Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

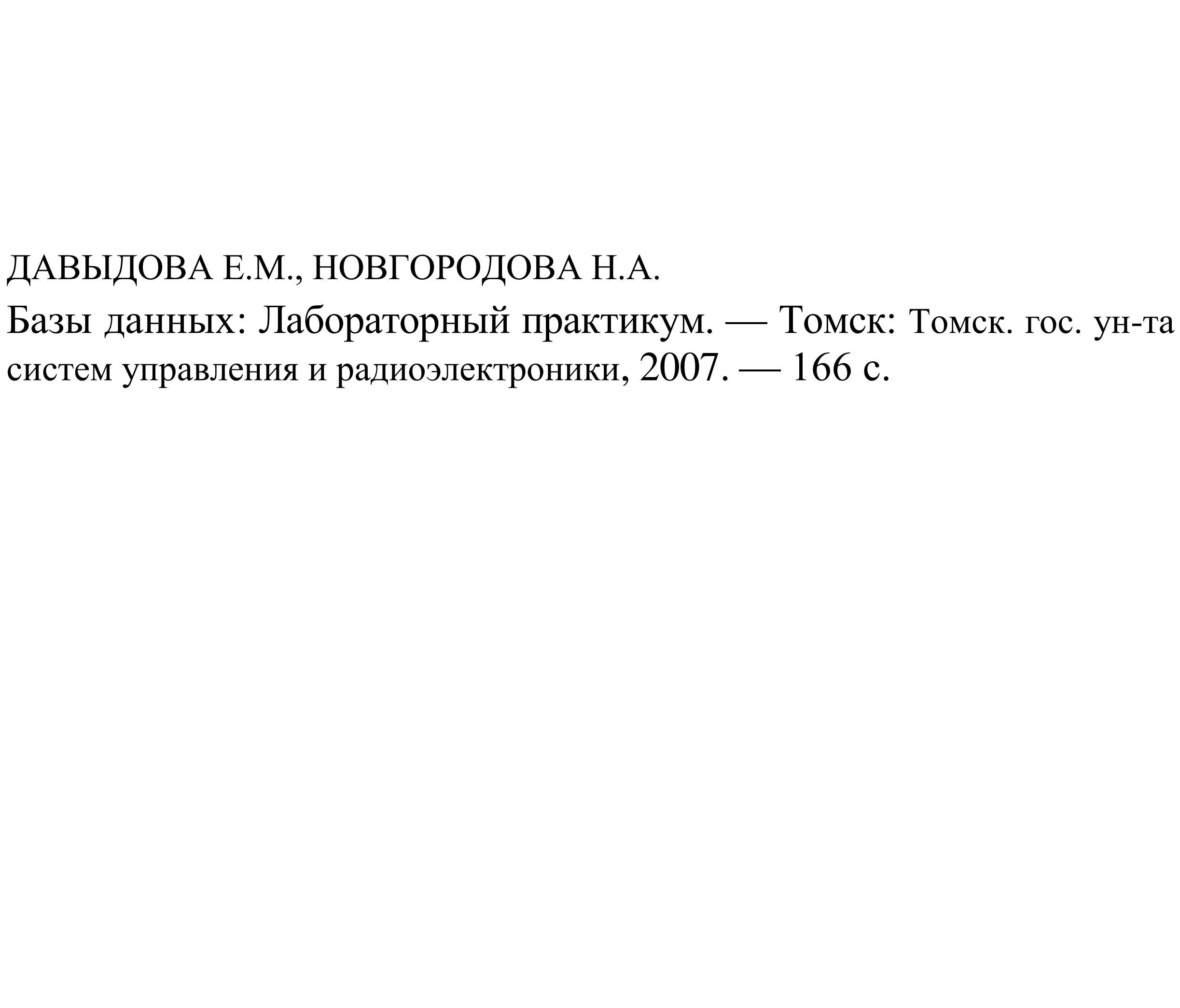
Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

Е.М. ДАВЫДОВА, Н.А. НОВГОРОДОВА

БАЗЫ ДАННЫХ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

2007



ДАВЫДОВА Е.М., НОВГОРОДОВА Н.А.
Базы данных: Лабораторный практикум. — Томск: Томск. гос. ун-та систем управления и радиоэлектроники, 2007. — 166 с.

© ДАВЫДОВА Е.М., НОВГОРОДОВА Н.А., 2007
© Томск, 2007

СОДЕРЖАНИЕ

Лабораторная работа №1. СОЗДАНИЕ ПРОСТОГО ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ В DELPHI 5
Задание 5
Краткая теория 5
Связь компонентов для работы с БД 6
Примеры невизуальных компонентов для работы с БД 8
Примеры визуальных компонентов для работы с БД 10
Практическая работа 16
Лабораторная работа №2. СОЗДАНИЕ БД ПОД УПРАВЛЕНИЕМ СУБД PARADOX 17
Создание таблиц локальных БД. СУБД Paradox. Типы данных Paradox. Работа с утилитами BDE Administrator, Database Desktop 17
Создание таблиц локальных БД через утилиту Database Desktop (DBD) 19
Изменение структуры существующей таблицы 22
Работа с записями в таблице в DBD 22
Определение индексов 22
Определение ссылочной целостности между таблицами БД (СУБД Paradox) в DBD 24
Первичные и вторичные ключи 25
Реляционные отношения между таблицами 26
Создание БД 28
Создание простого приложения для работы со связанными таблицами 34
Реализация связи Master-Detail между наборами данных 34
Лабораторная работа №3. РАБОТА С ПОЛЯМИ — КОМПОНЕНТ TFIELD 36
Применение TField 36
Использование редактора полей для организации компонентов TField 38
Типы полей 39
Обращения к полям и их значениям 42

Особенности показа полей в визуальных компонентах во время выполнения ... 48
Создание вычисляемых полей ... 51
Создание полей выбора данных (lookup-полей) ... 53
Требования к выполнению лабораторной работы №3 ... 57
Лабораторная работа №4. ОБЩИЕ ПРИНЦИПЫ РАБОТЫ С НАБОРАМИ ДАННЫХ ... 58
Понятие наборов данных ... 58
Состояния наборов данных ... 59
Краткое описание ... 60
Навигация по набору данных ... 63
Определение начала и конца набора данных ... 64
Спонтанные перемещения по набору данных ... 66
Реакция на изменение курсора набора данных ... 66
Временное отключение визуализации при работе с НД ... 66
Внесение изменений в НД ... 67
Практическая работа ... 74
Особенности работы с мемо-полем и полем с графическим изображением ... 74
Лабораторная работа №5. РАБОТА С SQL ... 77
Задание ... 77
Компонент TQuery ... 77
SQL оператор запросов SELECT ... 79
Практика ... 105
Лабораторная работа №6. ПОИСК, ФИЛЬТРАЦИЯ И СОРТИРОВКА ЗАПИСЕЙ В НД ... 110
Задание ... 110
Обзор методов поиска записей в НД ... 110
Фильтрация записей в НД ... 115
Сортировка записей в НД ... 119
Использование методов поиска, фильтрации и сортировки данных в приложении ... 120
Лабораторная работа №7. ПОСТРОЕНИЕ ОТЧЕТНЫХ ФОРМ ... 127
Задание ... 127
Компоненты для построения отчетов ... 127

Разработка варианта отчетной формы по информации из БД отдела кадров 139

Лабораторная работа №8. ПОСТРОЕНИЕ ГРАФИКОВ ПО ДАННЫМ БД 145

Задание 145

Создание графика. Компонент TDBChart 145

Разработка формы с реализацией построения графиков для БД отдела кадров 162

СПИСОК ЛИТЕРАТУРЫ 165

ЛАБОРАТОРНАЯ РАБОТА № 1
СОЗДАНИЕ ПРОСТОГО ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ В DELPHI

Задание

1. Изучение визуальных компонентов и их свойств.
2. Создание простого приложения для работы с таблицей «biolife».

Краткая теория

В Delphi для доступа к базам данных из приложения, как известно, используется цепочка «Приложение -> BDE -> База данных». Это означает, что при любом обращении к БД из приложения реально адресуется BDE(Borland Database Engine). BDE, используя собственные функции, связывается непосредственно с базой данных.

Для работы с конкретной базой данных BDE, во-первых, должна знать:

– где БД физически расположена;
– параметры этой БД;
– общие параметры драйвера БД того типа, к которому принадлежит обрабатываемая БД;
– общие системные установки.

Машина баз данных фирмы Borland(BDE), представляет собой библиотеку функций специализированного программного интерфейса (API) и набор драйверов для работы с множеством «популярных» СУБД. Она должна устанавливаться на каждом компьютере, который использует приложения для работы с БД, написанные на Delphi с использованием компонент, работающих с ней. BDE может использоваться независимо от Delphi (например, прямой вызов API функций из любых других программ).

Начиная с Delphi 5, включена специализированная библиотека для прямого доступа к СУБД InterBase (закладка InterBase). В этом случае можно не инсталлировать BDE на компьютере, где будет запускаться конечное приложение. Также существует множество компонент для Delphi третьих производителей для доступа к другим «популярным» СУБД (Oracle, dBase, MS SQL и др.),

которые осуществляют прямой доступ к «своей» СУБД и также не требуют инсталляции BDE.

Приложение состоит из невизуальных и визуальных компонентов работы с БД, компонентов для выдачи отчетов (которые представляют собой разновидность визуальных компонентов), а также модулей данных.

Невизуальные компоненты имеют прямой выход, например, на BDE, которая, в свою очередь, контактирует с БД. К этим компонентам относятся, например, компоненты со страниц Data Access, InterBase, BDE, dbExpress (начиная с Delphi 6) и др.

Визуальные компоненты служат для представления данных из невизуальных компонентов, т.е. служат целям обеспечения интерфейса пользователя при работе с данными. К визуальным компонентам относятся, например, компоненты со страниц Data Controls, QReport и др.

Модули данных служат для централизованного хранения отдельных экземпляров невизуальных компонентов с целью придания тем или иным наборам данных единообразного поведения во всем приложении.

Приложение состоит из одной или нескольких форм. Каждая форма может:

– хранить и использовать свои «собственные» невизуальные компоненты;

– использовать невизуальные компоненты, хранящиеся в одном или нескольких модулях данных;

– использовать невизуальные компоненты, хранящиеся и используемые в других формах.

Каждая форма может воспользоваться только «собственными» визуальными компонентами, поскольку визуальные компоненты выполняют интерфейсный функции и при деактивации формы теряют свою видимость на экране.

Связь компонентов для работы с БД

Визуальные и невизуальные компоненты связываются друг с другом при помощи свойств. Значения этих свойств обычно устанавливаются на этапе разработке приложения, но могут устанавливаться и во время выполнения приложения.

Примеры невизуальных и визуальных компонент для работы с БД приведены в таблице 1 и таблице 2 соответсвенно.

Для связи визуальных и невизуальных компонентов для работы с БД используется компонент «промежуточного уровня» TDataSource (источник данных), который находится на странице Data Access. Его свойство DataSet содержит имя соответствующего компонента типа «набор данных».

Понятие набора данных несколько шире, чем понятие таблицы БД, так как набор данных может содержать:

– подмножество записей или полей таблицы БД;

– записи, сформированные из нескольких таблиц БД;

– записи, значения которых формируются при помощи вычислений (вычисляемые).

Визуальные компоненты разделяются на:

– работающие с множеством записей НД, например TDBGrid, TDBNavigator и др.;

– работающие с отдельным полем НД, например TDBEdit, TDBMemo, TDBText и др.

Например:

1) для компонента TTable:

- свойство DatabaseName — тип String; имя Алиаса BDE или строка, в которой указан путь к каталогу на диске, где находится БД;
- свойство TableName — тип String; имя выбранной таблицы БД;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TTable;
- свойство Active — тип Boolean; указывает активен ли (открыт, готов для доступа к данным) соответствующий НД (значение True) или не активен (значение False).

2) для компонента TQuery:

- свойство DatabaseName — тип String; имя Алиаса BDE или строка, в которой указан путь к каталогу на диске, где находится БД;
- свойство SQL — тип String; SQL оператор Select;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TQuery;

- свойство Active — тип Boolean; указывает активен ли (открыт, готов для доступа к данным) соответствующий НД (значение True) или не активен (значение False).
- для компонента TDataSource:
- свойство DataSet — содержит имя соответствующего компонента типа «набор данных»;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDataSource.

3) для компонента TDBGrid:

- свойство DataSource — имя необходимого компонента TDataSource;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDBGrid.
- для компонента TDBEdit:
- свойство DataSource — имя необходимого компонента TDataSource;
- свойство DataField — тип String, имя небходимого поля соответсвующего НД.
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDBEdit.

Примеры невизуальных компонентов для работы с БД

К этим компонентам относятся, например, компоненты со страниц Data Access, InterBase, BDE, dbExpress (начиная с Delphi 6) и др. Примеры невизуальных компонентов для работы с БД приведены в таблице 1. Остальные будут рассмотрены в ходе дальнейшего выполнения лабораторных работ.

Таблица 1 — **Примеры невизуальных компонент для работы с БД**

Компонент	Назначение
TTable	Реализует набор данных (НД), источником данных для которого, является одна таблица БД. Содержит множество методов, свойств и событий, многие из которых расширяют множество методов, свойств и событий, определенных в предках TTable — TDataSet и TBDEDataSet.
TQuery	Реализует набор данных, источником данных для кото-

Продолжение табл. 1

Компонент	Назначение
	рого является одна или несколько таблиц БД. Структура записи НД, состав НД определяются SQL запросом (оператор Select). Кроме выдачи НД, используется для групповых операций обновления, добавления или удаления в таблицах БД, а также может выполнять другие действия, предусмотренные реализацией языка SQL для той СУБД, с которой работает TQuery. Для типов данных персональных СУБД позволяет реализовывать «локальный» вариант SQL. При помощи TQuery можно реализовывать как статические, так и динамические (изменяющиеся в процессе выполнения приложения) SQL запросы. Содержит множество методов, свойств и событий, многие из которых расширяют множество методов, свойств и событий, определенных в предках TQuery — TDataSet, TBDEDataSet и TDBDataSet.
TDataSet TBDEDataSet TDBDataSet	Являются предками активно используемых в приложении компонентов типа «набор данных» (например, TTable TQuery). TDataSet определяет свойства, методы и события для работы с БД, независимые от машины БД. Многие из них являются абстрактными или виртуальными. TBDEDataSet, наоборот, определяет ряд свойств, методов и событий, зависящих от используемой машины БД. TDBDataSet дополнительно вводит ряд свойств.
TDataSource	Служит промежуточным звеном в цепочке «набор данных — TDataSource — визуальные компоненты для работы с данными». Позволяет устанавливать некоторые параметры НД, устанавливать состояние НД, отслеживать изменения в НД.
TField	Реализует поле НД. Помимо полей, физически определенных в таблице БД и включенных в состав конкретного НД, компонент TField создается для каждого вычисляемого поля или поля, возвращающего значение из другого НД (Lookup или поля подстановочного типа), а также для результатов вычисления выражений и агрегатных функций в SQL запросах. Предоставляет набор методов, свойств и событий, посредством которых можно управлять поведением поля. TField есть родительский класс для дочерних

Окончание табл. 1

Компонент	Назначение
	компонентов, реализующих поля конкретных типов (например, TstringField, TIntegerField и т.д.). TField определяет свойства, методы, события, которые по праву наследования доступны и во всех дочерних классах полей.

