии преподавателя;
8. В конце выполнения практической работы иметь готовые таблицы БД.

Практическая работа №2. Организация связи с базами данных в C++Builder

Основой работы C++Builder с базами данных является Borland Database Engine **(BDE)** — процессор баз данных фирмы Borland. BDE служит посредником между приложением и базами данных. **Он** предоставляет пользователю единый интерфейс для работы, развязывающий пользователя от конкретной реализации базы данных. Благодаря этому не надо менять приложение при смене реализации базы данных. Приложение C++Builder никогда не обращается непосредственно к базе данных, а только к BDE. Таким образом, общение с базами данных соответствует схеме, приведенной ниже.

Схема связи приложения C++Builder с базами данных



Приложение C++Builder, когда ему нужно связаться с базой данных, обращается к BDE и сообщает обычно псевдоним базы данных и необходимую таблицу в ней. BDE реализован в виде динамически присоединяемых библиотек DLL (файлы IDAPI01.DLL, IDAPI32.DLL). Они, как и любые библиотеки, снабжены API (Application Program Interface — интерфейсом прикладных программ), названным IDAPI (Integrated Database Application Program Interface). Это список процедур и функций для работы с базами данных, которым и пользуются приложения.

BDE по псевдониму находит подходящий для указанной базы данных драйвер. *Драйвер* — это вспомогательная программа, которая понимает, как общаться с базами данных определенного типа. Если в BDE имеется собственный драйвер соответствующей СУБД, то BDE связывается через него с базой данных и с нужной таблицей в ней, обрабатывает запрос пользователя и возвращает в приложение результаты обработки. BDE поддерживает естественный доступ к таким базам данных, как Microsoft Access, FoxPro, Paradox и dBase.

Если собственного драйвера нужной СУБД в BDE нет, то используется драйвер ODBC. ODBC (Open Database Connectivity) — **DLL,** аналогичная по функциям BDE, но разработанная фирмой Microsoft. Она хранится в файле ODBC,DLL. Поскольку Microsoft включила поддержку ODBC в свои офисные продукты и для ODBC созданы драйверы практически к любым СУБД, фирма Borland включила в BDE драйвер, позволяющий использовать ODBC. Правда, работа через ODBC осуществляется несколько медленнее, чем через собственные драйверы СУБД, включенные в BDE, но благодаря связи с ODBC масштабируемость C++Builder существенно увеличилась и сейчас из C++Builder можно работать с любой сколько-нибудь значительной СУБД.

BDE поддерживает SQL — стандартизованный язык запросов, позволяющий обмениваться данными с SQL-серверами, такими, как Sybase, Microsoft SQL, Oracle, Interbase. Эта возможность используется особенно широко при работе на платформе клиент/сервер.

Для того, что создать свой псевдоним БД необходимо выполнить следующее:

1. Запустить программу BDE Administrator;
2. Выполнить команду Object->New, из списка выбрать STANDART;
3. Задать имя псевдонима dbP;
4. В правой части окна BDE Administrator в свойстве PACH указать путь к папке, в которой хранятся таблицы БД;
5. Сохранить созданный псевдоним.

Теоретический материал.

Компоненты, используемые при создании баз данных и связи с ними.

Используя Borland C++ Builder, можно создавать приложения, работающие как с однопользовательскими базами данных (БД), так и с серверными СУБД (система управления базами данных). Возможности Borland C++ Builder, связанные с созданием приложений, использующих базы данных, весьма обширны. Набор данных в C++ Builder - это объект, состоящий из набора записей, каждая из которых, в свою очередь, состоит из полей, и указателя текущей записи. Набор данных может иметь полное соответствие с реально существующей таблицей или быть результатом запроса, он может быть частью таблицы или объединять между собой несколько таблиц.

Набор данных в C++ Builder является потомком абстрактного класса TDataSet (абстрактный класс - это класс, от которого можно порождать другие классы, но нельзя создать экземпляр объекта данного класса). Классы TQuery (запрос к БД) и TTable (таблица БД), содержащиеся на странице палитры компонентов Data Access, содержат абстракции, необходимые для непосредственного управления таблицами или запросами, обеспечивая средства для того, чтобы открыть таблицу, выполнить запрос или перемещаться по строкам.

Компоненты, используемые для работы с базами данных, расположены в

библиотеке компонентов на страницах Data Access (доступ к данным) и Data Control (управление данными).

Каждое приложение, использующее базы данных, обычно имеет, по крайней мере, по одному компоненту следующих трех типов:

— Компоненты — наборы данных (data set), непосредственно связывающиеся с базой данных. Это такие компоненты, как **Table, Query, StoredProc.**

— Компонент— источник данных (data source), осуществляющий обмен информацией между компонентами первого типа и компонентами визуализации и управления данными. Таким компонентом является **DataSource.**

— Компоненты визуализации и управления данными, такие, как **DBGrid, DBText, DBEdit** и множество других.

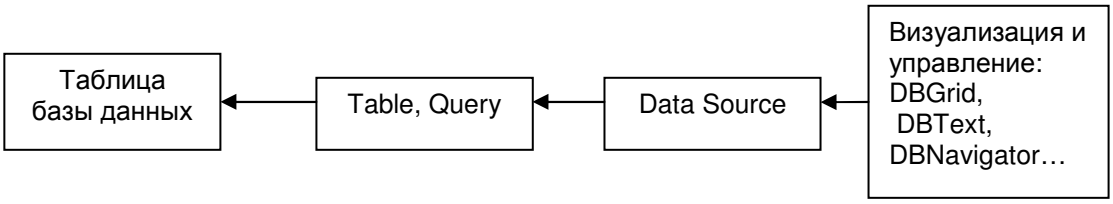


Рис. 2.9. Схема связи компонентов

Свойства компонентов. Свойства являются атрибутами компонента, определяющими его внешний вид и поведение. Многие свойства компонента в колонке свойств имеют значение, устанавливаемое по умолчанию (например, высота кнопок). Свойства компонента отображаются на странице свойств (Properties). Инспектор объектов отображает опубликованные (published) свойства компонентов. Помимо published-свойств, компоненты могут и чаще всего имеют общие (public), опубликованные свойства, которые доступны только во время выполнения приложения. Инспектор объектов используется для установки свойств во время проектирования. Список свойств располагается на странице свойств инспектора объектов. Можно определить свойства во время проектирования или написать код для видоизменения свойств компонента во время выполнения приложения.

При определении свойств компонента во время проектирования нужно выбрать компонент на форме, открыть страницу свойств в инспекторе объектов, выбрать определяемое свойство и изменить его с помощью редактора свойств (это может быть простое поле для ввода текста или числа, выпадающий список, раскрывающийся список, диалоговая панель и т.д.).

События. Страница событий (Events) инспектора объектов показывает список событий, распознаваемых компонентом (программирование для операционных систем с графическим пользовательским интерфейсом, предполагает описание реакции приложения на те или иные события, а сама операционная система занимается постоянным опросом компьютера с целью выявления наступления какого-либо события). Каждый компонент имеет свой собственный набор обработчиков событий. В C++ Builder следует писать функции, называемые обработчиками событий, и связывать события с этими функциями. Создавая, обработчик того или иного события, вы поручаете программе выполнить написанную функцию, если это событие произойдет.

Для того чтобы добавить обработчик событий, нужно выбрать на форме с помощью мыши компонент, которому необходим обработчик событий, затем открыть страницу событий инспектора объектов и дважды щелкнуть левой клавишей мыши на колонке значений рядом с событием, чтобы заставить C++ Builder сгенерировать прототип обработчика событий и показать его в редакторе кода.

Методы. Метод является функцией, которая связана с компонентом, и которая объявляется как часть объекта. Отметим, что при создании формы связанные с ней

модуль и заголовочный файл с расширением *.h генерируются обязательно, тогда, как при создании нового модуля он не обязан быть связан с формой (например, если в нем содержатся процедуры расчетов). Имена формы и модуля можно изменить, причем желательно сделать это сразу после создания, пока на них не появилось много ссылок в других формах и модулях.

Характеристики основных видов компонентов, используемых при создании базы данных:

1. Компонент TDataSource

Компонент DataSource действует как посредник между компонентами TTable и TQuery и компонентами Data Controls - элементами управления, обеспечивающими представление данных на форме. Компоненты TDataSet управляют связями с библиотекой Borland Database Engine (BDE), а компонент DataSource управляет связями с данными в компонентах Data Controls.

В типичных приложениях БД компонент DataSource, как правило, связан с одним компонентом TTable или TQuery, и с одним или более компонентами такими, как DBGrid, DBEdit. Связь этого компонента с компонентами TDataSet и DataControls осуществляется с использованием следующих свойств и событий:

— Свойство DataSet компонента DataSource идентифицирует имя компонента TDataSet. Можно присвоить значение свойству DataSet на этапе выполнения или с помощью инспектора объектов на этапе проектирования.

— Свойство Enabled компонента DataSource активизирует или останавливает взаимосвязь между компонентами TDataSource и Data Controls. Если значение свойства Enabled равно true, то компоненты Data Controls, связанные с TDataSource, воспринимают изменения набора данных. Использование свойства Enabled позволяет временно разъединять визуальные компоненты Data Controls и TDataSource, например, для того, чтобы в случае поиска в таблице с большим количеством записей не отображать на экране пролистывание всей таблицы.

— Свойство AutoEdit компонента DataSource контролирует, как инициируется редактирование в компонентах Data Controls. Если значение свойства AutoEdit равно true, то режим редактирования начинается непосредственно при получении фокуса компонентом Data Controls, связанным с данным компонентом TDataSet. В противном случае режим редактирования начинается, когда вызывается метод Edit компонента TDataSet.

— Событие OnUpdateData компонента DataSource наступает, когда пользователь пытается изменить текущую запись в TDataSet. Обработчик этого события следует создавать, когда требуется соблюсти условия ссылочной целостности или ограничения, накладываемые на значения полей изменяемой базы данных.

2. Компонент TTable

Наиболее простым способом обращения к таблицам баз данных является использование компонента TTable, предоставляющего доступ к одной таблице. Для этой цели наиболее часто используются следующие свойства:

— Active - указывает, открыта (true) или нет (false) данная таблица.

— TableName - имя таблицы.

— Exclusive - если это свойство принимает значение true, то никакой другой пользователь не может открыть таблицу, если она открыта данным приложением. Если это свойство равно false (значение по умолчанию), то другие пользователи могут открывать эту таблицу.

— IndexName - идентифицирует вторичный индекс для таблицы. Это свойство нельзя изменить, пока таблица открыта.

— MasterFields - определяет имя поля для создания связи с другой таблицей.

— MasterSource - имя компонента TDataSource, с помощью которого TTable будет получать данные из связанной таблицы.

— ReadOnly - если это свойство равно true, таблица открыта в режиме "только для чтения". Нельзя изменить свойство ReadOnly, пока таблица открыта.

— Eof, Bof - эти свойства принимают значение true, когда указатель текущей записи расположен на последней или соответственно первой записи таблицы.

— Fields - массив объектов TField. Используя это свойство, можно обращаться к полям по номеру, что удобно, когда заранее неизвестна структура таблицы.

Наиболее часто при работе с компонентом TTable используются следующие методы (некоторые из методов входят в методы навигатора (DBNavigator)):

— Open и Close устанавливают значения свойства Active равными true и false соответственно.

— Refresh позволяет заново считать набор данных из БД.

— First, Last, Next, Prior перемещают указатель текущей записи на первую, последнюю, следующую и предыдущую записи соответственно, например:

— MoveBy перемещает указатель на указанное число строк (оно может быть и отрицательным) в пределах таблицы

— Insert, Edit, Delete, Append - переводят таблицу в режимы вставки записи, редактирования, удаления, добавления записи соответственно.

— Post - осуществляет физическое сохранение измененных данных.

— Cancel - отменяет внесенные изменения, не сохраненные физически.

— FieldByName - предоставляет возможность обращения к данным в полях по имени поля:

— SetKey переключает таблицу в режим поиска.

— GotoKey начинает поиск строки, значение Fields[n] которой равно выбранному, где n - номер колонки таблицы, начиная с 0.

3. Компонент TDBGrid

Компонент TDBGrid обеспечивает табличный способ отображения на экране строк данных из компонентов TTable или TQuery. Приложение может использовать TDBGrid для отображения, вставки, уничтожения, редактирования данных БД. Обычно DBGrid используется в сочетании с DBNavigator, хотя можно использовать и другие интерфейсные элементы, включив в их обработчики событий методы First, Last, Next, Ptior, Insert, Delete, Edit, Append, Post, Cancel компонента TTable. Внешний вид таблицы (например, надписи в заголовках столбцов) может быть изменен с помощью редактора свойств Columns Editor.

Поля Float, Integer и Date обладают свойством DisplayMask. Это свойство можно использовать, чтобы форматировать данные в компоненте DBGrid или другом компоненте Data Controls. Например, экранный формат mm-dd-yy может использоваться для размещения полей типа дата. Компоненты TStringField обладают свойством EditMask, которое можно установить, вводя данные в DBGrid с уже заранее заданным условием ввода.

Все эти компоненты мы использовали в процессе проектирования базы данных, их практическое применение рассмотрено в следующих параграфах.

Практическая работа №3. Основные свойства компонента Table и простейшие приложения на его основе

1. Установка связей между компонентами и базой данных, навигация по таблице

Давайте построим простейшее приложение, работающее с базой данных. Будем использовать ту таблицу Paradox **Pers,** которую вы создавали ранее в базе данных с псевдонимом **dbP.**

В нашем простейшем приложении мы будем в качестве набора данных использовать компонент **Table.** Откройте новое приложение и перенесите на форму компонент **Table** со страницы библиотеки Data Access. Перенесите также на форму с той же страницы библиотеки компонент **DataSource,** который будет являться источником

данных. Оба эти компоненты невизуальные, пользователю они будут не видны, так что их можно разместить в любом месте формы. В качестве компонента визуализации данных возьмите компонент **DBGrid** со страницы Data Control. Это визуальный компонент, в котором будут отображаться данные таблицы. Поэтому растяните его пошире, или можете в его свойстве Align установить **alClient.**. (Рис.1.)

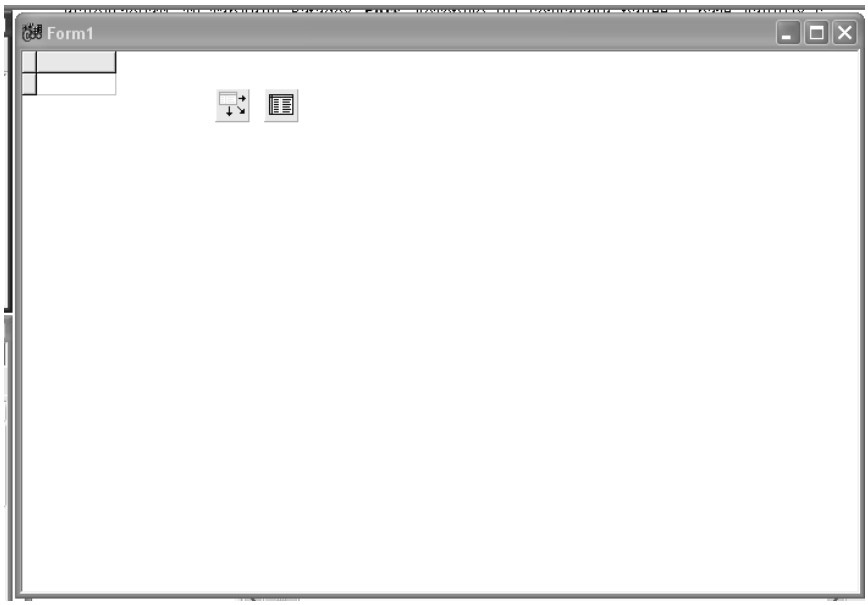


Рис.1. Форма простого приложения, работающего с базой данных

Главное свойство **DBGrid** и других компонентов визуализации и управления данными — **DataSource.** Выделите на форме компонент **DBGridI** и щелкните на его свойстве **DataSource** в Инспекторе Объектов. Вы увидите выпадающий список, в котором перечислены все имеющиеся на форме источники данных. В нашем случае имеется только один источник данных — **DataSourcel.** Установите его в качестве значения свойства **DataSource.** Далее надо установить связь между источником данных и набором данных. Выделите компонент **DataSourcel** и найдите в Инспекторе Объектов его главное свойство — **DataSet.** Щелкните на этом свойстве и из выпадающего списка выберите **Tablel** (если бы у вас было несколько компонентов — наборов данных, то все они были бы в этом списке).

Теперь осталось связать компонент **Tablel** с необходимой таблицей базы данных. Для этого служат два свойства компонента **Table: DatabaseName и TableName.** Прежде всего надо установить свойство **DatabaseName.** В выпадающем списке этого свойства в Инспекторе Объектов вы можете видеть все доступные BDE псевдонимы баз данных. Выберите из этого списка псевдоним **dbP,** который вы сами ввели ранее. Если этого псевдонима там нет, значит вы забыли его создать.

После этого можно устанавливать значение свойства TableName. В выпадающем списке этого свойства перечислены таблицы, доступные в данной базе данных. Выберите таблицу **Pers.**

А теперь наступает самый ответственный момент. Вы можете прямо в процессе проектирования соединиться с базой данных. Соединение осуществляется свойством **Active.** По умолчанию оно равно **false.**

Установите его в true. Если все сделано вами правильно, то вы увидите в поле компонента **DBGridI** данные из таблицы (рис. 2) и сможете просмотреть **их.**

Так как ваша таблица не содержит данные, заполните ее.

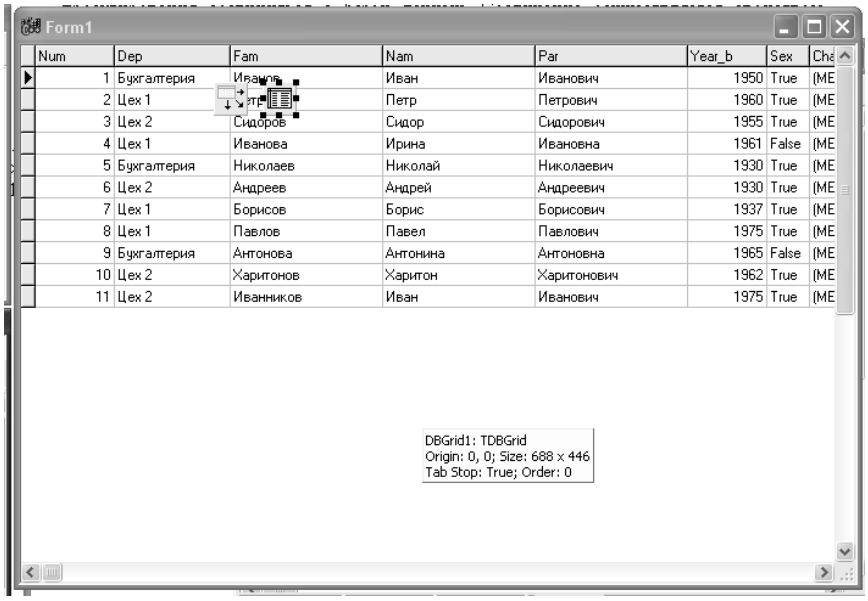


Рис. 2. Форма простого приложения после соединения компонента Table с базой данных

Следует сразу отметить, что заранее выставлять для таблиц **Active = true** допустимо только в процессе настройки и отладки приложения, работающего с локальными базами данных.