Примеры визуальных компонентов для работы с БД

К визуальным относятся компоненты со страниц: Data Controls, QReport и др.

Примеры визуальных компонентов для работы с БД приведены в таблице 2. Остальные будут рассмотрены в ходе дальнейшего выполнения лабораторных работ.

Таблица 2 — **Примеры визуальных компонент для работы с БД**

Компонент	Назначение
TDBText	Показывает «только для чтения» значение поля текущей записи набора данных.
TDBEdit	Обеспечивает просмотр и изменение значения поля текущей записи набора данных. Поля — любого типа, кроме полей комментариев и BLOP.
TDBMemo	Позволяет просматривать и корректировать значения Memo-поля (поля комментария) в режиме текстового редактора.
TDBImage	Позволяет просматривать графические поля, заносить в них содержимое.
TDBGrid	Показывает содержимое полей набора данных в «табличном виде», когда записям соответсвуют строки, а полям — столбцы.
TDBCtrlGrid	Версия компонента TDBGrid, позволяющая показывать содержимое одной записи набора данных в нескольких строках.
TDBNavigator	Позволяет осуществлять навигацию по записям набора данных, переводить набор данных в состояние вставки, изменения, удаления, запоминания изменений.
TDBCheckBox	Обеспечивает просмотр и изменение значения поля типа Boolean текущей записи набора данных.
TDBListBox	Применяется, когда необходимо выбрать значение

Окончание табл. 2

Компонент	Назначение
	поля из предустановленного списка значений; значения показываются в виде строк в списке фиксированного размера. Содержимое списка определяется свойством Items.
TDBComboBox	Применяется для тех же целей, что и TDBListBox, но список выпадающий.
TDBRadioGroup	Обеспечивает возможность выбора значения для поля, которое может содержать фиксированное число вариантов значений. Значения показываются в виде радиокнопок.
TDBRichEdit	Имеет то же значение, что и TDBMemo, но позволяет работать с текстом формата RTF, включающим разные шрифты, графику и пр.
TQuickRep	Компонент TQuickRep и связанная с ним группа компонентов позволяет разрабатывать формы отчетов.
TDBChart	Используется для построения графиков.

Остальные будут рассмотрены в ходе дальнейшего выполнения лабораторных работ. Рассмотрим более подробно использование визуальных компонентов.

Компонент TDBText:

Применяется для показа значения текстового поля текущей записи набора данных. Изменять значение, показываемое при помощи TDBText нельзя.

Для использования TDBText необходимо:

- свойство DataSource — имя необходимого компонента TDataSource, связанного с НД;
- свойство DataField — тип String, имя небходимого поля соответсвующего НД;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDBText.

Компонент TDBEdit:

Обеспечивает просмотр и изменение значения поля текущей записи набора данных. Поля — любого типа, кроме полей комментариев и BLOP.

Для использования TDBEdit необходимо:

- свойство DataSource — имя необходимого компонента TDataSource, связанного с НД;
- свойство DataField — тип String, имя небходимого поля соответсвующего НД;
- свойство ReadOnly:Boolean — если содержит True, то значение поля доступно только для чтения, если False — значение поля можно изменять;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDBEdit.

При вводе значения в TDBEdit приложение автоматически отслеживает, чтобы введенное значение было совместимо по формату с полем набора данных. Ввод неверных данных блокируется путем генерации исключений.

Событие OnChange: TNotifyEvent наступает при изменении значения поля.

Событие OnEnter: TNotifyEvent наступает при получении и утрате фокуса управления компонентом TDBEdit.

Компонент TDBNavigator:

Позволяет осуществлять навигацию по записям набора данных, переводить набор данных в состояние вставки, изменения, удаления, запоминания изменений.

Компонент TDBNavigator является набором кнопок, с помощью которых пользователь может перемещаться между записями набора данных, выполнять операции вставки и удаления записей, вносить сделанные изменения в базу данных или отменять их. Полный вид TDBNavigator приведен на рис. 1.

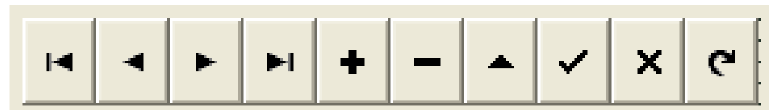


Рис. 1 — Полный вид компонента TDBNavigator

В таблице 3 приведена расшифровка значений кнопок навигатора.

Таблица 3 — **Расшифровка значений кнопок навигатора**

Кнопка	Значение
⏮	Переместиться в начало НД
◀	Перейти к предыдущей записи НД
▶	Перейти к следующей записи НД
⏭	Переместиться в конец НД
+	Добавить новую запись
−	Удалить существующую запись
▲	Изменить существующую запись
✓	Сохранить новую или измененную запись
✕	Отменить ввод или изменение записи
↻	Обновить содержание НД

Для использования TDBNavigator необходимо:

- свойство DataSource — имя необходимого компонента TDataSource, связанного с НД;
- свойство VisibleButtons — определяет, какие кнопки будут отображены в навигаторе.

Компонент TDBCheckBox:

Позволяет «отметить» и «снять отметку» с логического поля в составе текущей записи НД.

Для использования TDBCheckBox необходимо:

- свойство DataSource — имя необходимого компонента TDataSource, связанного с НД;
- свойство DataField — тип String, имя небходимого поля соответсвующего НД;
- свойство ReadOnly: Boolean — если содержит True, то значение поля доступно только для чтения, если False — значение поля можно изменять;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDBCheckBox;
- свойство Checked: Boolean — позволяет определить отмечено ли поле, на которое ссылается TDBCheckBox (значение True) или не отмечено (False);

- свойство State: TCheckBoxState — возвращает состояние поля. Возможные значения: cbChecked — поле отмечено, cbUnchecked — поле не отмечено, cbGrayed промежуточное состояние, когда поле не отмечено, но в нем показывается серый символ отметки. Он означает, что поле не содержит ни True, ни False, а содержит пустое значение. Это состояние присуще при добавлении записей.

Компонент TDBCheckBox можно связывать не только с логическим, а с символьным полем. В этом случае необходимо:

- свойство ValueChecked: String — устанавливает значения поля, при которых TDBCheckBox переходит в состояние cbChecked. При наличии нескольких значений они разделяются точкой с запятой: DBCheckBox1.ValueChecked:='Да;Yes;True';
- свойство ValueUnchecked: String — устанавливает значения поля, при которых TDBCheckBox переходит в состояние cbUnchecked. При наличии нескольких значений они разделяются точкой с запятой: DBCheckBox1.ValueUnchecked:='Нет;No;False'.

Событие OnClick: TNotifyEvent наступает, если на TDBCheckBox щелкнуть один раз левой кнопкой мыши.

Событие OnEnter: TNotifyEvent наступает при получении и утрате фокуса управления компонентом TDBCheckBox.

Компонент TDBRadioGroup:

Служит для предоставления фиксированного набора возможных значений поля при помощи группы зависимых переключателей.

Для использования TDBRadioGroup необходимо:

- свойство DataSource — имя необходимого компонента TDataSource, связанного с НД;
- свойство DataField — тип String, имя небходимого поля соответсвующего НД;
- свойство ReadOnly:Boolean — если содержит True, то значение поля доступно только для чтения, если False — значение поля можно изменять;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDBRadioGroup;

- свойство Items: TStrings — определяет число и названия вариантов возможных значений поля, содержащихся в TDBRadioGroup;
- свойство TDBRadioGroup.ItemIndex: Integer — позволяет определить индекс текущего выбора. Возвращает номер выбранного значения в порядке, в котором они определены в TDBRadioGroup.Items. Отсчет ведется от нуля;
- свойство Columns: Integer — указывает сколько назначено столбцов для вывода переключателей.

Событие OnClick: TNotifyEvent наступает, если на TDBRadioGroup щелкнуть мышью.

Событие OnEnter: TNotifyEvent наступает при получении и утрате фокуса управления компонентом TDBRadioGroup.

Компонент TDBListBox:

Применяется, когда нужно выбрать значения поля из предустановленного списка значений. Возможные значения содержатся в качестве строк компонента TDBListBox.

Для использования TDBListBox необходимо:

- свойство DataSource — имя необходимого компонента TDataSource, связанного с НД;
- свойство DataField — тип String, имя небходимого поля соответсвующего НД;
- свойство ReadOnly:Boolean — если содержит True, то значение поля доступно только для чтения, если False — значение поля можно изменять;
- свойство Name — логическое имя компонента, по которому можно обратиться к конкретному компоненту TDBListBox;
- свойство Items: TStrings — содержит список возможных значений поля, содержащихся в TDBListBox.

Событие OnClick: TNotifyEvent наступает, если на TDBListBox щелкнуть мышью.

Событие OnDblClick: TNotifyEvent наступает, если на TDBListBox щелкнуть двойным нажатием мыши.

Событие OnEnter: TNotifyEvent наступает при получении и утрате фокуса управления компонентом TDBListBox.

Компонент TDBComboBox:

Этот компонент применяется для тех же целей, что и TDBListBox, но список выпадающий.

Практическая работа

Задание 1. Создание простого приложения, предназначенного для работы со справочником по биологическим рыбам.

1. Запустить Delphi.
2. На закладке BDE выбрать компонент TTable (с Delphi6). Для версий ниже Delphi6 выбрать компонент TTable с закладки Data Access.
3. DatabaseName псевдоним DBDEMOS при помощи выпадающего списка или вручную.
4. TableName в biolife.db.
5. Расположить на форме компонент TDataSource из Data Access.
6. Свойство DataSet в Table1.
7. Расположить на форме компонент TDBGrid из палитры DataControls.
8. Вставить компонент TDBNavigator.
9. Связать компоненты TDBGrid и TDBNavigator с TDataSource.
10. Свойства компонента DataSource значение DataSourse1.
11. Сохранить форму и запустить программу.
12. Для добавления записей в TDBGrid клавиша-Ins или курсор вниз, отказ-Esc, удаление Ctrl+Delete. Либо использовать функции TDBNavigator.
13. Запустить приложение.

Задание 2. Изучение визуальных компонентов.

Изучите основные свойства визуальных компонентов. Попробуйте применить их для работы с таблицей «Biolife».

ЛАБОРАТОРНАЯ РАБОТА № 2
СОЗДАНИЕ БД ПОД УПРАВЛЕНИЕМ СУБД PARADOX

Создание таблиц локальных БД. СУБД Paradox. Типы данных Paradox. Работа с утилитами BDE Administrator, Database Desktop

При работе с таблицами локальных БД (в число которых входит и таблицы СУБД Paradox) БД считается каталог на диске, в котором хранятся файлы таблиц БД, индексов, примечаний (мемо-полей) и т.д.

Для хранения одной таблицы создается отдельный файл. Такие же отдельные файлы создаются для хранения индексов таблицы и мемо-полей.

Создайте отдельный каталог диске С для объектов БД. Например, на диске C:\Proba.

Создание псевдонима БД

Обращение к БД из утилит и программ может осуществляться по псевдониму (алиасу, Alias) БД. Параметры БД и ее местоположение определяются псевдонимом БД. Псевдоним (имя) — должен быть зарегистрирован в файле конфигурации конкретного компьютера при помощи утилиты BDE Administrator.

BDE, используя собственные функции, связывается непосредственно с базой данных. Действия, осуществляемые при этом BDE, мы здесь обсуждать не будем, поскольку эта тема отдельного рассмотрения.

Для работы с конкретной базой данных BDE должна знать:

– где БД физически расположена;
– параметры этой БД;
– общие параметры драйвера БД того типа, к которому принадлежит обрабатываемая БД;
– общие системные установки.

Параметры драйвера БД определяют параметры конкретной БД, значения которых не указаны.

Системные установки являются общими для всех драйверов.

Псевдоним должен быть зарегистрирован в файле конфигурации конкретного компьютера при помощи утилиты BDE Administrator.

Присвоим псевдоним PROBA создаваемой БД. BDE считывает параметры, поставленные в соответствие данному псевдониму, что во многом определяет ее дальнейшие действия по физической работе с БД. Порядок работы:

1. Запустите утилиту BDE Administrator:
(Пуск-Программы-Delphi6- BDE Administrator).

2. Выбирете в появившемся окне имя драйвера базы данных (STANDARD для Paradox и dBase, MSACCESS для Microsoft Access, ORACLE, INTRBASE, SYSBASE, MSSQL, INFORMIX, DB2 соответственно для баз данных Oracle, InterBase, Sysbase, MS SQL Server, Informix, DB2 и, если установлен, драйвер ODBC).

Выбирете в главном меню окна утилиты элемент Object | New. Оставьте в появившемся окне тип создаваемой БД без изменений (STANDART), <OK>.

3. В левом поле окна администратора БД вы увидите строку с именем STANDART1. Измените это имя на PROBA (рис. 2).

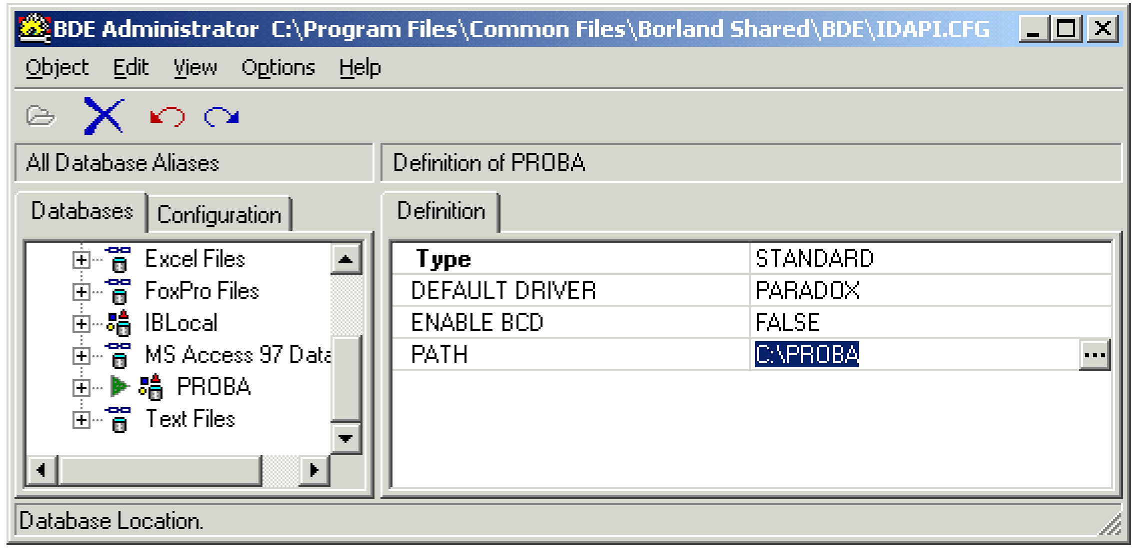


Рис. 2

4. В правом поле указаны параметры БД. Измините параметр PATH, который указывает маршрут доступа к каталогу, в котором располагается БД. Можно ввести путь вручную, но луч-

ше воспользоваться средствами администратора. Для этого нужно щелкнуть по полю PATH и нажать на появившуюся в правом углу поля кнопку… Затем выбрать каталог c:\PROBA, <OK>.

5. Запомните определение псевдонима: В левом окне Администратора щелкнуть по имени псевдонима правой кнопкой мыши и выберете опцию Apply, и <OK> в появившемся диалоговом окне.

6. Закройте окно утилиты BDE Administrator.

Создание псевдонима завершено и к нему можно обращаться из других утилит и приложений. Однако каталог, на который ссылается псевдоним БД, ещё пуст. Необходимо создать в нём таблицы БД

Создание таблиц локальных БД через утилиту Database Desktop (DBD)

Для создания таблицы БД можно использовать утилиту Database Desktop (DBD), предназначеннную для создания локальных БД.

Создать таблицу БД можно:

– File|New|Table после чего в появившемся окне необходимо указать тип таблиц выбранной СУБД (выберете СУБД Paradox 7 и нажмите OK). Затем появится окно определения структуры таблицы БД где необходимо указать:

1) название поля Field Name. Не рекомендуется использование названия полей на русском языке, пробелов в названиях полей. Желательно называть поля в соответствии с семантикой выбранной предметной области или название полей в соответствии с именами полей логической модели данных;

2) тип данных Type значений хранимых в данном поле. Для того чтобы определить тип поля щелкните по столбу Type либо при помощи правой кнопки мыши либо нажмите клавишу пробела;

3) размер (если необходимо указать для выбранного типа данных значений в поле) Size;

4) указать Key (содержит *) для полей, входящих в состав в первичного ключа. Для этого нажмите любой символ на клавиатуре. Если в первичный ключ входит несколько полей, то они должны определяться в той последовательности, в которой они

присутствуют в первичном ключе. Ключевые поля должны быть описаны в первую очередь.

В строке статуса в нижней части левого окна Вам выдается подсказка о Ваших возможных действиях.

Если Вы допустили ошибку при определении структуры таблицы, то СУБД выдаст сообщение об ошибке в строке состояний.

Для того чтобы определить требование обязательного заполнения поля значением, то необходимо включить переключатель Required Field. Другие строки ввода ниже переключателя служат для наложения ограничений на значение поля:

1) Minimum value — определяет минимальное значение поля.
2) Maximum value — определяет максимальное значение поля.
3) Default value — определяет значение поля по умолчанию.
4) Picture — определяет шаблон изображения поля. Для формирования шаблона следует нажать кнопку Assist.

Отсутствие значения в одной из строк ввода означает отсутствие ограничений на значение поля.

Для того чтобы запомнить таблицу на диске необходимо нажать кнопку Save As. Затем необходимо указать имя таблицы. При желании можно указать каталог или псевдоним, отличных от принятых по умолчанию. По умолчанию принимается рабочий каталог или каталог, определяемый рабочим псевдонимом.

Для установки рабочего псевдонима нужно выбрать элемент главного меню File|Working Directory где указать каталог БД или имя созданного псевдонима.

Типы данных СУБД Paradox

Тип поля	Обозначение	Хранимые значения
Alpha	**A**	Символьные значения до 255 символов
Number	**N**	Числовые значения с плавающей точкой в диапазоне $-10^{307}..+10^{308}$. Точность до 15 значащих цифр.
Money	**$**	Аналогичен типу Number, но предназначен для хранения денежных сумм. Число знаков после запятой по умолчанию –2. При показе значения выводится знак денежной единицы.
Short	**S**	Целочисленные значения в диапазоне

Тип поля	Обозначение	Хранимые значения
		–32767…+32767.
LongInteger	**I**	Целочисленные значения в диапазоне –2147483648…+2147483647.
BCD	**#**	Числовые значения, в том числе и дробные, в двоично-десятичном формате. Применяется в вычислениях, где важна точность (финансовые учреждения и др.). Для проведения вычислений больше времени.
Date	**D**	Значения даты (в диапазоне от 01.01.9999 до н.э. до 31.12.9999).
Time	**T**	Значение времени.
Timestamp	**@**	Значение даты и времени.
Memo	**M**	Строковые значения длиной более 255 символов. Максимальная длина не ограничена. От 1 до 240 символов могут храниться вместе с таблицей БД, остальные хранятся в виде Memo файла (имеет расширение .mb).
Formatted Memo	**F**	Аналогично мемо-полю, но может хранить форматированные тексты, в которых фрагменты текста представлены разным шрифтом, цветом, стилями.
Graphic Fields	**G**	Графические изображения в формате файлов .bmp, .pcx, .tif, .gif, .esp, которые при хранении преобразуются к формату .bmp. Хранятся отдельно от основной таблицы.
OLE	**O**	Информация в форматах, поддерживаемых технологией OLE (Object Linking and Embedding) фирмы Microsoft
Logical	**L**	Логические значения («True», «False»). Значения регистров не имеет значения.
Autoicrement	**±**	Автоинкрементное поле. Значения допустимы только для чтения. Обычно ключевое поле в составе первичного ключа. При добавлении новой записи значение поля вычисляется автоматически таким образом, чтобы в одной таблице не было одинаковых значений. Значение поля из удаленных записей не используется.
Binary	**B**	Произвольные двоичные значения. Должны интерпретироваться приложениями

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

Изменение структуры существующей таблицы

Если в структуру существующей таблицы необходимо внести изменения, то следует выбрать File|Open|Table, затем в появившемся диалоге указать имя таблицы и нажать ОК. Чтобы изменить структуру таблицы выберите элемент меню Table|Restructure или выбрать соответствующую кнопку на панели инструментов.

Работа с записями в таблице в DBD

Если Вы хотите вносить записи или изменять значения в некоторых записях или провести удаление в DBD, то следует нажать F9 или выбрать Table|Edit Data или выбрать соответствующую кнопку на панели инструментов. Для работы с записями можно выбрать пункт меню Record, где, например:

1) Next (F12) — перейти к следующей записи.
2) Previous (F11) — перейти к предыдущей записи.
3) Next Set (Shift+F12) — следующий набор данных.
4) Previous Set (Shift+F11) — предыдущий набор данных.
5) First (Ctrl+F11) — перейти к первой записи.
6) Last (Ctrl+F12) — перейти к последней записи.
7) Insert (Ins) — добавить новую запись перед текущей.
8) Delete (Ctrl+Del) — удалить выбранную запись.

Определение индексов

Индексы представляю собой механизмы быстрого доступа к данным в таблицах БД. Сущность индексов состоит в том, что они хранят значения индексных полей (т.е. полей по которым построен индекс) и указатель на запись в таблице.

Существуют следующие индексы:

– первичный индекс — таблица может иметь только один первичный индекс, его значение должно быть уникальным;

– вторичный уникальный индекс — таблица может иметь несколько таких индексов, значение индекса должно быть уникальным. Этот индекс используется для упорядочения записей;

– простой вторичный индекс — таблица может иметь несколько таких индексов, их значения не должны быть уникальными.

Ключи используются для связывания данных. Существуют первичные и вторичные ключи: первичный ключ родительской таблицы связывается с вторичным ключом дочерней таблицы.

Для большинства СУБД ключ может быть установлен только на индексированное поле. Поэтому понятие индекса и ключа часто путают. Следует помнить, что, говоря о связи таблиц, мы всегда имеем в виду ключ, а говоря об упорядочении записей или их уникальности, — индекс.

Поля первичного ключа составляют первичный индекс. Индексы, создаваемые пользователем дополнительно называются вторичными.

Для того чтобы создать вторичный индекс в правом верхнем углу окна создания структуры таблицы в списке Table Properties необходимо выбрать элемент Secondary Indexes и нажать кнопку Define. В появившемся диалоговом окне в поле Fields содержится список полей выбранной нами таблицы. Поле Index Fields предназначено для хранения полей, входящих в создаваемый индекс. Чтобы скопировать конкретное поле из списка Fields в список Index Fields необходимо нажать кнопку с изображением правой стрелки (убрать при помощи левой стрелки). Последовательность добавления полей в список важна, так как она определяет порядок чередования полей в списке. После того, как определены нужные поля/поле в список Index Fields нажмите OK. В появившемся окне запрашивается имя индекса. Следует ввести имя индекса и нажать OK. Не рекомендуется составлять название индекса только из имен полей, поскольку такой способ именования индексов используется автоматически при создании ссылочной целостности между таблицами.

В последствии, щелкнув по имени индекса, его можно удалить (кнопка Erase) или изменить (кнопка Modify).

После определения индексов таблицу необходимо сохранить (кнопка Save).

Определение ссылочной целостности между таблицами БД (СУБД Paradox) в DBD

Между двумя и более таблицами БД могут существовать отошения подчиненности (связи). Отношения подчиненности (связи) определяют, что для каждой записи главной таблицы (master, или родительской Parent) может существовать одна или несколько записей в подчиненной таблице (detail, или дочерней Child).

Под ссылочной целостностью имеет в виду совокупность связей между отдельными таблицами во всей БД. Нарушение хотя бы одной такой связи делает информацию в БД недостоверной.

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

Механизм каскадных воздействий состоит в обеспечении следующих требований:

1) необходимо запретить изменения поля связи в записи дочерней таблицы без синхронного изменения полей связи в дочерней и родительской таблицах. Обычно инициатива изменения поля связи реализуется в записи родительской таблицы;

2) при изменении поля связи в родительской таблице, следует синхронно изменить значения полей связи в соответствующих записях дочерней таблицы;

3) при удалении записи в родительской таблице, следует удалить соответсвующие записи в дочерней таблице.

Для обеспечения ссылочной целостности в дочерней таблице создается внешний ключ. Во внешний ключ входят поля связи дочерней таблицы. Для связей типа «один-ко-многим» внешний ключ по составу должен совпадать с первичным ключом родительской таблицы или (реже) с частью первичного ключа (в этом случае следует признать, что нормализация таблиц БД произведена не полностью). По полям внешнего ключа создается вторичный индекс.

Для определения ссылочной целостности необходимо открыть дочернюю таблицу в режиме реструктурирования таблицы. Затем в правом верхнем углу окна создания структуры таблицы в списке Table Properties необходимо выбрать элемент Referential Integrity и нажать кнопку Define. В появившемся диалоговом ок-

не в поле Fields содержится список полей выбранной нами дочерней таблицы, а в списке Tables — таблицы созданной БД. В списке Fields необходимо выбрать поле внешнего ключа дочерней таблицы и при помощи кнопки с избражением стрелки вправо записать его в список Child fields, который содержит поля внешнего ключа дочерней таблицы. В списке Tables необходимо выбрать сответсвующую родительскую таблицу и нажать кнопку с изображением стрелки влево. В поле Parent's key (ключ родительской таблицы) будут показаны поля из первичного ключа родительской таблицы.

Переключатели Update rules определяют вид каскадных воздейсвий на родительскую таблицу при изменении значения поля связи в дочерней таблице или при удалении записи в дочерней таблице:

1) Cascade — каскадные изменения и удаления подчиненных записей в родительской таблице.

2) Prohibit — запрет на изменение поля связи или удаление записи в дочерней таблице, если для данной записи есть связанные записи в родительской таблице.

В Paradox ссылочные целостности именуются. Введите имя и нажмите ОК. После определения ссылочной целостности сохраните изменения в таблице (кнопка Save).

Используя SQL операторы:

для этого выберете File|New|SQL file после чего в появившемся окне редактора SQL необходимо указать необходимые SQL операторы создания таблиц, индексов, доменов и др.

Первичные и вторичные ключи

В каждой таблице БД может существовать **первичный ключ (PK)** — поле или набор полей, однозначно определяющий запись. Значение первичного ключа в таблице БД должно быть уникальным, т.е. в таблице не должно существовать двух и более записей с одинаковым значением первичного ключа. Поскольку первичный ключ должен быть уникальным, для него могут использоваться не все поля таблицы. Если в таблице нет полей, значения которых уникальны, для создания первичного ключа в неё обычно вводят дополнительное числовое поле.

Вторичные ключи — устанавливаются по полям, которые часто используются при поиске или сортировке данных: постро-

енные по вторичным ключам индексы помогут системе значительно быстрее найти нужные значения, хранящиеся в соответствующих полях. В отличие от первичных ключей, поля для вторичных ключей могут содержать неуникальные значения.

Внешний ключ (FK) — поле или набор полей, указывающий на запись в другой таблице (родительской), связанную с данной записью (в дочерней таблице). Внешний ключ по составу полей должен совпадать с первичным ключом родительской таблицы. Внешний ключ является вторичным ключом.

Реляционные отношения между таблицами

Отношение один-ко-многим

Одной записи родительской таблицы может соответствовать несколько записей в дочерней таблице.

Различают две разновидности связи один-ко-многим:

– всякой записи в родительской таблице должна соответствовать запись в дочерней таблице (жёсткая связь);

– некоторые записи в родительской таблице могут не иметь связанных с ней записей в дочерней таблице.

Связь один-ко-многим является самой распространённой для реляционных баз данных, позволяет моделировать иерархические структуры данных.

Рассмотрим пример отношения один-ко-многим:

Товар	Ед.измерения	Цена ед.
Сахар	кг.	5000
Макароны	кг.	7000
Фанта 6000	бут. 1л.	