В законченном приложении во всех таблицах сначала должно быть установлено Active = false, затем при событии формы OnCreate эти свойства могут быть установлены в true, э при событии формы OnDestroy эти свойства опять должны быть установлены в false. Это исключит неоправданное поддержание связи с базой данных, которое занимает ресурсы, а при работе в сети мешает доступу к базе данных других пользователей.

Вы можете прямо сейчас, хотя приложение еще не закончено, сохранить проект, запустить его на выполнение и убедиться, что с ним можно работать. Вы можете просматривать данные, редактировать их (редактированные данные будут помещаться в базу данных в момент перехода от редактируемой записи к любой другой). Вы можете также убедиться, что в таблице **Pers** базы данных dbP невозможно изменить поле Num, поскольку оно автоматически изменяется и доступно только для чтения (см. раздел 9.2.2). Вы увидите также, что нельзя задать произвольное имя подразделения Dep, поскольку имеется таблица просмотра **Dep,** задана целостность на уровне ссылок и допустимы только те значения Dep, которые имеются в головной таблице **Dep.**

Отметим еще одно свойство компонента **Table — Exclusive.** Это свойство определяет доступ к используемой таблице при одновременном обращении к ней нескольких приложений (например, при работе в сети или в многозадачном режиме). Если задать значение этого свойства **true,** то таблица будет закрыта для других приложений. Свойство можно изменять только при **Active = false.**

Практическая работа № 4. Компонент DBNavigator, свойства и применение. Свойства полей таблицы.

В спроектированное вами простейшее приложение можно добавить еще один компонент, управляющий работой с таблицей — навигатор **DBNavigator,** расположенный на странице Data Control библиотеки компонентов. Измените свойство **Align** компонента **DBGridI** на **alBottom,** сдвиньте верхний край этого компонента немного вниз и на верх формы поместите компонент **DBNavigator**.

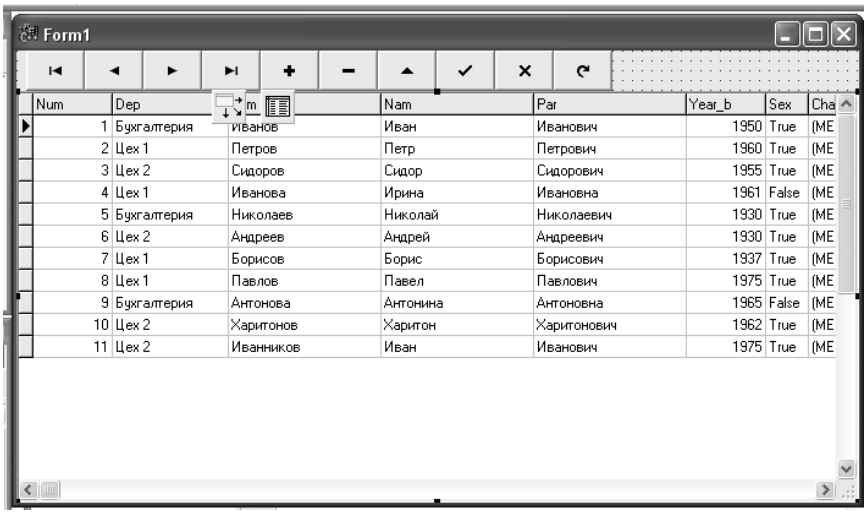


Рис.1. Форма простого приложения с навигатором

Компонент имеет ряд кнопок, служащих для управления данными. Перечислим их названия и назначение, начиная с левой кнопки:

nbFirst	**перемещение к первой записи**
nbPrior	**перемещение к предыдущей записи**
nbNext	**перемещение к следующей записи**
nbLast	**перемещение к последней записи**
nbInsert	**вставить новую запись перед текущей**
nbDelete	**удалить текущую запись**
nbEdit	**редактировать текущую запись**
nbPost	**послать отредактированную информацию в базу данных**
nbCancel	**отменить результаты редактирования или добавления новой записи**
nbRefresh	**очистить буфер, связанный с набором данных**

Пользуясь свойством навигатора **VisibleButtons,** можно убрать любые ненужные в данном приложении кнопки. Например, если вы не хотите разрешить пользователю вводить в базу данных новые записи, то можете установить в **false** кнопку **nblnsert.** Если вы хотите вообще запретить редактирование, то можно оставить только кнопки **nbFirst, nbPrior, nbNext и nbLast,** а все остальные убрать.

Чтобы приложение с навигатором работало, надо установить основное свойство навигатора — **Data**Source — источник данных (имя компонента DataSource).

Откомпилируйте приложение, выполните его и посмотрите в работе.

Задание:

1. **Создайте собственный навигатор, используя компонент SpeedButton, при щелчке на котором будут происходить изменения в таблице.**

Например:

```
void __fastcall TForm3::SpeedButton1Click(TObject *Sender)
{
Table1->First();
}
```

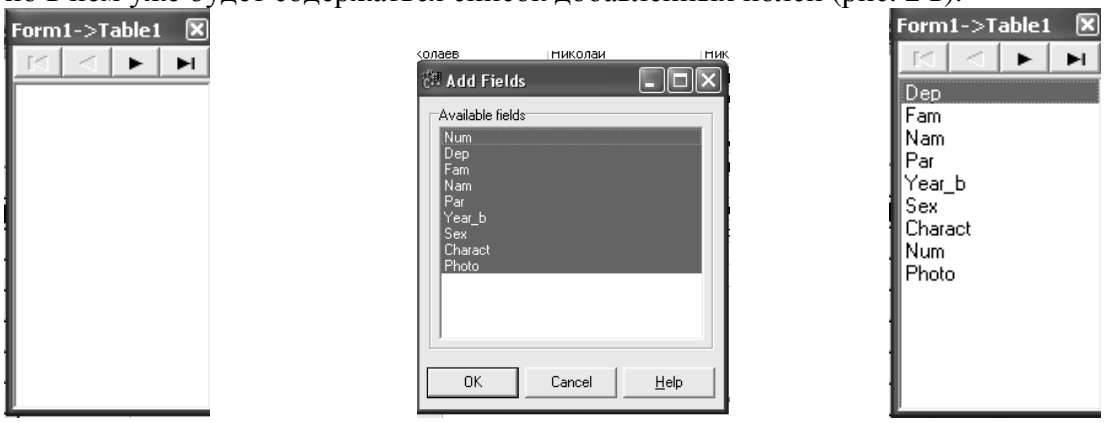
2. **Поместите на компонент SpeedButton, соответствующие иконки, используя свойство Glyph.**

Наше приложение выглядит, конечно, очень плохо. Во-первых, последовательность записей определяется ключевым полем Num, а хотелось бы, чтобы записи были расположены по алфавиту или по отделам и алфавиту. Первое поле с номерами записей вообще пользователю не нужно и надо бы, чтобы его не было видно. Шапка таблицы

содержит непонятные пользователю имена полей Num, Fam и т.д., а надо, чтобы были записаны нормальные заголовки по-русски. В графе Sex значения **true и false,** а нужны нормальные обозначения типа «м», «ж» или «мужской», «женский».

Все это можно легко поправить. Начнем с упорядочивания записей. Выделите на форме компонент **Tablel.** В Инспекторе Объектов вы увидите среди прочих свойства **IndexName и IndexFieldName.** Первое из них содержит выпадающий список индексов, созданных для вашей таблицы. Выберите, например, индекс **fio,** и увидите, что записи окажутся упорядоченными по алфавиту, поскольку в этот индекс. включены поля Fam, Nam и Par. При индексе **depfio** упорядочивание будет по подразделениям, а внутри каждого подразделения — по алфавиту. Альтернативный вариант индексации предоставляет свойство **IndexFieldName.** В нем просто перечислены предусмотренные комбинации полей и вы можете выбрать необходимую, если забыли, что обозначают имена индексов.

Теперь займемся отдельными полями. Для их редактирования служит Редактор Полей. Вызвать его проще всего двойным щелчком на компоненте **Tablel.** Сначала вы увидите пустое поле этого редактора (рис.2 а). Щелкните на нем правой кнопкой мыши и из всплывающего меню выберите раздел Add fields... (добавить поля). Вы увидите окно, изображенное на рис. 2б, в котором содержится список всех полей таблицы. Выберите из него курсором мыши интересующие вас поля. Если вы при этом будете держать нажатой клавишу Ctrl, то может выделить любую комбинацию полей. Однако, имейте в виду, что только к тем полям, которые вы добавите, вы сможете в дальнейшем обращаться. Так что в данном случае вам имеет смысл выделить все поля, кроме Charact, Photo и, может быть. Num. Выделив поля, щелкните на ОК и вы вернетесь к основному окну Редактора Полей, но в нем уже будет содержаться список добавленных полей (рис. 2 в).



А) б) в)

Рис.2 Редактор Полей: исходное состояние (а), окно выбора полей (б), состояние после выбора (в)

Эти поля будут соответствовать колонкам таблицы. Изменить последовательность их расположения можно, перетащив мышью идентификатор какого-то поля на нужное место. Как мы увидим далее, те поля, которые не должны отображаться в таблице, могут быть сделаны невидимыми.