Товар	Дата	Кол-во
Сахар	10.01.03	100
Сахар	12.01.03	200
Макароны	10.02.03	200
Макароны	11.02.02	300
Фанта	12.05.02	400
Фанта	13.05.03	100

Рис. 3

Отношение один-к-одному

Отношение один-к-одному имеет место, когда одной записи в родительской таблице соответствует одна запись в дочерней таблице. Данное отношение используют, если не хотят, чтобы таблица БД «распухала» от второстепенной информации.

Связь один-к-одному приводит к тому, что для чтения связанной информации в нескольких таблицах приходится производить несколько операций чтения, что замедляет получение нужной информации.

Подобно связи один-ко-многим, связь один-к-одному может быть жёсткой и не жёсткой. Пример отношения:

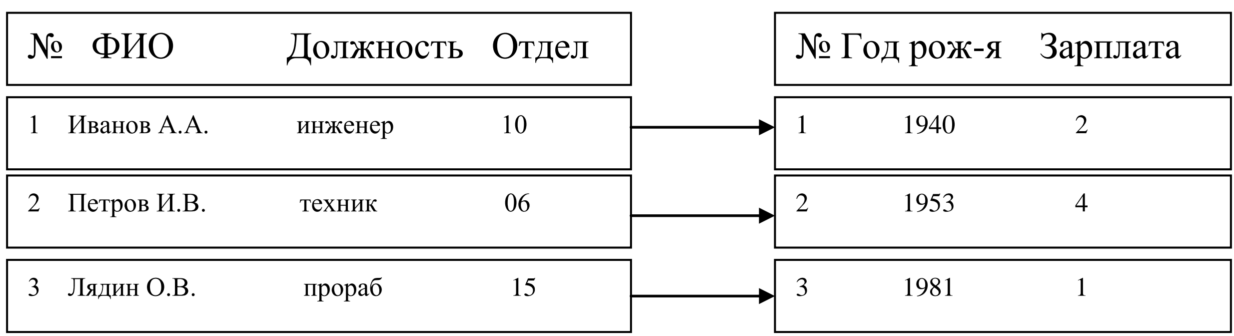


Рис. 4

Отношение многие-ко-многим.

Имеет место, когда одной записи родительской таблицы может соответствовать несколько записей в дочерней таблице. И одной записи дочерней таблицы может соответствовать несколько записей в родительской таблице.

Пример: каждой учебной группе соответствует несколько преподавателей. Каждый преподаватель может вести, во-первых, несколько разных предметов и, во-вторых, преподавать в разных группах.

Некоторые СУБД не поддерживают связи многие-ко-многим на уровне индексов и ссылочной целостности. Считается, что БД можно перестроить так, чтобы любая связь многие-ко-многим была заменена на одну или более связей один-ко-многим.

Пример отношения многие-ко-многим:

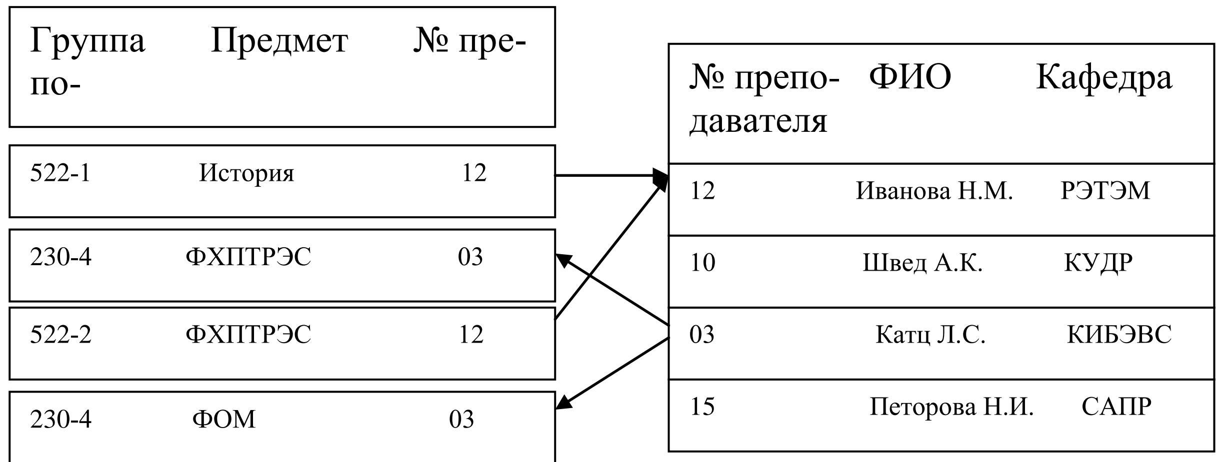


Рис. 5

Создание БД

Рассмотрим автоматизация одного из направлений деятельности отдела кадра некоторого предприятия.

Необходимо учитывать ситуацию приема сотрудников в определенное подразделение на определенную должность. Факт отслеживания увольнений необходимо вести через отдельную сущность Приказы об увольнении.

Факт приема фиксируется в документе Договор о приеме на работу сотрудника. Один и тот же сотрудник может работать в различных подразделениях и на разных должностях в разное время (в том числе сменить должность и пр.). При этом важно знать дату приема на работу сотрудника в подразделение на определенную должность. В соответствии с занимаемой должностью сотруднику назначается оклад.

Адрес не является обязательной частью информации о сотруднике. Данная информация хранится, как примечание к сотруднику.

Получаем отношения 1 Отдел: Много Договоров, 1 Должность: Много Договоров, 1 Сотрудник: Много договоров.

В соответствии с выше сказанным получаем следующую логическую модель данных (рис. 6).

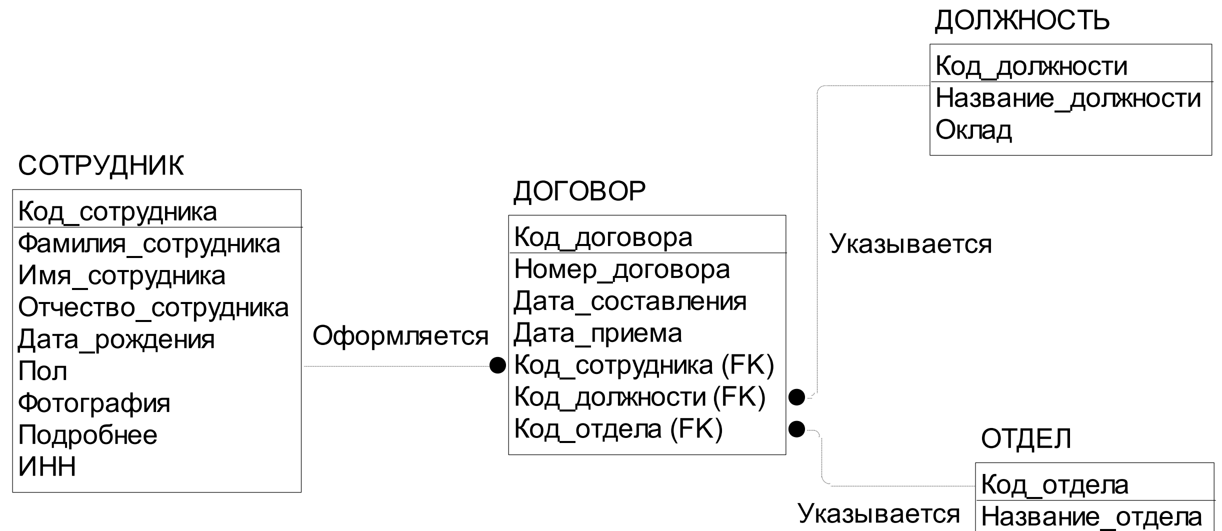


Рис. 6 — Логическая модель данных

Переходим от логической модели БД к ее физической реализации.

Для создания БД используется утилита Database Desktop (входит в поставку с Delphi).

Приведем соответствие логических и физических имен (табл. 4).

Таблица 4 — **Соответствие логических и физических имен**

Логическое имя	Физическое имя	Тип данных	Размер
Сотрудник	**People.db**	**Таблица Paradox 7**	
Код_сотрудника (PK)	ID_People	Autoicrement (±) — Автоинкрементное поле. Значения допустимы только для чтения. Обычно ключевое поле в составе первичного ключа. При добавлении новой записи значение поля вычисляется автоматически таким образом, чтобы в одной таблице не было одинаковых значений. Значение поля из удаленных записей не используется.	Первичный ключ
Фамилия_сотрудника	Last_Name	Alpha (A) — Символьные значения до 255 символов	50
Имя_сотрудника	First_Name	Alpha (A) — Символьные значения до 255 символов	30

Продолжение табл. 4

Логическое имя	**Физическое имя**	**Тип данных**	**Размер**
Отчество_сотрудника	Second_Name	Alpha (A) — Символьные значения до 255 символов	30
Дата_рождения	BirthDay	Date (D) — Значения даты (в диапазоне от 01.01.9999 до н.э. до 31.12.9999)	
Пол	Sex	Logical (L) — Логические значения («True», «False»). Значения регистров не имеет значения.	
Фотография	Photo	Graphic Fields (G) — Графические изображения	
Подробнее	Notes	Memo (M) — Строковые значения длиной более 255 символов. Максимальная длина не ограничена. От 1 до 240 символов могут храниться вместе с таблицей БД, остальные хранятся в виде Memo файла (имеет расширение .mb).	200
ИНН	INN	Alpha (A) — Символьные значения до 255 символов	20
Отдел	**Otdel.db**	**Таблица Paradox 7**	
Код_отдела (PK)	ID_Otdel	Автоинкрементное поле. Значения допустимы только для чтения. Обычно ключевое поле в составе первичного ключа. При добавлении новой записи значение поля вычисляется автоматически таким образом, чтобы в одной таблице не было одинаковых значений. Значение поля из удаленных записей не используется.	Первичный ключ
Название_ отдела	Otdel_Name	Alpha (A) — Символьные значения до 255 символов	50
Должность	**Job.db**	**Таблица Paradox 7**	
Код_должности (PK)	ID_Job	Автоинкрементное поле. Значения допустимы только для чтения. Обычно ключевое поле в составе первичного ключа. При добавлении новой записи значение поля вычисляется автоматически таким образом, чтобы в одной	Первичный ключ

Продолжение табл. 4

Логическое имя	**Физическое имя**	**Тип данных**	**Размер**
		таблице не было одинаковых значений. Значение поля из удаленных записей не используется.	
Название_ должности	Job_name	Alpha (A) — Символьные значения до 255 символов	40
Оклад	Oklad	Number (N) — Числовые значения с плавающей то кой в диапазоне $-10^{307}..+10^{308}$. Точность до 15 значащих цифр. Или Money ($) — Аналогичен типу Number, но предназначен для хранения денежных сумм. Число знаков после запятой по умолчанию –2. При показе значения выводится знак денежной единицы.	
Договор	Contract.db	**Таблица Paradox 7**	
Код_договора(PK)	Id_Contract	Autoicrement (±) — Автоинкрементное поле. Значения допустимы только для чтения. Обычно ключевое поле в составе первичного ключа. При добавлении новой записи значение поля вычисляется автоматически таким образом, чтобы в одной таблице не было одинаковых значений. Значение поля из удаленных записей не используется.	Первичный ключ
Номер_договора	Number	Alpha (A) — Символьные значения до 255 символов	20
Дата_составления	CreateDay	Date (D) — Значения даты (в диапазоне от 01.01.9999 до н.э. до 31.12.9999)	
Дата_приема	StartDay	Date (D) — Значения даты (в диапазоне от 01.01.9999 до н.э. до 31.12.9999)	
Код_сотрудника	Id_People	LongInteger (I) — Целочисленные значения в диапазоне –2147483648…+2147483647.	Внешний ключ
Код_должности	ID_Job	LongInteger (I) — Целочисленные значения в диапазоне	Внешний

Окончание табл. 4

Логическое имя	**Физиче-ское имя**	**Тип данных**	**Размер**
		–2147483648…+2147483647.	ключ
Код_отдела	ID_Otdel	LongInteger (I) — Целочисленные значения в диапазоне –2147483648…+2147483647.	Внешний ключ

Необходимо создать вторичные индексы (по полям внешних ключей) в таблице Contract.db:

1) вторичный индекс по полю внешнего ключа Id_People (Код_сотрудника (FK)) IDPeopleIndex. Хранится в файлах Contract.XG0 и Contract.YG0;

2) вторичный индекс по полю внешнего ключа ID_Job (Код_должности (FK)) IDJobIndex. Хранится в файлах Contract.XG1 и Contract.YG1;

3) вторичный индекс по полю внешнего ключа ID_Otdel (Код_отдела (FK)) IDOtdelIndex. Хранится в файлах Contract.XG2 и Contract.YG2.

Определить требования обязательного заполнения полям:

1) в таблице Otdel.db полю Otdel_Name;

2) в таблице Job.db полю Job_name;

3) в таблице People.db полю Last_Name.

В таблице Otdel.db полю Otdel_Name указать уникальность значений в поле, создав уникальный вторичный индекс OtdelNameIndex.

Таким образом, БД отдела кадра в нашем случае состоит из следующих объектов:

- Otdel.db — файл таблицы Отделы (Otdel):

– Otdel.px —файл первичного индекса по первичному ключу ID_Otdel;

– Otdel.val — файл по полю обязательному для заполнения Otdel_Name;

– Otdel.xg0 — файл уникального вторичного индекса OtdelNameIndex по полю Otdel_Name;

– Otdel.yg0 — файл уникального вторичного индекса OtdelNameIndex по полю Otdel_Name.

- Job.db — файл таблицы Должности (Job):

– Job.px — файл первичного индекса по первичному ключу ID_Job;

– Job.val — файл по полю обязательному для заполнения Job_name.

- People.db — файл таблицы Сотрудники (People):

– People.px — файл первичного индекса по первичному ключу ID_People;

– People.mb — файл мемо-поля Notes;

– People.val — файл по полю обязательному для заполнения Last_Name.