Выделите в списке какое-то поле и посмотрите его свойства в Инспекторе Объектов. Вы увидите, что каждое поле — это объект, причем его класс зависит от типа поля: **TStringField, TSmallintField, TBoo-leanField** и т.п. Все эти классы являются производными от **TField** — базового класса полей. Таким образом, каждое поле является объектом и обладает множеством свойств. Рассмотрим основные из них, которые чаще всего необходимо задавать.

Свойство **Alignment** определяет выравнивание отображаемого текста внутри колонки таблицы: влево, вправо или по центру.

Свойство **DisplayLabel** соответствует заголовку столбца данного поля. Например, для поля Fam значение **DisplayLabel** можно задать равным «Фамилия», для Nam — «Имя» и т.д.

Свойство **DisplayWidth** определяет ширину колонки — число символов.

Свойства **EditMask** для строк и **EditFormat** для чисел определяют форматы отображения данных.

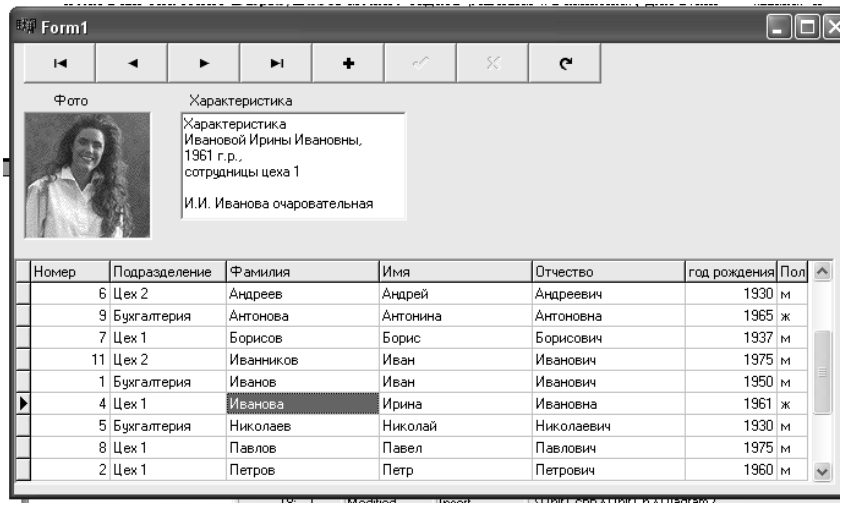
Для логических полей (в нашем примере для поля Sex) очень важным свойством является **Display-Values.** Это свойство определяет, какие значения должны отображаться, если поле имеет значение **true** или **false.** Отображаемые значения разделяются точкой с запятой. Первым пишется значение, соответствующее **true.** Например: «м;ж» или «мужской; женский».

Свойство **ReadOnly,** установленное в **true,** запрещает пользователю вводить в данное поле значения. Свойство **Visible** определяет, будет ли видно пользователю соответствующее поле. В нашем примере, вероятно, можно задать **Visible = false** для поля Num, если вы его не исключили из списка полей **Tablel.**

После установки всех необходимых свойств приложение уже приобретает приемлемый вид.

Отметим еще одну особенность Редактора Полей — возможность перетаскивать из него поля на **форму** с помощью мыши. Захватите в Редакторе Полей какое-нибудь поле, например, Fam, и перетащите его на форму. На форме появится связанный с этим полем компонент типа **TDBEdit** и метка, содержащая его свойство **DisplayLabel.** При переносе на форму булевского поля Sex на ней появится связанный с этим полем индикатор — компонент типа **TDBCheckBox.**

При перетаскивании полей Charact и Photo появятся соответственно связанные с этими полями компоненты типов **TDBMemo и TDBImage.** И во всех этих компонентах, если вы переключите свойство **Active** компонента **TTable в true,** сразу отобразятся соответствующие данные из таблицы.



Номер	Подразделение	Фамилия	Имя	Отчество	год рождения	Пол
6	Цех 2	Андреев	Андрей	Андреевич	1930	м
9	Бухгалтерия	Антонова	Антонина	Антоновна	1965	ж
7	Цех 1	Борисов	Борис	Борисович	1937	м
11	Цех 2	Иванников	Иван	Иванович	1975	м
1	Бухгалтерия	Иванов	Иван	Иванович	1950	м
4	Цех 1	Иванова	Ирина	Ивановна	1961	ж
5	Бухгалтерия	Николаев	Николай	Николаевич	1930	м
8	Цех 1	Павлов	Павел	Павлович	1975	м
2	Цех 1	Петров	Петр	Петрович	1960	м

Рис.3. Приложение с установленными свойствами, именами полей, удаленными кнопками редактирования в навигаторе и т.д.

Практическая работа №5.

1.Компонент MainMenu – главное окно, настройка, основные свойства.

2.Мастер форм – Database ->FormWizard

1 .Расположите на форме компонент MainMenu. Основное свойство компонента MainMenu – Items. Его заполнение производиться с помощью конструктора Меню, вызываемого двойным щелчком на компоненте MainMenu или нажатием кнопки с многоточием рядом со свойством Items в окне инспектора проектов. В результате

откроется окно, вид которого представлен на рис. 1. В этом окне вы можете спроектировать все меню. На рис. 2. показано то меню, которое соответствует проектируемому на рис.1.

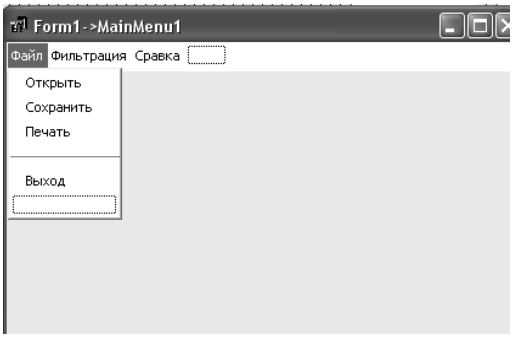


Рис. 1. Окно конструктора Меню

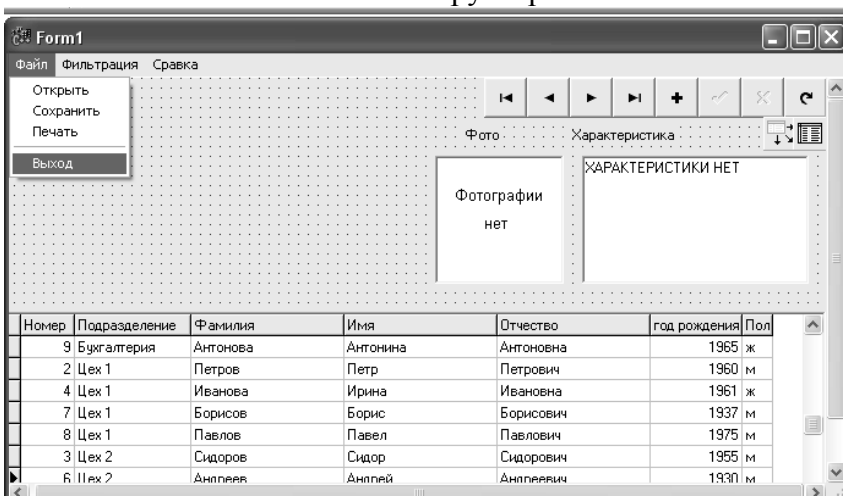


Рис.2. Результат конструирования меню.

Свойство **Caption** обозначает надпись раздела. Если в качестве обозначения поставить знак «-», то вместо раздела в меню появится разделитель.

Свойство **ShortCut,** определяет клавиши быстрого доступа к разделу меню – горячие клавиши, с помощью которых пользователь даже на заходя в меню может открыть интересующий его отдел.

Так же предусмотрена возможность ввода в меню изображений, за это ответственны свойства Bitmap и ImageIndex. Первое из них позволяет непосредственно ввести изображения в раздел, выбрав его из указанного вами файла. Второй – позволяет указать индекс изображения, хранящегося во внешнем документе ImageList.

Основное событие раздела OnClick, возникающее при щелчке пользователя на разделе или при нажатии «горячих» клавиш быстрого доступа.

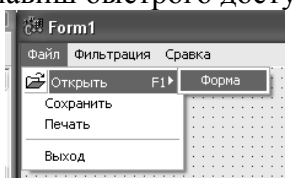


Рис. 3. Добавление в меню «Открыть»картинки, горячей клавиши, а также SubMenu «Форма»

2.Задание:

— Создайте с помощью мастера форм вторую форму, следуйте инструкциям мастера.

— Поменяйте имена колонок. Создайте еще одну форму, отличную от первой.

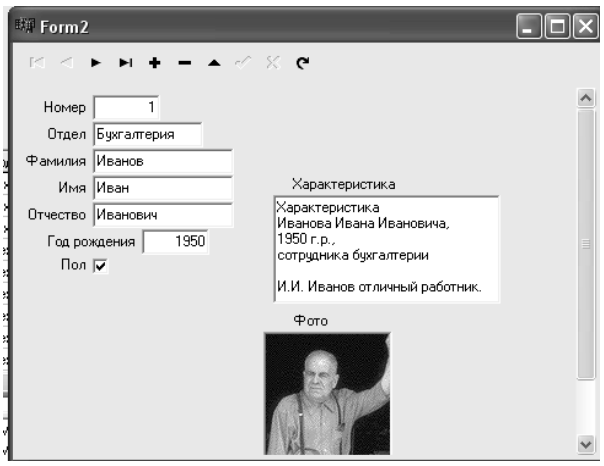


Рис.4. Форма, созданная с помощью мастера форм.

Практическая работа №6. Фильтрация данных

Компонент **Table** позволяет не только отображать, редактировать и упорядочивать данные, но и отсортировывать записи по определенным критериям. Пользователю в нашем примере могло бы захотеться иметь возможность просматривать не всю базу данных, а отдельно записи по тому или иному отделу, или, например, просмотреть записи сотрудников, имеющих возраст в определенном диапазоне.