- Contract.db — файл таблицы Договоры (Contract):

– Contract.px — файл первичного индекса по первичному ключу Id_Contract;

– Contract.val — файл по полям обязательным для заполнения Number, StartDay,Id_People, ID_Job, ID_Otdel;

– Contract.xg0 — файл вторичного индекса по полю внешнего ключа Id_People (Код_сотрудника (FK)) IDPeopleIndex. Был создан первым;

– Contract.yg0 — файл вторичного индекса по полю внешнего ключа Id_People (Код_сотрудника (FK)) IDPeopleIndex. Был создан первым;

– Contract.xg1 — файл вторичного индекса по полю внешнего ключа ID_Job (Код_должности (FK)) IDJobIndex. Был создан вторым;

– Contract.yg1 — файл вторичного индекса по полю внешнего ключа ID_Job (Код_должности (FK)) IDJobIndex. Был создан вторым;

– Contract.xg2 — файл вторичного индекса по полю внешнего ключа ID_Otdel (Код_отдела (FK)) IDOtdelIndex. Был создан третьим;

– Contract.yg2 — файл вторичного индекса по полю внешнего ключа ID_Otdel (Код_отдела (FK)) IDOtdelIndex. Был создан третьим.

Создание простого приложения для работы со связанными таблицами

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

Добавим в приложение 2 компонента TTable. Один (с именем tOtdel вместо имени по умолчанию Table1) для работы с таблицей Otdel.db базы данных, а второй (с именем tContract вместо имени по умолчанию Table2) для работы с таблицей Contract.db базы данных. Добавим в форму 2 компонента TDataSource (srcOtdel и srcContract вместо имен по умолчанию DataSource1 и DataSource2). Установим свойство DataSet компонентов TDataSource в значения tOtdel и tContract соответственно. Разместим в форме 2 компонента TDBGrid (имена по умолчанию DBGrid1 и DBGrid2) и установим их свойство DataSource в значения srcOtdel и srcContract соответственно.

Реализация связи Master-Detail между наборами данных

Нам известно, что таблицы базы данных Otdel.db и Contract.db находятся в отношении «один-ко-многим». Поскольку мы определили ссылочную целостность между этими таблицами (первичный ключ — внешний ключ), можно сделать так, чтобы при установке указателя на запись в наборе данных tOtdel (ассоциированном с Otdel.db) показывались только записи о договорах приема на работу сотрудников в текущий отдел в tOtdel. Это реализуется через механизм связи наборов данных Master-Detail.

В инспекторе объектов для компонента tContract подчиненного набора данных установим значение свойства MasterSource в источник данных родительского набора данных srcOtdel. Переместимся на значение свойства MasterFields и нажмем кнопку эллипса.

В появившемся окне Field Link Designer установим параметры связи. В нашем случае в качестве параметра текущего индекса в поле Available Indexes необходимо указать вторичный индекс по полю внешнего ключа IDOtdelIndex, так как внешний ключ не входит в состав первичного ключа подчиненной таблицы. Если

бы внешний ключ входил в состав первичного ключа подчиненной таблицы, то в качестве текущего индекса необходимо было установить Primary. В списке Detail Fields (Detail — для подчиненного, дочернего набора данных) выберем поле IDOtdel, в списке Master Fields (Master — для главного, родительского набора данных) выберем поле IDOtdel и нажмем кнопку Add. В поле Joined Fields будет сформировано выражение IDOtdel -> IDOtdel. Произошла установка связи внешнего ключа IDOtdel в таблице Contract.db и первичного ключа IDOtdel в таблице Otdel.db. Нажмем кнопку Ок.

Как можно заметить, в компоненте tContract текущий индекс (свойство IndexName) заменен на индекс IDOtdelIndex, построенный по полю IDOtdel и свойство IndexFieldNames в значение IDOtdel. При этом свойство MasterFields установилось в IDOtdel — первичный ключ главной таблицы. В дальнейшем при программировании можно использовать данные свойства.

Исполним приложение. Теперь в наборе данных tContract показываются только записи по договорам отдела, текущего в наборе данных tOtdel. При этом при добавлении записи в набор данных tContract значение поля IDOtdel по умолчанию берется равным значению поля IDOtdel из текущей записи в наборе данных tOtdel.

ЛАБОРАТОРНАЯ РАБОТА № 3
РАБОТА С ПОЛЯМИ — КОМПОНЕНТ TFIELD

Применение TField

Компонент TField позволяет обращаться к полям таблиц баз данных.

Каждый набор данных — неважно, TTable или TQuery или др. — состоит из записей, а те, в свою очередь, состоят из полей. Таким образом, в составе записи имеется минимум одно поле.

В Delphi имеется возможность использовать при работе с НД или все поля, определенные в данной таблице БД на текущий момент, или использовать только часть существующих полей, или проводить добавление вычисляемых полей, полей из других наборов данных.

Использование только части имеющихся полей дает то преимущество, что значения неиспользуемых полей не могут быть изменены ни вследствие алгоритмической ошибки, ни вследствие злонамеренного умысла, поскольку неиспользуемые поля считаются неизвестными.

Существует два способа задания состава полей для НД.

Первый способ состоит в том, что после создания НД (компонент TTable или TQuery) не предпринимается никаких дополнительных действий по уточнению состава полей. Тогда:

– для компонента TTable — будет разрешен доступ ко всем полям, определенным в данный момент в таблице БД, связанной с компонентом TTable;

– для компонента TQuery — будет разрешен доступ ко всем полям (в том числе и результатам выражений), указанным в списке возвращаемых полей в операторе SELECT. Хотя этот оператор в результате выполнения SQL-запроса может возвращать НД, составленный из нескольких физических ТБД, сами поля будут считаться принадлежащими к единому НД, образовавшемуся в результате выполнения SQL-запроса из свойства TQuery SQL.

К полю в этом случае можно обращаться с помощью метода (функции) FieldByName компонентов TTable и TQuery

function FieIdByName(const FieldName: string): TField;

или через свойство указанных компонентов Fields [Index], которое возвращает указатель на тип TField. Подробнее об этом свойстве будет сказано ниже.

Второй способ определения состава полей заключается в том, что для данного НД поля как компоненты TField добавляются в форму с помощью *редактора полей* Delphi. О нем будет сказано чуть ниже.

К таким полям можно обращаться через его имя, определяемое в свойстве Name компонента TField, соответствующего данному полю.

По умолчанию, при добавлении в форму компонента TField, его имя генерируется так: берется имя НД (т.е. значение свойства *Name* компонента TTable или TQuery, к которому принадлежит поле) и к нему добавляется имя поля, взятое из структуры таблицы БД (TTable) или из запроса (TQuery).

Так, поле с именем Last_Name, используемого в НД tPeople, при добавлении в форму средствами редактора полей, получит имя tPeopleLast_Name. Это имя будет относиться целиком к компоненту TField, а не только к значению (например, «Иванов Иван Васильевич»), которое поле Last_Name содержит в текущей записи таблицы БД.

Таким образом, экземпляр tPeopleLast_Name компонента TField трактуется не как конкретное значение, которое принимает поле Last_Name в конкретной строке, а как весь столбец набора данных, обладающий единым поведением, едиными свойствами, методами и событиями для всех записей набора данных.

Компонент TField, как будет показано ниже, обладает рядом свойств, методов и событий, обращаться к которым следует через указание имени компонента TField и имени свойства, метода или обработчика события.

Когда поле набора данных определено в форме в качестве экземпляра компонента TField к нему можно обращаться, например, по имени (содержащемуся в свойстве Name), а также через метод (функцию) НД FieldByName и свойствоFields [Index].

Если хотя бы для одного поля НД создан компонент TField, то первый принцип, т.е. принцип использования всех полей таблицы БД или результата SQL запроса, отвергается. В НД будут считаться определенными только те поля, для которых созданы

компоненты TField, а иные поля — отвергаться как несуществующие для данного НД (TTable или TQuery).

К «несуществующим» полям обратиться из данного НД нельзя никак. Такие попытки будут возбуждать исключительные ситуации с сообщением «Field <имя> not found».

Вновь вернуться к принципу использования всех полей таблицы БД или результата SQL-запроса можно, удалив в редакторе полей все определенные ранее TField или добавив с его помощью недостающие поля.

То, какие поля определять в данном НД, зависит от назначения разрабатываемого приложения и диктуется целесообразностью.

Использование редактора полей для организации компонентов TField

Для того, чтобы определить один или несколько компонентов TField нужно:

1. Выбрать необходимый НД (компонент TTable или TQuery).
2. Нажать правую кнопку мыши.
3. Во всплывающем меню выбрать режим Fields Editor (запуская тем самым редактор полей).
4. Или вместо пунктов 2 и 3 два раза щелкнуть мышью по выбранному набору данных (запуская тем самым редактор полей).
5. Вновь нажать правую кнопку мыши и во всплывающем меню выбрать Add Fields.
6. В появившемся списке полей таблицы БД (TTable) или полей, участвующие запросе (TQuery), выбрать необходимые.
7. Если требуется добавить в редактор полей все поля таблицы БД, то нажать Add all Fields.
8. Нажать OK. Для каждого из указанных полей будет создан компонент TField (рисунок 6.1).
9. Если необходимо изменить свойства конкретного поля или написать обработчик для какого-либо события, необходимо в редакторе полей выбрать нужное поле и, используя инспектор объектов, установить значение в свойство или определить обработчик события.

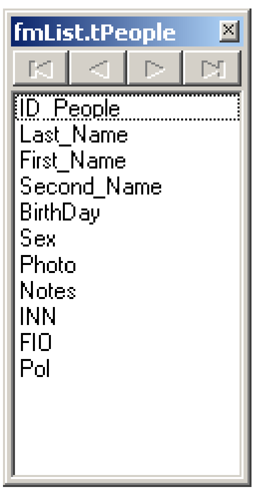


Рис. 7 — Список полей, определенных в редакторе полей

Типы полей

Поля| в таблицах баз данных различаются по типу — символьные, целочисленные, логические, BLOB-поля и т.д.

Соответственно, по типу различаются и компоненты TField, и собственно, TField есть родительский тип, определяющий базовые свойства и методы для своих потомков, типизированных полей. Иерархия компонентов — полей такова:

TField
- TBlobField большой двоичный объект
  - TGraphicField графическое поле (работает с содержимым BLOB-поля как с графическим изображением)
  - TMemoField мемо-поле (интерпретирует BLOB-поле как большой текст)
- TBooleanField логическое поле
- TBinaryField не типизированное двоичное поле
  - TBytesField поле для хранения байтовых значений фиксированной длины
    - TVarBytesField поле для хранения байтовых значений переменной длины
- TDateTimeField поле для хранения даты и времени
  - TDateField поле для хранения только даты

TTimeField поле для хранения только времени
TNumericField поле для хранения числовых значений
TBCDField BCD-значений
TFloatField значений с плавающей точкой
TCurrencyField в том числе в денежном формате
TIntegerField целочисленных значений
TAutoIncField в том числе автоинкрементных
TSmallIntField в том числе коротких целых
TWordField в том числе в формате беззнакового длинного целого
TStringField поле для хранения строковых значений.

Рассмотрим более подробно.

1) TStringField — хранит строковое значение длиной до 255 символов. Строки большей длины нужно хранить в BLOB-полях (TMemoField).

2) Целочисленные поля — применяются для хранения целых чисел различной длины:

- **TIntegerField** — от –2,147,483,648 до 2,147,483,647
- **TSmallintField** — от –32,768 до 32,767
- **TWordField** — от 0 до 65,535

Свойства **property MaxValue: Longint; и property MinValue: Longint;** могут определять максимальное и минимальное значение поля.

3) Числовые поля с плавающей точкой — применяются для хранения целых чисел различной длины:

– **TFIoatField** — числа, чьи абсолютные значения — 5.0*10-324 to 1.7*10+308 до 15-16

– **TCurrencyField** — аналогично TFIoatField, но в денежном формате

– **TBCDField** — вещественные десятичные числа с фиксированным числом разрядов после точки. До 18 символов. Диапазон представляемых чисел зависит от числа знаков. *Применяется только для Paradox.*

Свойство **property Precision: Integer;** позволяет указать число знаков после десятичной точки (по умолчанию 15).

Свойства **property MaxValue: Longint; и property MinValue: Longint;** могут определять максимальное и минимальное значение поля.

4) TBooleanField — содержит значения True или False.

5) Поля даты и времени:

– **TDateTimeField** — содержит значения даты и времени в формате TDateTime.

– **TDateField** — значения даты в формате Tdate

– **TTimeField** — значения времени в формате TTime.

6) поля для хранения значений произвольных форматов:

– **TBlobField** — произвольное байтовое поле без ограничения длины.

Метод **procedure LoadFromFile(const FileName: string);** загружает содержимое поля из файла, метод **procedure LoadFromStream(Stream: TStream);** — из потока.

Метод **procedure SaveToFile(const FileName: string);** сохраняет содержимое поля в файле, метод **procedure SaveToStream(Stream: TStream);** — в потоке.

Свойство **property BlobSize: Integer;** содержит размер в байтах BLOB-поля данной записи.

Свойство **property Transliterate: Boolean;** указывает, следует ли производить преобразование символов в ANSI в том случае, если BLOB-поля в таблице БД-источнике находятся не в ANSI-кодировке или содержат расширенные ASCII-символы. Когда свойство установлено в True, для преобразования ASCII символов в иную кодировку используется функция *AnsiToNative* и функция *NativeToAnsi* для перевода в ANSI.

Свойство **property BlobType: TBlobType;** возвращает тип BLOB-поля. Возможные значения: ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle, ftDBaseOle, ftTypedBinary.

– **TBytesField** — произвольное байтовое поле без ограничения длины. Не имеет методов LoadFromFile, LoadFromStream, SaveToFile, SaveToStream. Свойство **DataSize: Word** позволяет определить во время выполнения, сколько байт нужно для хранения поля в памяти.

– **TVarBytesField** — произвольное байтовое поле длиной до 65,535 байт. Текущая длина может быть получена из первых двух байт поля.

– **TMemoField** — строковое значение неопределенной длины (мемо-поле).

Метод **procedure Clear;** очищает мемо-поле.

Метод **procedure LoadFromFile(const FileName: string);** загружает содержимое поля из файла, метод **procedure LoadFromStream(Stream: TStream);** — из потока.

Метод **procedure SaveToFile(const FileName: string);** сохраняет содержимое поля в файле, метод **procedure SaveToStream(Stream: TStream);** — в потоке.

Свойство **property BlobSize: Integer;** содержит размер в байтах BLOB-поля данной записи.

Свойство **property Transliterate: Boolean;** указывает, следует ли производить преобразование символов в ANSI в том случае, если BLOB-поля в таблице БД — источнике находятся не в ANSI-кодировке или содержат расширенные ASCII-символы. Когда свойство установлено в True, для преобразования ASCII символов в иную кодировку используется функция *AnsiToNative* и функция *NativeToAnsi* для перевода в ANSI.

Для работы с мемо-полями в БД Delphi предоставляет компонент **TDBMemo.**

– **TGraphicField** — произвольное байтовое поле, трактуемое как графическое изображение. Для работы с полями графических изображений Delphi предоставляет компонент **TDBImage.**

Обращения к полям и их значениям

Как было сказано в пункте 1, следует различать обращение к полю и обращение к его значению. Общая запись обращения к значению поля:

НД.ПолеНД.Значение

Обращение к значению поля

К значению поля можно обратиться, например, при помощи свойств Value и AsNNN.

Свойство

property **Value : Variant**

возвращает значения следующих типов:

```
property Value: Variant;     // Все компоненты
property Value: string;      //TStringField, TBlobField
property Value: Longint;     //TAutoIncField, TlntegerField
                             //TSmallintField, TWordField
property Value: Double;      //TBCDField, TCurrencyField
                             //TFloatField
property Value: Boolean;     //TBooleanField
property Value: TDateTime,        // TdateTimeField, TDateField,
TTimeField
```

Аналогичные значения возвращает свойство набора данных FieldValues.

Обращение к значению поля через свойство AsNNN

Часто необходимо пользоваться свойствами приведения типов полей.

Существуют следующие свойства приведения типов полей:

```
property AsBoolean: Boolean;
property AsCurrency: Currency;
property AsDateTime: TDateTime;
property AsFloat: Double;
property Aslnteger: Integer;
property AsString: String;
property AsVariant: Variant.
```

Рассмотрим свойства семейства AsNNN более подробно:

– property AsBoolean: Boolean; — числовые значения приводятся к типу Boolean, если содержат 0 (False) или 1 (True). Символьные значения — если содержат в качестве первого символа «Y», «y», «T» или «t» (или «Yes» или «True»), и False во всех иных случаях;

– property AsDateTime: TDateTime; — для приведения к типу TDateTime значений TDateField, TDateTimeField и TTimeField,

а также для приведения к типу TDateTime строковых значений, находящихся в соответствующем формате;

– property AsFloat: Double; — служит для приведения к типу Double значении полей TFloatField, TBCDField и TCurrencyField;

– property AsInteger: Longint; — служит для приведения к типу Longint полей типа TIntegerField, TSmallintField и TwordField. Для полей типа TStringField преобразование к Longint выполняется, если оно возможно.

– property AsCurrency: Currency; — служит для приведения к типу Currency;

– property AsString: string; — служит для приведения к типу String;

– property AsVariant: Variant; — служит для приведения к типу Variant.

Каждое из этих свойств приводит значение поля к соответствующему типу данных, обозначенному в названии свойства. Например, если tPeopleID_Otdel — компонент TIntegerField (поле, хранящее целочисленные значения), то для приведения его к типу String можно воспользоваться свойством:

```
Edit1.Text:=tPeopleID_Otdel.AsString;
```

Несомненно, тип поля должен быть совместимым с типом данных, к которому приводится значение поля. Например, если tPeopleBirthDay — компонент TDateField (поле, хранящее значения даты в формате TDate), то попытка привести его к несовместимому логическому типу Boolean:

```
if tPeopleBirthDay.AsBoolean then
```

приведет к ошибке.

В таблице 5 показана совместимость значений полей разных типов.

Используемые обозначения:

= — типы равнозначны;

+ — преобразование возможно;

+RI — преобразование возможно, округление до ближайшего целого;

? — преобразование происходит, если возможно; часто зависит от формата показа (свойство DisplayFormat);

x — преобразование не разрешено;

memo — имеет значение для memo-поля.

Таблица 5 — **Совместимость значений полей разных типов**

Тип поля	**AsString**	**AsInteger**	**AsFloat**	**AsDate-Time**	**AsBoolean**
TStringField	=	?	?	?	?
TIntegerField	+	=	+	x	x
TSmallIntField	+	=	+	x	x
TWordField	+	=	+	x	x
TFloatField	+	+RI	=	x	x
TCurrencyField	+	+RI	=	x	x
TBCDField	+	+RI	=	x	x
TDateTimeField	+	x	Преобразование даты к числу дней с 01.01.0001	=	x
TDateField	?	x	Преобразование даты к числу дней с 01.01.0001	=	x
TTimeField	?	x	Преобразование времени делением на 24 часа	=	x
TBooleanField	В строку «True» или «False»	x	x	x	=
TBytesField	+	x	x	x	x
TVarBytesField	+	x	x	x	x
TBlobField	memo				
TMemoField	memo				
TGraphicField	memo				

Обращение к полю

К полю можно обратиться, указав имя поля несколькими способами, например:

1. Если полю соответствует компонент TField — через имя данного компонента, которое определяется свойством Name.

Повторим, что по умолчанию имя компонента TField устанавливается как результат сцепления имени TTable или TQuery и собственно имени поля в таблице БД. Например, для поля Last_Name, определенного в таблице БД, работа с которой происходит через набор данных tPeople (ассоциированном с People.db), по умолчанию будет выбрано имя tPeopleLast_Name. Например:

```
tPeopleLast_Name.Value:='Петров';
//или
tPeopleLast_Name.AsString:='Петров';
```

2. Используя метод (функцию) FieldByName ('ИмяПоля') набора данных,

```
function FieldByName(const FieldName: string): TField;
```

Например:

```
tPeople.FieldByName('Last_Name').Value:='Сидоров';
//или
tPeople.FieldByName('Last_Name').AsString:='Сидоров';
```

3. Используя свойство Fields[индекс] набора данных,

```
property Fields[Index: Integer]: TField;
```

Индекс является порядковым номером поля в определении таблицы БД. Отсчет идет от 0. Например, пусть поле First_Name определено в таблице БД третьим по счету. Тогда его индекс равен 2 и использование поля может происходить так:

```
tPeople.Fields[2].AsString:='Василий';
//или
tPeople.Fields[2].Value:='Василий';
```

Если поле входит в индекс данного НД, свойство IsIndexField: Boolean, возвращает во время выполнения значение True.

4. Используя свойство набора данных

property FieldValues[const FieldName: string]: Variant;

Это свойство позволяет обращаться к полю через его имя, указываемое как содержимое параметра FieldName, например:

tPeople.FieldValues['Last_Name']:='Петров';

Поскольку свойство FieldValues принимается для набора данных умолчанию, его имя при обращении к полю можно опускать:

tPeople['Last_Name']:='Петров';

Примечание: Более предпочтительным считается обращение к полю через его имя или через метод (функцию) FieldByName, поскольку в этом случае мы обращаемся к конкретному полю по его имени. Следовательно, к несуществующему полю обратиться нельзя.

Если поле Last_Name удалить из структуры таблицы БД, то обращение к нему по имени из ассоциированного с данной таблицей БД НД приведет к ошибке.

Менее предпочтительным является обращение к полю через свойство набора данных Fields[индекс]. Если поле Last_Name было объявлено в таблице БД вторым по счету (обращение tPeople.Fields[1]), а затем удалено из структуры таблицы БД, то обращение Fields[l] в программном коде будет воспринято как обращение к физически второму полю в таблице БД. А этим полем будет третье поле, следовавшее за Last_Name перед тем, как Last_Name удалено из структуры таблицы БД. После удаления третье поле станет по счету вторым.

Налицо алгоритмическая ошибка, которая может быть не распознана, если поле, ставшее после удаления Last_Name вторым, имеют одинаковые или совместимые типы.

Такие ошибки очень трудно локализовать. Поэтому, если это возможно, то следует воздержаться от обращения к полю через Fields[индекс].

Свойство Index компонента TField содержит порядковый номер поля:

– в таблице БД, если для компонента TTable, ассоциированного с данной таблицей БД, не создано ни одного компонента TField;

– в списке компонентов TField, относящихся к данному НД — для компонента TTable, если для него определен хотя бы один TField, и всегда — для компонента TQuery.

Порядок следования полей важен, когда содержимое НД визуализируется посредством компонента TDBGrid. В этом случае номер столбца в компоненте TDBGrid соответствует номеру поля в списке TField. И наоборот, если изменить порядок столбцов в компоненте TDBGrid (например, «перетащив» столбец на другое место), то это приведет к изменению индекса и Fields[1] до перетаскивания будет относиться к другому полю, нежели Fields[l] после перетаскивания столбцов, поскольку свойство Index перетаскиваемого и некоторых других полей изменится.

Это относится как к случаю, когда TField для НД определены, так и к случаю, когда используются все поля таблицы БД. В последнем случае «перетаскивание» столбца в компоненте TDBGrid на новое место, конечно, не изменит физического порядка следования полей в структуре таблицы БД; однако с логической точки зрения его индекс (порядковый номер) для данного НД изменится.

Последствия, которые могут принести в структуру НД изменения свойства Index, делают обращения к полю через свойство Fields[индекс] набора данных нежелательными.

Особенности показа полей в визуальных компонентах во время выполнения

Свойства property DisplayLabel:string и DisplayName:string

DisplayLabel используется, чтобы назначать заголовки столбцов, например, в компоненте TDBGrid. Заголовки TDBGrid используют свойство DisplayName полей, чьи величины они представляют. Установка DisplayLabel изменяет только для чтения свойство DisplayName от FieldName до строки определенной как DisplayLabel.

Например:

tPeopleLast_Name.DisplayLabel:='Фамилия';

Кроме того, свойство DisplayName, например, используется для отображения ошибок ввода данных для этого поля (исключения exceptions BDE).

Свойство property Visible:boolean

Свойство Visible применяется для того, чтобы скрывать столбцы, например, в компоненте TDBGrid.

Если свойство Visible установлено в False, то поле становится невидимым, например, в компоненте TDBGrid. Например:

tPeopleLast_Name.Visible:=False;

Свойство property DisplayWidth:integer

DisplayWidth определяет ширину видимой части поля, например, в компоненте TDBGrid.

Например:

tPeopleLast_Name.DisplayWidth:=30;

Свойство property ReadOnly:boolean

Если свойство ReadOnly установлено в значение True, то значения в поле доступны «только для чтения». Например:

tPeopleLast_Name.ReadOnly:=True;

Свойство property Required:boolean

Если свойство Required установлено в значение True, то значения в поле должны быть обязательно заданы. Например:

tPeopleLast_Name.Required:=True;

Другие свойства, события будут рассмотрены в ходе дальнейшего выполнения лабораторных работ.

Проверка введенного в поле значения

Свойство IsNull

Свойство **IsNull:Boolean** во время выполнения возвращает True, если поле содержит пустое значение.

Проверить введенное в поле значение на его соответствие некоторым ограничениям или условиям можно, например, в обработчиках события OnSetText, OnValidate, OnChange (вызываются в приведенной последовательности). Данные события наступают при изменении значения поля либо вручную, либо программно. Будут рассмотрены позже. Данный подход контроля правильности значений называется ориентированным на поля.

Например:

```
procedure TForm1.tPeopleLast_NameValidate(Sender: TField);
begin
 if Sender.AsString='' then
 raise Exception.Create('Обязательно необходимо указать фамилию!');
end;
```

Существуют и другие подходы. Например, подход, ориентированный на записи. Он состоит в том, что в структуре таблицы БД при ее определении описываются ограничения на значения, которые может принимать данное поле. В этом случае контроль правильности ведется автоматически.

Значение поля по умолчанию и ограничения на значение поля

Значение поля по умолчанию можно установить при помощи свойства

property DefaultExpression: string;

В случае указания значений, отличных от целочисленного, они должны заключаться в кавычки. В компоненте TField могут быть определены ограничения на значения этого или иных полей. Ограничение указывается при помощи SQL-подобного синтаксиса в свойстве

property CustomConstraint: string;

Например:

```
JobOklad.CustomConstraint:='Oklad >=1500 and Oklad <=10000';
```

Свойство

property ConstraintErrorMessage: string;

позволяет указать сообщение об ошибке, выдаваемое пользователю в случае, если введенное значение поля не удовлетворяет ограничению, указанному в свойстве CustomConstraint, например:

tJobOklad.ConstraintErrorMessage('Оклад должен быть в диапазоне 1500...10000');

И другие.

Создание вычисляемых полей

Если необходимо создать вычисляемое поле, значение которого вычисляется по значениям других полей, поступают так:

1. В редакторе полей необходимо создать новое поле, пометив его как поле Calculated. Для этого нужно сделать текущим (при помощи мыши) необходимый НД, нажать правую кнопку мыши, выбрать в меню Fields Editor (или двойное нажатие мышью) и снова нажать правую кнопку мыши и выбрать в меню New Field. Затем в окне диалога необходимо указать имя поля, его тип и для строковых полей длину (рис. 8). Создадим вычисляемое поле Age (возраст сотрудника).

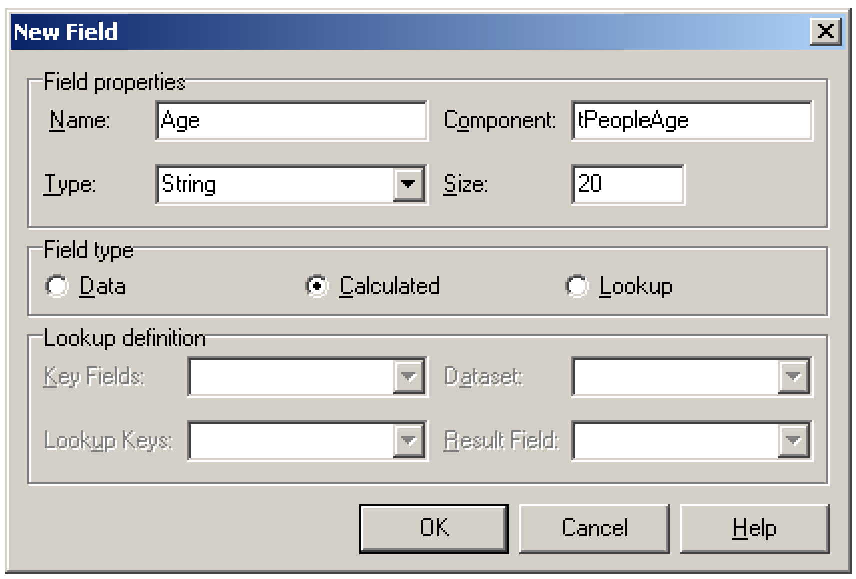


Рис. 8 — Создание нового вычисляемого поля Age (возраст сотрудника)

Для нового поля будет создан компонент TField, доступ к которому отныне можно осуществлять в редакторе полей.

2. Для компонента НД, к которому принадлежит вычисляемое поле, необходимо определить обработчик события OnCalcFields.