Фильтрация может задаваться свойствами **Filter, Filtered и FilterOptions** компонента **Table.** Свойство **Filtered** включает или выключает использование фильтра. А сам фильтр записывается в свойство **Filter** в виде строки, содержащей определенные ограничения на значения полей. Например, вы можете задать в свойстве **Filter**

```
Dep='Цех 1'
```

установить свойство **Filtered в true, и** увидите, что уже в процессе проектирования в таблице отобразятся только те записи, в которых поле Dep имеет значение «Цех 1". Обратите внимание на то, что в условии фильтрации строки заключаются в одинарные, а не в двойные кавычки.

В условиях сравнения строк можно использовать символ звездочки «*», который как и в обычных шаблонах означает: «любое количество любых символов». Например, фильтр

```
Dep='Цех*'
```

приведет к отображению всех записей, в которых значение поля Dep начинается о «Цех». В нашем примере будут отображены записи, относящиеся к первому и второму цехам. Но для того, чтобы это сработало, надо, чтобы в опциях, содержащихся в свойстве **FilterOptions** была выключена опция **foNoPartialCompare,** запрещающая частичное совпадение при сравнении (эта опция выключена по умолчанию). Другая опция в свойстве **FilterOptions — foCaseInsensitive** делает сравнение строк нечувствительным к регистру, в котором записано условие фильтра. Если включить эту опцию, то слова «Цех 1« и »цех 1« будут считаться идентичными.

При записи условий можно использовать операции отношения *s», **>, >==, <, <==",** а также логические операции and, or и not. Например, вы можете написать фильтр

```
(Dep='Цех 1')and(Year_b<=1970)and(Year_b>=1940)
```

и отобразятся записи сотрудников цеха **1,** чей год рождения лежит в заданных пределах.

Конечно, свойства, определяющие фильтрацию, можно задавать не только в процессе проектирования, но и программно, во время выполнения. Давайте введем такую возможность в наше приложение.

Осталось написать операторы, обеспечивающие фильтрацию. Ниже приведен соответствующий текст, для настроек компонента MainMenu:

— **Фильтрация по подразделению**:

```
void __fastcall TForm1::N15Click(TObject *Sender)
```

{
Table1->Filtered = true;
Table1->Filter = "Dep='Бухгалтерия'";
}//для фильтрации по Цех 1, Цех 2 аналогично;

Для фильтрации по фамилии пометите на форму по компонента Label, Edit. Задайте им имена и текст соответственно рис.1.

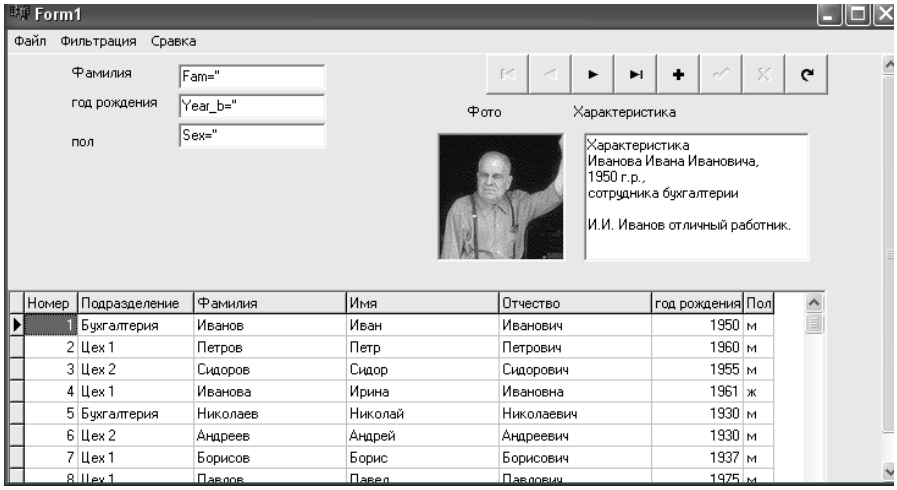


Рис.1. Приложение с поиском записей

— **Фильтрация по фамилии:**

void __fastcall TForm1::Label3Click(TObject *Sender)
{
Form1->Table1->Filtered = true;
Form1->Table1->Filter = Edit1->Text;
}// для поиска по году рождения и полу аналогично

— За пунктом меню **«Закрыть»** закрепим следующий оператор:

Form1->Close();

Рис. 2. Конечное приложение для п/р№6

Практическая работа №7. Кэширование изменений

По умолчанию все изменения в наборе данных, завершаемые методами **Post, Insert, Delete** и т.п., заносятся в базу данных. Однако, возможен и другой режим работы, при котором все изменения, вносимые пользователем в записи, кэшируются — т.е. сохраняются в памяти локально. В этом случае пользователь работает не с реальными данными, а с их копиями. И только по специальной команде все внесенные пользователем изменения заносятся в базу данных.

Режим кэширования определяется свойством **CachedUpdates** компонента **Table.** По умолчанию это свойство равно **false** и кэширование не производится. Если установить его в **true,** то все изменения набора данных будут кэшироваться. Они передаются в базу данных только после выполнения метода **Apply-Updates.** Если же после множества изменений выполнить метод **CancelUpdates,** то все изменения, произведенные после последнего выполнения **ApplyUpdates,** отменяются.

Метод **ApplyUpdates** только передает изменения в базу данных на сохранение, но еще не фиксирует изменения данных. Метод **CommitUpdates,** который должен выполняться после **ApplyUpdates,** очищает буфер кэша, после чего он готов для приема новой порции информации, а измененные данные фиксируются в базе данных. Если при фиксации данных произошла ошибка, можно применить метод **Rollback,** который аннулирует все изменения.

Проверьте все это в своем приложении вид которого показан на рис. **1.** Индикатор Кэширование переключает режим кэширования. Кнопка Фиксация фиксирует все сделанные изменения в базе данных. Кнопка Отмена отменяет сделанные изменения. Когда приложение закрывается, надо проверить, не работало ли оно в режиме кэширования и не было ли сделано изменений, которые не зафиксированы в базе данных. Если были, то следует спросить пользователя о необходимости их сохранения и при положительном ответе зафиксировать изменения.

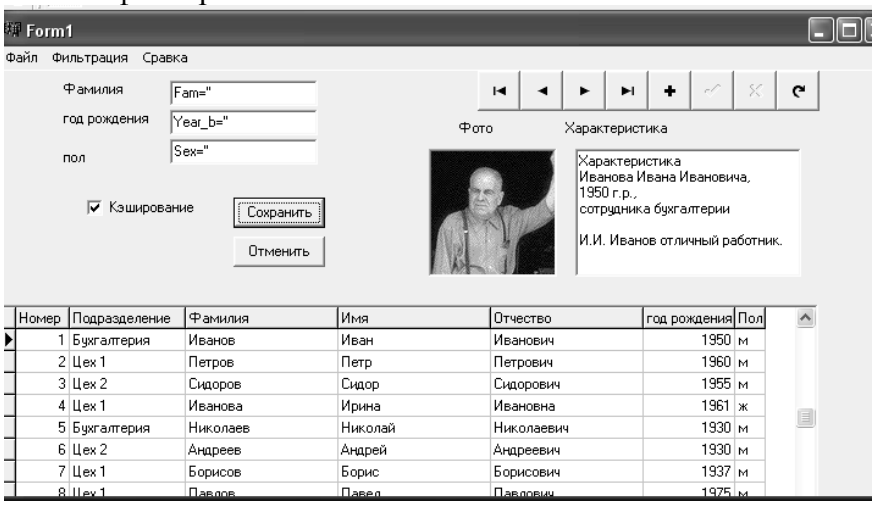


Рис.1. Приложение с оператором кэширования.

Коды этого приложения очень простые. Для таблицы **Tablel** свойство **CachedUpdates** первоначально надо задать равным **true.** В приложение можно ввести переменную **modif,** которая будет фиксировать наличие не сохраненных изменений:

```
bool  modif = false;

void __fastcall TForm2::CheckBox1Click(TObject *Sender)
{
Table1->Active = false;
Table1->DatabaseName="dbP";
Table1->TableName="Pers.db";
Table1->Active = true;
Table1->CachedUpdates = ! Table1->CachedUpdates;
```

```
BApplyUpdates->Enabled = Table1->CachedUpdates;
BCancelUpdates->Enabled = Table1->CachedUpdates;
if (Table1->CachedUpdates)bool modif = false;
}
```

В обоих обработчиках переменная **modif** сбрасывает в **false,** фиксируя отсутствие не сохраненных изменений.

Он изменяет режим кэширования таблицы, делает доступными или недоступными в зависимости от режима кнопки Фиксация (ее имя **— BApplyUpdates)** и Отмена (ее имя — **BCancelUpdates)** и, если включается режим кэширования, то сбрасывается в **false** значение **modif.**

Для события **AfterEdit** компонента **Tablel** можно предусмотреть обработчик:

```
void __fastcall TForm1::Table1AfterEdit(TDataSet *DataSet)
{
bool modif;
if (Table1->CachedUpdates) modif = true;
modif = false;
}
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
bool modif = false;
if (Table1->CachedUpdates) modif = true;
}
void __fastcall TForm1::Table1AfterInsert(TDataSet *DataSet)
{
bool modif = false;
if (Table1->CachedUpdates) modif = true;
}
```

который фиксирует в переменной **modif** факт редактирования записи.

Обработчик события **OnClick** кнопки Фиксация имеет вид:

```
void __fastcall TForm1::BApplyUpdatesClick(TObject *Sender)
{
Table1->ApplyUpdates();
Table1->CommitUpdates();
bool  modif = false;
}
```

Обработчик события **OnClick** кнопки Отмена имеет вид:

```
void __fastcall TForm2::BCancelUpdatesClick(TObject *Sender)
{
Table1->CancelUpdates();
 bool modif = false;
}
```

В обоих обработчиках переменная **inodif** сбрасывает в **false,** фиксируя отсутствие не сохраненных изменений. Обработчик события **OnCIoseQuery** формы имеет вид:

```
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{
bool modif =  true;
if (Table1->CachedUpdates && modif)
switch (Application->MessageBox(
                        "Сохранить изменения в базе данных?",
```

```
                    "Подтвердите изменения",
                    MB_YESNOCANCEL + MB_ICONQUESTION))
{
case IDYES: Table1->ApplyUpdates();
        break;
case IDCANCEL: CanClose = false;
        break;
case IDNO: Table1->CancelUpdates();
}
}
```

Практическая работа №8. Приложения с несколькими связанными таблицами. Оператор для связи форм.

1. Связь головной и вспомогательной таблиц

Создайте в своем приложении одну новую форму, и располагайте компоненты на ней.

Предыдущие приложения общались с одной таблицей. Теперь посмотрим, как строить приложения с несколькими связанными друг с другом таблицами.

Две таблицы могут быть связаны друг с другом по ключу. Одна из этих связанных таблиц является головной (master), а другая — вспомогательной, детализирующей (detail). Например, мы хотим построить приложение, в котором имеется таблица **Dep,** содержащая список подразделений учреждения (поле Dep) и характеристику этих подразделений (поле Proizv булева типа, в котором **true** означает «Производство», а **false —** «Управление»). И хотим, чтобы пользователь, перемещаясь по этой таблице, видел не только характеристику подразделения, но и список сотрудников этого подразделения, т.е. записи из таблицы **Pers,** в которых значение поля Dep совпадает со значением поля Dep в текущей записи первой таблицы.

В этом случае головной является таблица **Dep,** вспомогательной — таблица **Pers,** а ключом, определяющим их связь, являются поля Dep из обеих таблиц.

Откройте новое приложение и разместите на нем 2 комплекта **Table, DataSource** и средств отображения данных. Результат может выглядеть например так, как показано на рис. 1. Комплект обычным образом настраивается на первую, головную (master) таблицу — **Dep.** Таблица должна быть индексирована по полю Dep, которое будет ключом для связи таблиц.

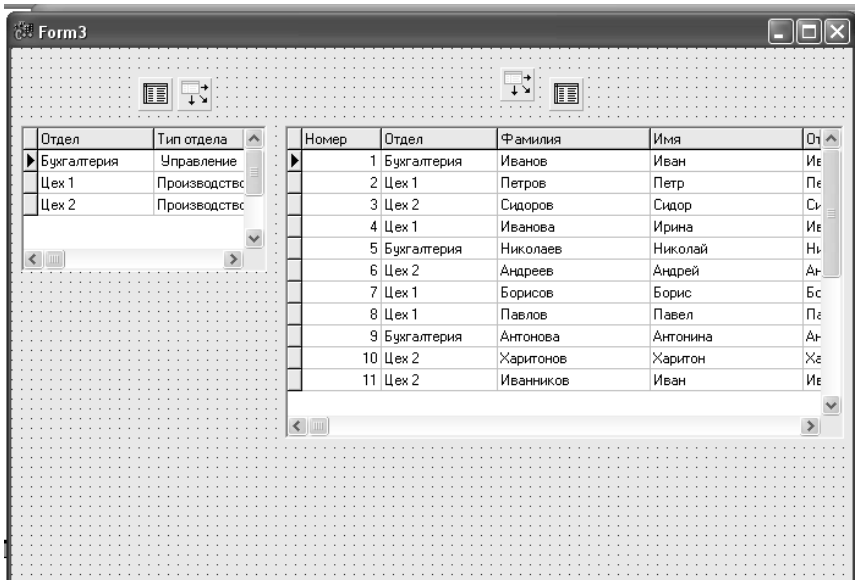


Рис.1. Простое приложение, независимо работающее с двумя таблицами данных.

Второй комплект компонентов так же использует для отображения данных компонент **DBGrid.** Комплект настраивается на вторую вспомогательную таблицу — Pers. Для таблицы должен быть задан индекс, содержащий ключевое поле связи Dep. После того, как все это сделано, можете выполнить приложение, чтобы убедиться, что все сделано правильно. Пока вы имеете две несвязанные друг с другом таблицы, по которым можете перемещаться независимо.

Теперь свяжите эти таблицы. Разорвите временно связь с базой данных во втором комплекте, настроенном на вспомогательную таблицу **Pers** (установите Active = false). Далее в свойстве MaSterSource компонента Table, настроенного на вспомогательную таблицу, установите имя головной таблицы. После этого щелкните на свойстве MasterFields. Откроется окно редактора связей полей (Field Link Designer). Его вид приведен на рис. 2. В нем слева в окне Detail Fields расположены имена полей вспомогательной таблицы, но только тех, по которым таблица индексирована. Именно поэтому надо индексировать таблицу так, чтобы индекс включал ключевое поле связи Dep. Справа в окне Master Fields расположены поля головной таблицы. Выделите в одном и другом окне ключевое поле (в нашем случае Dep). При этом активизируется кнопка Add, и после щелчка на ней ключевые поля переносятся в нижнее окно Joined Fields — соединяемые поля. Если ключ составной (например, фамилия, имя, отчество) — эта операция повторяется для других полей. В конце формирования связей щелкните на OK и в свойстве MasterFields компонента Table появится текст — связанные поля. После этого можете восстановить связь с базой данных **(Active** = true) и запустить приложение.

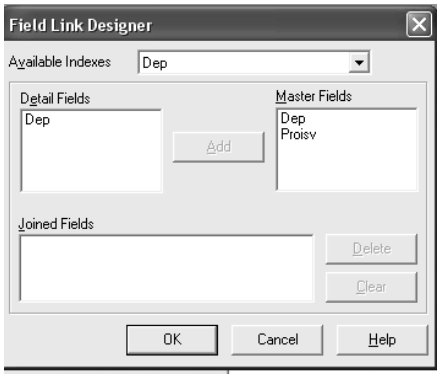


Рис.2. Окно редактора связей полей головной и вспомогательной таблиц

2. Для того, чтобы ваша форма работала в приложении необходимо в компонент MainMenu в строку «Открыть» добавить «Связанные таблицы» и написать на нее оператор:

Form 3>ShowModal();

Так же необходимо в директиве препроцессора сделать следующее объявление:

#include "Unit3.h"

Вы увидите (рис. 3), что в зависимости от того, какую запись вы выделяете в списке отделов, вам отображается список сотрудников этого отдела. Таким образом, курсор скользит по головной таблице, а вспомогательная таблица отображает только те записи, в которых ключевые поля совпадают с ключевыми полями головной таблицы.

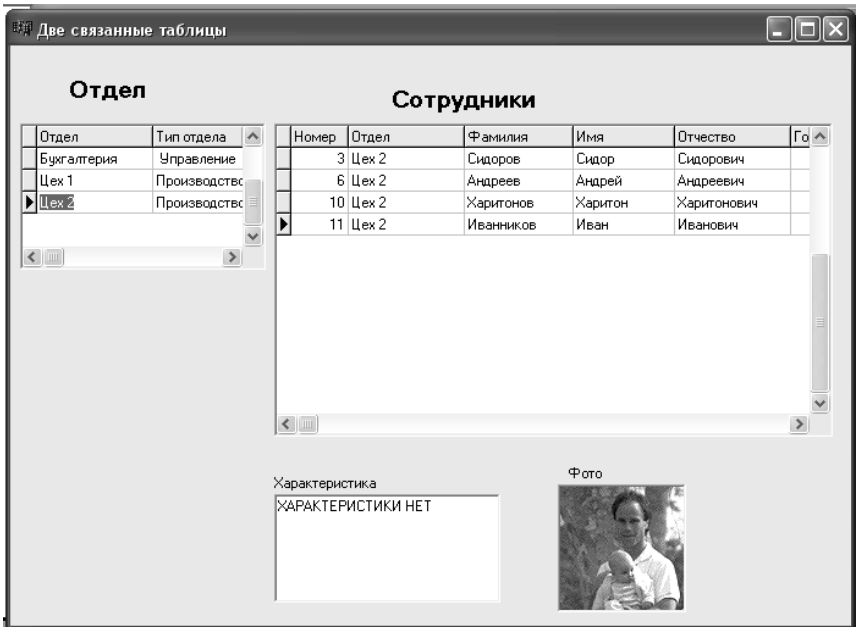


Рис.3. Приложение с двумя связаннми таблицами

Практическая работа № 9. Поиск записей в таблице.

Одна из важнейших для пользователя операций с базами данных — поиск записей по некоторому ключу. Существует несколько методик поиска записей, которые можно назвать **SetKey, FindKey, Lookup и Locate.**

Для выполнения практической работы необходимо создать новое приложение с таблицей.