Например, для НД tPeople, ассоциированному с таблицей БД «Сотрудники», будем заносить в вычисляемое поле tPeopleAge: TStringField значение возраста сотрудника строкой, например,»42 года», «35 лет», «51 год» и т.д. В случае, если дата рождения сотрудника не указана, то поле tPeopleAge содержит значение «Неизвестен».

Возраст сотрудника будем вычислять, как разность между текущим годом (взятому из полной текущей даты) и годом даты рождения (взятому из полной даты рождения). Для генерации строки «лет», «год», «года», «лет» создадим специальную функцию

function GetAge(Dt:TDate):string;

для которой:

```
function GetAge(Dt:TDate):string;
var y,y2,M,D:word;
begin
  Result:='Неизвестен!';
  DecodeDate(Dt,Y,M,D);
  DecodeDate(Now,Y2,M,D);
  Y:=Y2-Y;
  if Y<=0 then exit;
  if (Y<=19)and(Y>10)
    then Result:='лет'
    else case Y mod 10 of
      1:Result:='год';
      2..4:Result:='года';
      else Result:='лет';
    end;
  Result:=IntToStr(Y)+' '+Result;
end;
```

И тогда для вычисляемого поля tPeopleAge на событии НД tPeople OnCalcFields можно записать:

```
procedure TForm1.tPeopleCalcFields(DataSet: TDataSet);
begin
  if not tPeopleBirthDay.IsNull
    then
tPeopleAge.AsString:=GetAge(tPeopleBirthDay.AsDateTime)
    else tPeopleAge.AsString:='Неизвестен';
end;
```

Событие OnCalcFields возникает всякий раз, когда курсор (указатель записи) перемещается в НД от записи к записи. Это событие возникает и при инициализации НД (после открытия), а также после фильтрации записей в НД, что, впрочем, также связано с изменением положения указателя записи.

Кроме того, если свойство набора данных AutoCalcFields установлено в True, то событие OnCalcFields наступает также и при модификации значений не вычисляемых полей в режимах добавления и редактирования данного НД.

Процедура — обработчик события OnCalcFields содержит реализацию алгоритма вычисления значения вычисляемого поля или группы полей.

Необходимо помнить, что в этом обработчике значение может быть присвоено только вычисляемому полю и не может — полю, определенному в структуре таблицы БД.

В наборе данных tPeople, кроме вычисляемого поля Age (Возраст сотрудника) можно создать, например, вычисляемые поля:

– поле tPeoplePol: TStringField — Пол сотрудника. Может принимать значения «мужской», «женский», «не указан» в зависимости от значения в поле Sex;

– поле tPeopleFIO: TStringField — Фамилия Имя Отчество сотрудника. Можно в форме Фамилия И.О.

Можно предложить другие варианты вычисляемых полей к рассматриваемой БД.

Создание полей выбора данных (lookup-полей)

Кроме обычных полей, связанных с полями таблицы БД и вычисляемых полей, в Delphi имеется возможность создавать поля выбора данных (lookup поля).

Поля выбора данных одного набора данных содержат значения их другого набора данных, связанного по ключу с НД, к которому принадлежит поле выбора данных. Поле выбора данных всегда доступно только для чтения и не может быть одновременно полем выбора данных и вычисляемым полем. Реляционное отношение НД, служащего источником значений для поля выбора данных и НД, к которому оно принадлежит, есть «один-ко-многим» и реже «один-к-одному». Это означает, что на один вариант значения в наборе данных — источнике должно приходиться одно или несколько связанных значений в НД, к которому принадлежит поле выбора.

Для определения поля набора данных необходимо создать новое поле в редакторе полей, сразу же установив радио-группу Field Type в значение Lookup (рис. 9).

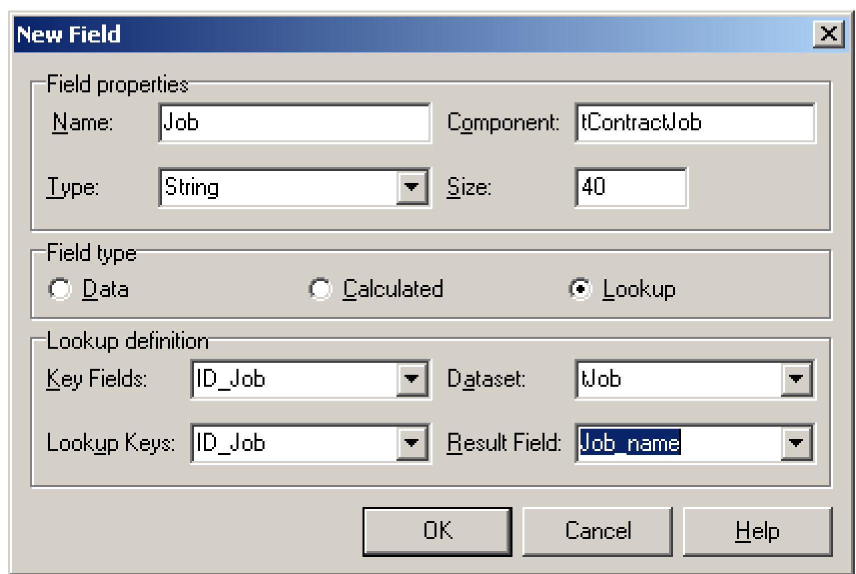


Рис. 9 — Создание поля выбора tContractJob

Затем устанавливаем значения свойств:

– DataSet — Имя НД — источника значений для поля выбора данных;

– Key Fields — Индексные поля набора данных — владельца поля выбора данных. По этим полям НД — владелец соединяется с НД — источником значений поля выбора данных. Если в индексе имеется несколько полей, они перечисляются через точку с запятой;

– Lookup Keys — Индексные поля НД — источника значений для поля выбора. По значениям этих индексных полей устанавливается связь набора — источника со значениями индексных полей НД — владельца поля выбора. Если в индексе имеется несколько полей, они перечисляются через точку с запятой;

Result Field — Поле набора данных — источника, возвращаемое в качестве результата. Необходимо следить, чтобы тип вновь создаваемого поля и поля результата совпадали.

Указанным выше параметрам редактора полей соответствуют свойства компонента TField:

- property LookupDataSet: TDataSet;
- property KeyFields: string;
- property LookupKeyFields: string;
- property LookupResultField: string;

Аналогичным по последствиям будет установка соответствующих свойств в инспекторе объектов для вновь добавляемого поля. Заметим, что свойство поля

property Lookup: Boolean;

должно быть установлено в True.

Для приведенного примера получаем:

- tContractJob.LookupDataSet:= tJob;
- tContractJob.KeyFields:='ID_Job';
- tContractJob.LookupKeyFields:='ID_Job';
- tContractJob.LookupResultField:='Job_name';

В данном примере показана реализация отношения «один-ко-многим» «1 Должность — М Договоров.

Таким образом, поля выбора данных можно использовать для автоматического занесения информации в одну таблицу БД из другой таблицы БД.

Стоит отметить, что для хранения индексов по цифровым полям и их использования для доступа к данным требуется много меньше дискового пространства и времени, нежели для хранения индексов по оригинальным значениям. В нашем примере это обосновано: лучше строить индекс по коду должности, нежели по символьному значению поля Название должности, что обуславливает ввод суррогатного ключа ID_Job. Это особенно актуально,

если символьное поле имеет большую длину. Индексы по таким полям получаются очень большими.

Другой выгодой от использования кодов является то, что в справочнике «Должности» коды должности ID_Job представляют собой автоинкрементное поле, т.е. поле, уникальное значение которого BDE устанавливает автоматически. В дальнейшем его менять нельзя. Это снимает необходимость каскадного изменения в дочерней таблице БД «Сотрудники» при изменении значения поля связи (ID_Job) в родительской таблице БД «Должности».

Механизм **каскадных изменений** используют, чтобы предотвратить потерю ссылочной целостности.

Когда говорят о **ссылочной целостности**, то имеют ввиду совокупность связей между отдельными таблицами во всей БД. Нарушение хотя бы одной такой связи делает информацию в БД недостоверной.

Обычно для реализации ссылочной целостности в дочерней таблице создают внешний ключ, в который входят поля связи дочерней таблицы. Этот ключ для дочерней таблицы по составу полей должен совпадать с первичным ключом родительской таблицы или реже— с частью первичного ключа.

Механизм каскадных изменений состоит в обеспечении следующих действий:

– при изменении поля связи в записи родительской таблицы следует синхронно изменить значение полей связи в соответствующих записях дочерней таблицы;

– при удалении записи в родительской таблицы следует удалить соответствующие записи в дочерней таблице.

Изменения или удаления в записях дочерней таблицы при одновременном изменении (удалении) записи родительской таблицы называются каскадными изменениями и каскадными удалениями.

Существует другая разновидность каскадного удаления: при удалении родительской записи в записях дочерней таблицы значения полей связи обнуляются, при этом дочерние таблицы будут содержать избыточные данные.

Механизм каскадных воздействий состоит в обеспечении следующих требований:

1) необходимо запретить изменения поля связи в записи дочерней таблицы без синхронного изменения полей связи в дочерней и родительской таблицах. Обычно инициатива изменения поля связи реализуется в записи родительской таблицы;

2) при изменении поля связи в родительской таблице, следует синхронно изменить значения полей связи в соответствующих записях дочерней таблицы;

3) при удалении записи в родительской таблице, следует удалить соответсвующие записи в дочерней таблице.

Автоинкрементные поля более свойственны локальным СУБД. Для удаленных (серверных) СУБД автоинкрементные поля заменяются другими механизмами, например, генераторами (InterBase).

Если бы связь между данными таблицами БД была установлена по полю Job_name, а не по ID_Job, то при изменении значения должности (например, с «приват-доцент» на «доцент») в таблице БД «Должности» должны были бы измениться значения всех записей в таблице БД «Договора», у которых поле Job_name содержит значение «приват-доцент».

Однако поскольку мы используем для связи между названными таблицами код, в таблице БД «Должности» значение поля Job_name можно менять сколь угодно много раз — на связь между таблицами БД это не окажет никакого влияния.

Заметим, что по полю Job_name в таблице БД «Должности» желательно построить уникальный индекс, чтобы предотвратить возможность ввода двух одинаковых должностей с разными окладами. Или отслеживать уникальность значения в поле Job_name программно.

Требования к выполнению лабораторной работы № 3

1. Провести ознакомление с работой с полями — компонентом TField.

2. Показать работу со свойствами полей в визуальных компонентах.

3. Создать вычисляемое поле.

4. Создать поле выбора данных (lookup поле).

ЛАБОРАТОРНАЯ РАБОТА № 4
ОБЩИЕ ПРИНЦИПЫ РАБОТЫ С НАБОРАМИ ДАННЫХ

Понятие наборов данных

Под набором данных в Delphi понимается группа записей из одной или нескольких таблиц БД, доступная для использования через компоненты TTable или TQuery.

Как набор данных, могут рассматриваться и другие компоненты, например, TStoredProc (хранимая процедура в серверной базе данных, возвращающая набор данных) и др. Однако для локальных СУБД применятся в основном компоненты TTable и TQuery. На них и будем ориентироваться.

Если рассматривать иерархию компонентов Delphi, компоненты TTable и TQuery являются наследниками компонента TDBDataSet, потомка компонент л TBDEDataSet, который, в свою очередь, является наследником компонент TDataSet.

Компоненты TTable и TQuery имеют общие свойства, методы и события; это обусловливается тем, что они имеют общих «родителей». При этом TTable и TQuery называются общим термином «набор данных» (НД). В литературе по Delphi TTable и TQuery при рассмотрении их общих свойств часто называют типом DataSet, по имени типа их предка. TTable и TQuery имеют также свойства, методы и события, присущие только TTable или только TQuery.

Перед рассмотрением общих возможностей для работы с НД разберем общие черты и различия TTable и TQuery.

Набор данных TTable

TTable содержит записи, источником которых может быть только одна таблица БД. Состав записей зависит от того, производится ли в TTable фильтрация по какому-либо условию. Если нет, то в TTable будут представлены все записи таблицы БД, ассоциированной с данной TTable; если да, то в TTable попадут только записи, удовлетворяющие условию фильтрации.

По той причине, что одна и та же TTable может в один момент времени содержать не отфильтрованные записи, в другой —

отфильтрованные по одному условию и в третий момент времени — записи, отфильтрованные по третьему условию, считают, что НД TTable содержит подмножество множества записей таблицы БД, которая ассоциирована с данным TTable.

Псевдоним БД указывается в свойстве DatabaseName, имя ассоциированной таблицы БД — в свойстве TableName.

Набор данных TQuery

TQuery содержит записи, источником которых могут являться несколько таблиц БД, а также агрегированные значения (такие как сумма, минимум, максимум, среднее), просчитанные по полям одной или нескольких таблиц.

НД формируется так: выполняется SQL-запрос, представленный оператором SELECT (свойство SQL), и в качестве НД возвращаются записи из таблиц-источников, удовлетворяющие определенным условиям (если они имеются).

Таблицы БД-источники перечисляются в разделе FROM оператора SELECT.

Условия выборки записей указываются в разделе WHERE.

Записи результирующего НД состоят из полей, перечисленных после ключевого слова SELECT (или, если указан символ * — всех полей). Более подробно об операторе SELECT и иных SQL-операторах рассмотрим позже.

Псевдоним БД указывается в свойстве DatabaseName. В случае, если псевдоним указан, все таблицы БД в разделе FROM оператора SELECT считаются принадлежащими данной БД. В противном случае для каждой таблицы БД необходимо указывать маршрут поиска, а если он не указан, таблицы БД должны располагаться в текущем каталоге ОС.

Как и в случае использования TTable, НД TQuery может содержать полное множество записей какой-либо таблицы БД или множество записей, удовлетворяющих условию фильтрации.

Состояния наборов данных

НД могут находиться, например, в одном из следующих состояний (табл. 6).

Таблица 6 — **Примеры состояний НД**

Состояние	Краткое описание
dsInactive	НД закрыт
dsBrowse	Состояние по умолчанию для открытого НД. Показывает, что записи просматриваются, но в данный момент не изменяются.
dsEdit	НД находится в состоянии редактирования текущей записи (после явно или неявно вызванного метода Edit).
dsInsert	НД находится в состоянии добавления новой записи (после явно или неявно вызванного метода Insert или Append).
dsSetKey	НД находится в состоянии поиска записи по критерию, заданному методами FindKey, GotoKey, FindNearest или GotoNearest. По окончании поиска НД переходит в состояние dsBrowse.
dsCalcFields	Выполняется установление значений вычисляемых полей (по алгоритму, заданному в обработчике события OnCalcFields). В данном режиме изменения в НД вноситься не могут. После выхода из режима НД переходит в предыдущее состояние.
dsFilter	Обрабатывается фильтрация записей в НД при свойстве Filtered, установленном в True. Имеет место текущий вызов события OnFilterRecord для определения того, удовлетворяет ли текущая запись условию фильтрации, описанному в обработчике данного события. После выполнения события OnFilterRecord НД переводится в состояние dsBrowse.