Начнем с методики **SetKey.** Для ее применения таблица предварительно должна быть индексирована по тому полю, по которому должен будет проводиться поиск. Затем таблица устанавливается в состояние поиска dsSetKey. Для этого используется метод SetKey. В состоянии dsSetKey набор данных воспринимает последующий оператор присваивания значения полю не как присваивание, а как задание ключа поиска. Поэтому после установки состоянии dsSetKey оператором присваивания устанавливается требуемое значение ключа поиска по интересующему полю. В заключение методом GotoKey курсор переводится на запись, в которой значение указанного поля равно ключу. Если таких записей несколько, то курсор переводится на первую из них. Если соответствующая запись не находится, то метод GotoKey возвращает false. Для полей типа строк лучше использовать не метод GotoKey, а метод GotoNea-rest. Этот метод перемещает курсор на первую запись, значение поля в которой максимально близко к ключу. Т.е. он сработает и тогда, когда совпадение не полное. Метод GotoNearest можно применять и к цифровым полям. В этом случае он переместит курсор на первую запись, значение поля в которой больше или равно заданному значению ключа-

Например, если вы хотите найти первую запись, в которой год рождения (поле Year_b) равен заданному пользователем в окне редактирования **EYear,** вы можете написать операторы:

```
Table1->IndexFieldNames = "Year_b";
Table1->SetKey();
Table1->FieldByName("Year_b")->AsString = EYear->Text;
if (! Table1->GotoKey())
 ShowMessage("Запись не найдена");
```

Первый оператор индексирует набор данных по полю Year_b, второй переводит набор данных в состояние **dsSetKey,** третий задает ключ поиска, а четвертый осуществляет переход к соответствующей. записи или сообщает об отсутствии такой записи.

Если вы хотите найти в таблице сотрудника по его фамилии, заданной пользователем в окне редактирование **EFam,** вы можете выполнить

```
Table1->IndexFieldNames = "Fam";
Table1->SetKey();
Table1->FieldByName("Fam")->AsString = EFam->Text;
Table1->GotoNearest();
```

код:

Даже если точно такой фамилии не найдется, курсор перейдет на наиболее похожую (совпадающую по первым символам).

Методика поиска FindKey еще богаче по своим возможностям. В этой методике таблица также должна быть проиндексирована по тем ключевым полям, по которым осуществляется поиск. Функция Find-**Key** определена следующим образом:

```
bool __fastcall FindKey(const System::TVarRec * KeyValues, const int KeyValues_Size);
```

Параметр **KeyValues** представляет собой открытый массив: разделяемый запятыми список значений полей, по которым индексирован набор данных, в той последовательности, в которой они входят в индекс. При этом не обязательно перечислять все поля — достаточно перечислить первое или несколько первых. Параметр **KeyValues_Size** определяет индекс последнего поля в массиве, участвующего в поиске. Поскольку индексы начинаются с 0, то **KeyValues_Size** на единицу, меньше количества полей, участвующих в поиске.

Вместо **FindKey** для полей строкового типа можно использовать аналогичный метод **FindNearest,** обеспечивающий переход к наиболее совпадающей строке, если полного совпадения не получено. Объявление и параметры этого метода те же, что и в методе **FindKey.** Метод **FindNearest** можно применять и к цифровым полям. В этом случае он переместит курсор на первую запись, значение полей в которой больше или равны заданных значений ключей.

- Для методов **FindKey и FindNearest** удобно применять макрос **OPENARRAY,** создающий временный открытый массив и определяющий его размер:

```
OPENARRAY(TVarRec,(список ключей))
```

Макрос **OPENARRAY** может воспринимать список, включающий до 19 элементов, разделенных запятыми.

Рассмотрим примеры. Пусть мы хотим выполнить тот же поиск по фамилии, что и приведенный выше. В данном случае соответствующий код может иметь вид:

```
Table1->IndexFieldNames = "Fam";
Table1->FindNearest(&TVarRec(EFam->Text),0);
```

Если мы хотим выполнить аналогичный поиск, но нас интересует не просто один из сотрудников с заданной фамилией, а работающий в конкретном подразделении, заданном пользователем в окне редактирования EDep, то поиск можно осуществить с помощью макроса **OPENARRAY** следующим образом:

```
Table1->IndexFieldNames = "Dep;Fam";
Table1->FindNearest(OPENARRAY(TVarRec,(EDep->Text,EFam->Text)));
```

Первый оператор индексирует таблицу по полям Dep и Fam, а второй задает ключи для этих полей. Конечно, не надо индексировать таблицу каждый раз перед осуществлением поиска. Достаточно выполнить это один раз.

Приведем еще один пример. Пусть мы хотим предоставить пользователю ускоренный поиск фамилии. Пользователь будет набирать фамилию в окне **EFam** и при вводе каждого нового символа курсор должен перемещаться на запись, наиболее близко совпадающую с уже введенными символами. Для этого сначала надо индексировать таблицу по полю Fam. А затем достаточно в событие **OnChange** окна редактирования **EFam** вставить обработчик

```
Table1->FindNearest(&TVarRec(EFam->Text),0);
```

Теперь остановимся на методе **Locate.** Этот метод объявлен следующим образом;

```
bool __fastcall Locate(const System::AnsiString KeyFields,
                       const System::Variant &KeyValues, TLocateOptions Options);
```

В качестве первого параметра **KeyFields** передается строка, содержащая список ключевых полей. В качестве второго параметра передается **KeyValues** — массив ключевых значений. А третий параметр **Options** является множеством опций, элементами которого могут быть **loCaseInsensitive** — нечувствительность поиска к регистру, в

котором введены символы, и **loPartialKey** — допустимость частичного совпадения. Метод возвращает **false,** если искомая запись не найдена.

В простейшем случае применение **Locate** отличается от рассмотренных ранее методов только отсутствием необходимости индексировать набор данных определенным образом. Например, поиск (в том числе и рассмотренный выше ускоренный поиск) по фамилии может осуществляться операторами

```
TLocateOptions SearchOptions;
SearchOptions << loPartialKey << loCaseInsensitive;
Table1->Locate("Fam",EFam->Text, SearchOptions);
```

причем он будет работать независимо от того, как индексирована база данных. Впрочем, если набор данных соответствующим образом индексирован, то поиск производится быстрее. Приведенный код можно сократить до двух операторов:

```
TLocateOptions SearchOptions;
Table1->Locate("Fam", EFam->Text, SearchOptions<<loPartialKey<<loCaseInsensitive);
```

При поиске по нескольким полям можно воспользоваться функцией **VarArrayOf,** которая формирует тип **Variant** из задаваемого ей массива параметров любого типа. Например, рассмотренный ранее поиск по отделу и фамилии может быть осуществлен операторами

```
TLocateOptions SearchOptions;
Variant locvalues[] = {EDep->Text, EFam->Text};
Table1->Locate("Dep;Fam", VarArrayOf(locvalues,1),
               SearchOptions<<loPartialKey<<loCaseInsensitive);
```

И в заключение о последнем методе поиска — **Lookup.** Этот метод определен следующим образом:

```
System::Variant __fastcall Lookup(const System:: AnsiString KeyFields,
                                  const System::Variant &KeyValues,
                                  const System::AnsiString ResultFields);
```

Первые два параметра аналогичны методу **Locate.** Третий параметр — строка, перечисляющая поля, значения которых возвращаются в виде массива **Variant.** Если не найдено соответствующей записи, функция возвращает **false.**

Например, если вы хотите найти запись, относящуюся к сотруднику, фамилия которого указана в окне **EFam,** и вывести в окно **EDep** название отдела, в котором он работает, то эти операции можно осуществить следующим образом:

```
EDep->Text = Table1->Lookup("Fam",EFam->Text,"Dep");
```

Метод **Lookup** не изменяет положения курсора. Он только возвращает значения указанных полей. Они могут использоваться любым способом. В частности, они могут использоваться вместо параметра **KeyValues** в другом операторе **Lookup** или **Locate.** Это открывает широкие возможности формирования сложных запросов по нескольким таблицам.

Практическая работа №10. Создание отчетов.

Для выполнения практической работы создайте в своем приложении новую форму, объявите ее модуль в директиве препроцессора.

QuickReport использует генератор отчетов, состоящих из множества полос. Полоса (band) — это часть отчета или раздел, содержащий некоторый текст, изображения, графики, диаграммы и т.п. Полоса является контейнером для других компонентов, вносящих в отчет информацию или графику. Если полоса и размещенные на ней компоненты связаны с базой данных, то содержание этой полосы печатается столько раз, сколько соответствующих записей имеется в источнике данных. Таким образом, достаточно расположить компоненты, связанные с данными, на полосе, а печатаемые значения и их количество будут автоматически управляться базой данных.

Как в компонентах, связанных с данными, вы могли задавать головную и вспомогательные таблицы, так и между полосами можно задавать аналогичные связи. Таким образом, все возможности, реализуемые в приложениях, реализуются с тем же успехом и в отчетах. Давайте зададимся задачей построить отчет по уже многократно использованной нами базе данных **dbP,** первая страница которого показана на рис. 9.50,

Отчет, озаглавленный «КАДРОВЫЙ СОСТАВ ПРЕДПРИЯТИЯ ПО СОСТОЯНИЮ НА ...» должен включать в себя следующие разделы:

Заголовок раздела	Информация, распечатываемая в разделе
ОТДЕЛЫ	Список всех отделов
СПИСОЧНЫЙ СОСТАВ ОТДЕЛОВ	Состоит из ряда подразделов, каждый из которых имеет свой подзаголовок «СОТРУДНИКИ ОТДЕЛА ...», после которого следует список фамилий сотрудников
ЛИЧНЫЕ ДЕЛА	Состоит из ряда подразделов, каждый из которых имеет свой подзаголовок «СОТРУДНИКИ ОТДЕЛА .,,», после которого следует информация о фамилии, имени, отчестве сотрудника, его пол, год рождения, фотография сотрудника и его характеристика

В разделе «ЛИЧНЫЕ ДЕЛА» каждый подраздел «СОТРУДНИКИ ОТДЕЛА ...» должен начинаться с вой страницы. Все страницы должны иметь нижний колонтитул, в котором слева печатается надписи «Кадровый состав», а справа — номер страницы.

Строя этот отчет мы по ходу дела рассмотрим необходимые нам свойства различных компонентов. Основным компонентом, на котором размещаются все полосы отчета, является **QuickRep. Открой новое** приложение и разместите на нем этот компонент. Большинство свойств и методов **QuickRep** рассмотрены в разделе 4.8.3. Не упоминалось там только об одном свойстве —**DataSet,** определяющем выбор данных. Этим набором может являться компонент типа **TTable, TQuery** и т.п. Поэтому для того, чтобы можно было построить наш отчет, поместите на форму компонент типа **TTable,** назовите его **TDep i** свяжите его с таблицей **Dep** базы данных **dbP.** Перенесите на форму компонент типа **TDataSource, нам** вите его **DSDep** и свяжите с **TDep.** Перенесите на форму еще один компонент типа **TTable,** назовите еп **TPers** и свяжите его с таблицей **Pers** базы данных dbP. Установите между двумя таблицами обычную связь, чтобы **TDep** была головной таблицей, а **TPers** связывалась с ней по полю Dep

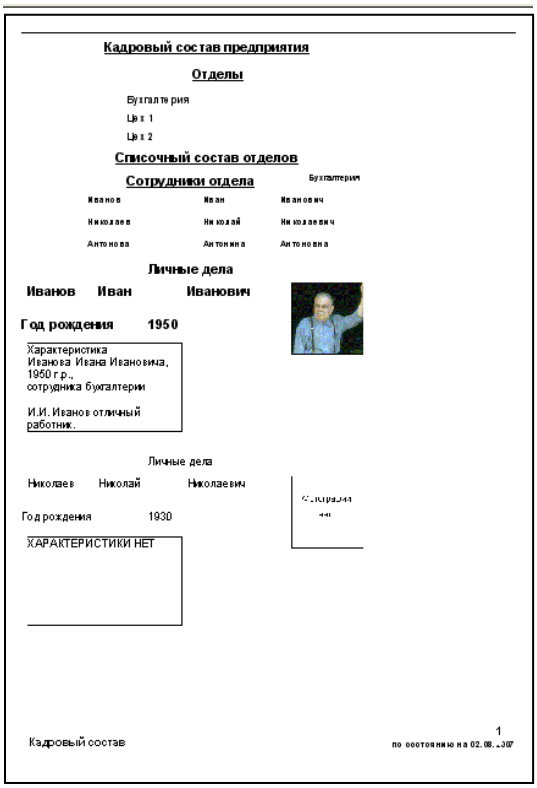


Рис.1.Первая страница разрабатываемого отчета

Установите в свойстве **DataSet** компонента **QuickRep** имя компонента **TDep,** связав его тем самым с головной таблицей. Теперь рассмотрим одно из основных свойств компонента **QuickRep — Bands. Оно** имеет ряд подсвойств:

HasTitle Имеется полоса заголовка отчета, которая печатается один раз в начале отчета.

HasDetail Имеется полоса детализации, которая печатается столько раз, сколько записей в нее передается.

HasPageHeader Имеется верхний колонтитул (заголовок) на каждой странице отчета **HasPageFooter** Имеется нижний колонтитул на каждой странице отчета.

HasColumnHeater Имеется заголовок печатаемой таблицы.

Для построения нашего отчета надо установить в **true** подсвойства **HasTitle и HasPageFooter** — полосы заголовка и нижнего колонтитула. На компоненте **QuichRep** появятся слабо видимые полосы с соотвующими надписями. На них и будут размещаться компоненты, которые отображают ту или иную информацию.

Разместите на полосе заголовка метку **QRLabel** и в ее свойстве **Caption** запишите первую часть заловка отчета

« КАДРОВЫЙ СОСТАВ ПРЕДПРИЯТИЯ ». Выровняйте эту метку по центру полосы (см. раз-вв-т 4.8.3). Ниже поместите компонент **QRSysData.** Его свойство **Data** установите равным **qrsDate, a** свойство **Text** задайте равным « ПО СОСТОЯНИЮ НА ». Это обеспечит (см. раздел 4.8.3) внесение в заголовок текущей даты.

На полосе нижнего колонтитула разместите два компонента: метку **QRLabel,** выровненную влево, с надписью, которая должна появляться в колонтитуле, и компонент **QRSysData,** задав его свойство **Data** равным **qrsPageNumber** и выровняв вправо. Этот компонент будет отображать номер страницы. Все эти операции подробно рассмотрены в разделе 4.8.3.

Можете щелкнуть правой кнопкой мыши на компоненте QuickRep и, выбрав из всплывшего меню команду Preview, полюбоваться достигнутым результатом. Теперь займемся собственно текстом отчета. Для этого надо разместить на компоненте **QuickRep** дополнительные полосы. Мы будем это делать, перенося на него соответствующие компоненты полос со страницы QReport палитры компонентов.

Наиболее универсальным является компонент **QRBand.** Его основное свойство **BandType** которое может иметь следующие значения:

rbChild	дочерняя полоса
rbColumnHeader	полоса заголовка таблицы
rbDetail	полоса детализации
rbGroupFooter	нижний текст группы
rbGronpHeader	заголовок группы
rbOverlay	оверлейная полоса
rbPageFooter	нижний колонтитул страницы
rbPageHeader	верхний колонтитул страницы
rbSubDetail	полоса дополнительной детализации
rbSammary	полоса итогов
rbTitIe	полоса заголовка

Как видно, эта полоса может использоваться вместо компонентов некоторых полос других типов. Но все-таки использовать специализированные полосы удобнее.

Для первого раздела нашего отчета — ОТДЕЛЫ нам надо организовать печать заголовка раздела далее циклическую печать названий отделов. Это может сделать полоса детализации в виде компонент» **QRSubDetaiI.** Перенесите ее на **QuickRep** и давайте рассмотрим некоторые ее свойства.

Свойство **DataSet** определяет набор данных, к которому должна подключаться

полоса. Поскольку в данном случае она должна обеспечивать просмотр всех записей таблицы **Dep,** в свойстве **DataSet** надо указать таблицу **TDep.** Этого достаточно, чтобы обеспечить циклическую печать полосы.

Свойство полосы **Bands** имеет два подсвойства: **HasFooter** определяет полосу, которая будет напечатана после окончания циклов, и **HasIIeader** определяет полосу заголовка, которая будет напечатана перед началом циклической печати. Нам требуется установить в **true** только **HasHeader.** В появившейся полосе Group Header поместите метку **QRLabel** с заголовком раздела, а в самой полосе детализации поместите компонент **QRDBText** — метку, связанную с данными. В ней, как и в других компонентах, связанных с данными, надо задать набор данных **DataSet** и его поле **DataField,** которое должно отображаться. В данном случае **DataSet= TDep и DataField = Dep.**

Можете опять осуществить предварительный просмотр отчета и убедиться, что первый раздел отчета печатается правильно.

Во втором разделе — СПИСОЧНЫЙ СОСТАВ ОТДЕЛОВ нам надо организовать два вложенных цикла печати: внешний по отделам и внутренний по сотрудникам очередного отдела. Для внешнего цикла повторяете все операции, которые делали для предыдущего раздела: переносите в отчет компонент **QRSubDetail,** связываете его с таблицей **TDep,** устанавливаете в **true** подсвойство **HasHeader** свойства **Bands.** На полосе заголовка раздела помещаете метку **QRLabel** с заголовком «СПИСОЧНЫЙ СОСТАВ ОТДЕЛОВ». На полосе детализации размещаете метку **QRLabel** с текстом «СОТРУДНИКИ ОТДЕЛА» и рядом с ней — метку **QRDBText,** настроив ее на поле Dep таблицы **TDep.**

Теперь нам надо организовать вложенный цикл, чтобы для каждого отдела пробегать по относящимся к нему записям таблицы **Pers.** Для задания таких вложенных циклов в компоненте **QRSubDetail** предусмотрено свойство **Master.** Оно определяет головную полосу для данной полосы. Это аналогично тому, как задается головная таблица для вспомогательной. Свойство **Master** позволяет организовывать внутренние циклы печати для каждого напечатанного значения головной полосы. Свойство **PrintBefore** определяет в этом случае, печатаются ли значения внутреннего цикла до (при **PrintBefore = true**) или после печати очередного значения внешнего цикла.

Чтобы все это реализовать, поместите в отчет еще один к компонент **QRSubDetail** и свяжите его с таблицей **TPers.** Его свойство **Master** установите в **QRSubDetail2** — имя второй полосы **QRSubDetail,** являющейся головной во втором разделе. На эту новую полосу поместите метку **QRDBText,** настроив ее на поле Fam таблицы **TPers.**

Запустите предварительный просмотр отчета и убедитесь, что все работает как надо. Теперь третий раздел отчета «ЛИЧНЫЕ ДЕЛА» не представит для вас сложности. В нем точно так же надо организовать внешний цикл печати по отделам и внутренний — по сотрудникам отдела. В этом втором цикле для отображения фотографий надо использовать компонент **QRDBImage,** а для печати характеристик — компонент **QRMemo.** Оба эти компонента должны быть настроены на соответствующие поля таблицы **TPers.** Осталось выполнить одно требование задания — каждый подраздел раздела «ЛИЧНЫЕ ДЕЛА», относящийся к новому подразделению, должен начинаться с новой страницы. Это можно слетать, воспользовавшись свойством полосы детализации **ForceNewPage** — форсировать переход на новую страницу. В полосе внешнего цикла установите это свойство в **true.** Тогда по окончании каждого внешнего цикла будет осуществляться переход на новую страницу.

Ваш отчет готов. Теперь необходимо связать его оператором в сроке меню (компонента MainMenu):

```
Form4->QuickRep1->Preview();
```