Рассмотрим методы, которые могут переводить БД из одного состояния в другое.

1) **dsInactive → dsBrowse**

НД во время выполнения программы можно открыть методами Table.Open, Query. Open. Во время разработки и во время выполнения НД можно открыть, установив в True свойства Table.Active и Query.Active.

2) **dsBrowse → dsInactive**

НД во время выполнения программы можно закрыть методами Table.Close, Query. Close. Во время разработки и во время

выполнения НД можно закрыть, установив в False свойства Table.Active и Query.Active.

Заметим, что если какая-либо запись на момент закрытия НД находится в режиме редактирования (dsEdit) или добавления новой записи (dslnsert), то применение метода Close не приводит к автоматической выдаче метода Post (сохранить). Таким образом, НД закрывается, находясь в режимах dslnsert или dsEdit, а не dsBrowse. В этом случае изменения, сделанные в записи, не запоминаются.

Для перевода НД из укачанных режимов в режим dsBrowse, перед тем как НД закрывается, используйте обработчик события BeforeClose. Заметим, что описанная ситуация будет встречаться в первую очередь для внезапно или принудительно закрываемых НД.

3) **dsBrowse → dsEdit**

Перевести НД в режим редактирования можно методом Edit. После этого значения полей текущей записи можно изменять.

4) **dsEdit → dsBrowse**

Метод Post приводит к запоминанию измененной записи в НД. Метод Cancel отменяет изменения, сделанные в полях записи. Запись не запоминается в НД.

5) **dsBrowse → dslnsert**

Перевести НД в режим вставки можно методами Insert или Append. После этого в программе становится доступна пустая запись, полям которой нужно присвоить какие-либо значения. Чтобы полям новой записи присвоить умалчиваемые значения, следует воспользоваться обработчиком события OnNewRecord.

6) **dslnsert → dsBrowse**

Метод Post добавляет новую запись в НД. Если НД не находится в режиме dsInsert, то возбуждается исключительная ситуация.

Метод Cancel отменяет добавление новой записи в НД. Содержимое полей, назначенных новой записи, теряется.

7) **dsBrowse → dsSetKey**

НД находится в данном состоянии, когда осуществляется поиск записи, удовлетворяющей условию, установленному методом SetKey (и затем, возможно, измененному методом EditKey). Именно эти методы и переводят НД в режим dsBrowse. Поиск за-

писи производится одним из следующих методов: GoToKey, GoToNearest, FindKey, FindNearest. В случае успешного или неуспешного завершения метода поиска, НД переводится в состояние dsBrowse.

8) **dsBrowse → dsFilter**

НД находится в данном состоянии всякий раз, когда приложение обрабатывает событие OnFilterRecord при фильтрации записей (при свойстве Filtered = True). При этом НД переводится из состояния dsBrowse в состояние dsFilter. Это предотвращает модификацию НД во время фильтрации. После завершения вызова обработчика события OnFilterRecord НД переводится в состояние dsBrowse. Вызов события OnFilterRecord производится для каждой записи НД при установке свойства Filtered в состояние True.

Получить текущее состояние НД можно, используя свойство

property State: TDataSetState;

Оно возвращает, например, следующие константы: dslnactive, dsBrowse, dsEdit, dslnsert, dsSetKey, dsCalcFields, dsFilter.

Например:

```
if tPeople.State = dsInactive        then tPeople.Active:=True;
```

Реакция на изменение состояния набора данных

Событие OnStateChange (компонент TDataSource) наступает всякий раз при изменении состояния НД. Следующий пример показывает, как отобразить на экране (в компоненте Label 1) сообщение о текущем состоянии НД:

```
procedure TForm1.DataSource1StateChange(Sender: TObject);
var s:string;
begin
  case tPeople.State of
    dsInactive      :   s:='Не активен';
    dsBrowse        :   s:='Просмотр';
    dsEdit   :   s:='Редактирование';
    dsInsert :   s:='Добавление';
    dsSetKey        :   s:='Установка ключа';
    dsCalcFields:  s:='Вычисляемое поле';
```

```
    dsFilter :   s:='Фильтрация';
  end; //case
  Label1.Caption:=s;
end;
```

Свойства некоторых компонентов, например Enabled, могу зависеть от состояния НД. Например, если кнопка Button1 должна быть доступной для нажатия только в режиме dslnsert, то допустим такой код:

```
procedure TForm1.DataSource1StateChange(Sender: TObject);
begin
  Button1.Enabled:=(tPeople.State = dsInsert);
end;
```

Навигация по набору данных

Существует два основных способа работы с записями в НД.

Способ, основанный на использовании операторов SQL, предполагает оперирование группами записей. Именно так работают SQL-операторы группового обновления НД UPDATE, INSERT, DELETE и выборки групп записей SELECT. Записи, удовлетворяющие некоторому условию, выдаются группами; даже если условию удовлетворяет только одна запись, считается, что в данном случае группа состоит из одной записи.

Второй способ состоит в оперировании единичными записями. Если необходимо изменить, добавить или удалить группу записей, необходимая операция выполняется для каждой из таких записей. Для этого такие записи в НД нужно отыскать, для чего применяются навигационные методы. Они всегда работают с единичной записью и связанны с понятием курсора НД.

Понятие курсора набора данных. Под курсором набора данных понимается указатель текущей записи в конкретном наборе данных. Текущая запись — та запись, над которой в данный момент времени можно выполнять какие-либо операции (удаление, изменение, чтение значений, содержащихся в записи полей).

Существует 5 основных методов для изменения курсора НД (табл. 7).

Таблица 7 — **Методы для изменения курсора НД**

Метод	Выполняемые действия
procedure First;	Устанавливает курсор на первую запись в наборе данных.
procedure Last;	Устанавливает курсор на последнюю запись в наборе данных.
procedure Next;	Перемещает курсор на следующую запись в наборе данных относительно текущей записи.
procedure Prior;	Перемещает курсор на предыдущую запись в наборе данных относительно текущей записи.
function MoveBy(n:integer):Integer;	Перемещает курсор на n записей к концу набора данных (n>0) или к началу набора (n <0.)

Определение начала и конца набора данных

Свойство

property BOF: Boolean;

возвращает True, если курсор установлен на первую запись в наборе данных.

Свойство

property EOF: Boolean;

возвращает True, если курсор установлен на последнюю запись в наборе данных.

Может сложиться впечатление, что первая и последняя записи набора всегда фиксированы, что это физически первая и последняя записи в НД. Это неверно. Во-первых, как уже отмечалось выше, набор данных может содержать часть записей из таблицы БД. Поэтому набор данных — понятие логическое, а не физическое. Для TTable последовательность расположения записей в наборе данных определяется используемым индексом, обуславливающим сортировку. Для TQuery порядок следования записей либо случаен, особенно при использовании в качестве источника более одной таблицы БД, либо упорядочен в порядке перечисления полей, в разделе ORDER BY.

Начиная работать с НД, имеющим одну сортировку записей, мы можем затем переопределить сортировку, и записи «пере-

строятся» в соответствии с новой сортировкой, т.е. логический порядок их следования изменится.

При изучении вопросов навигации по НД следует говорить прежде всего о логическом характере следования записей, поскольку физический характер их расположения в конкретном случае неизвестен.

Навигация по НД вниз

Для выполнения действий, начиная от некоторой стартовой записи и до конца набора данных, используют цикл WHILE not EOF. Стартовая запись может устанавливаться различными способами.

Приведем пример для случая, когда стартовая запись — первая в наборе:

```
with tPeople do begin
  First;
  while not Eof do begin
    // Какие-либо действия
    Next;
  end; // while
end; // with
```

Навигация по НД вверх

Для выполнения действий, начиная от некоторой стартовой записи и до начала набора данных, используют цикл WHILE not BOF**.** Стартовая запись может устанавливаться различными способами. Приведем пример для случая, когда стартовая запись — последняя в наборе:

```
with tPeople do begin
    Last;
    while not Bof do begin
      // Какие-либо действия
      Prior;
    end; // while
  end; // with
```

Спонтанные перемещения по набору данных

Вообще говоря, часто необходимо в зависимости от каких-либо условий «прыгать» по НД взад-вперед. Поэтому распространен вариант одновременного использования Next, Prior и MoveBy в одном программном блоке. При этом важно помнить о том обстоятельстве, что применение метода Edit, когда изменяется значение индексного поля, по которому в настоящий момент ведется сортировка в НД, может переместить запись вниз или вверх.

Поэтому следует всегда придерживаться правила: не изменять значения индексного поля при прохождении набора данных в цикле. Данную проблему можно решить различными способами.

Реакция на изменение курсора набора данных

Событие OnDataChange (компонент DataSource) возникает всякий раз при изменении курсора НД, т.е. при переходе к новой текущей записи. Это событие возникает, когда курсор НД уже находится на новой записи.

Событие происходит и в режимах dslnsert и dsEdit:

– при изменении какого-либо поля;

– при первом перемещении с измененного поля на другое поле.

Два события компонента типа «набор данных» также происходят при переходе к новой записи:

– BeforeScroll: TDataSetNotifyEventl;

Событие наступает перед переходом на другую запись в наборе данных.

– AfterScroll: TDataSetNotifyEventl;

Событие наступает после перехода на другую запись в наборе данных.

Временное отключение визуализации при работе с НД

При выполнении действий с НД, влекущих за собой частое изменен местоположения курсора БД, в визуальном компоненте, показывающем записи (например, TDBGrid) или текущую запись (TDBEdit и др.), будет возникать эффект «прокрутки» записей. Он не всем нравится. Кроме этого, при смене местоположения

курсора БД (т.е. при смене текущей записи НД) необходимо время для отражения произошедших изменений в визуальном компоненте.

Для устранения данной проблемы имеются методы

procedure DisableControls;

procedure EnableControls;

Первый отключает связь с визуальным компонентом, а второй восстанавливает ее. Например, при последовательном переборе записей произведенном таким образом:

```
with tPeople do begin
  DisableControls;
  First;
  while not Eof do begin
    // какие-либо действия
    Next;
  end; // while
  EnableControls;
 end; // with
```

в компоненте TDBGrid не будет видно эффекта прокрутки записей. Наоборот, у пользователя возникнет иллюзия, что курсор БД сразу переустановился с текущей записи набора данных на его последнюю запись.

Свойство набора данных property RecordCount:Integer возвращает текущее число записей в НД.

Свойство набора данных property RecNo:Integer возвращает порядковый номер текущей записи

Внесение изменений в НД

Изменение текущей записи

Чтобы изменить запись в НД, этот НД нужно перевести методом **Edit** из состояния dsBrowse в состояние dsEdit, затем произвести изменение значения одного или нескольких полей записи и использовать метод **Post** для запоминания измененной записи в НД. Post в данном случае при благополучном исходе переводит НД из состояния dsEdit в состояние dsBrowse.

Для отказа от запоминания измененной записи в НД используется метод **Cancel**. Он также переводит НД из состояния dsEdit в состояние dsBrowse.

Метод procedure Edit

Редактирование записи должно быть разрешено (свойство **property ReadOnly: Boolean;** должно быть установлено в False). Помимо этого, могут быть запрещены для корректировки отдельные поля записи (когда свойство ReadOnly соответствующих компонентов TField установлено в True).

Метод Edit может вызываться:

– программно;

– автоматически, когда пользователь в визуальном компоненте, связанном с НД, выполняет определенные действия. Вид этих действий завит от визуального компонента.

Автоматический перевод набора данных в режим редактирования должен быть разрешен свойством AutoEdit соответствующего компонента TDataSourse (значение True).

Например:

```
tPeople.Edit;
tPeople.FieldByName('Last_Name').Value:=Edit1.Text;
tPeople.Post;
```

Добавление новой записи

Чтобы добавить новую запись в НД, нужно вызвать метод **Insert** или **Append** для перевода из состояния dsBrowse в состояние dsInsert. Затем производится присваивание значения одному или нескольким полям записи, после чего выполняется метод **Post** для запоминания новой записи в НД. Post при благополучном исходе переводит НД из состояния dsInsert в состояние dsBrowse.

Для отказа от запоминания новой записи в НД используется метод Cancel. Он также переводит НД из состояния dsInsert в состояние dsBrowse.

Метод procedure Insert

При добавлении записи изменение НД должно быть разрешено (свойство **property ReadOnly: Boolean;** должно быть уста-

новлено в False). Помимо этого, могут быть запрещены для корректировки отдельные поля записи (когда свойство ReadOnly соответствующих компонентов TField установлено в True). В этом случае в них нельзя ввести новые значения.

Метод Insert может вызываться:

– программно;

– автоматически, когда пользователь в визуальном компоненте, связанном с Н Д, предпринимает соответствующие действия. Для перехода в режим dsInsert в компоненте TDBGrid достаточно нажать на клавиатуре клавишу Insert или, находясь на последней записи НД, попытаться перейти на нижнюю, несуществующую запись. То же происходит при нажатии соответствующей кнопки связанного с данным НД компонента TDBNavigator.

Например:

```
tPeople.Insert;
// установка значений полей добавляемой записи
tPeople.Post;
```

Метод procedure Append

Аналогичен методу Insert, но он добавляет запись в конец набора данных, в то время как Insert добавляет ее после текущей записи.

Запоминание изменений — метод procedure Post

Выполнение метода **Post** приводит к запоминанию изменений, сделанных в режиме добавления или изменения записи.

Если НД не находится в режиме dsInsert или dsEdit, то применение **Post** приводит к возбуждению исключительной ситуации.

Вызов Post зависит от способа, которым ранее был вызван метод Insert или Edit:

– программно;

– автоматически.

Post обычно вызывается автоматически, если пользователь предпринимает соответствующие действия, направленные на запоминание измененной записи в НД. Вид этих действий зависит

от визуального компонента, связанного с НД. Например, для компонента TDBGrid, связанного с набором данных, это — переход к другой записи. Для НД, управляемого компонентом TDBNavigator; это — нажатие соответствующей экранной клавиши. Реже изменения в наборе данных, автоматически переведенном в режим редактирования, запоминаются путем программного вызова метода Post.

Метод Post, независимо от того, вызывается он программно или автоматически, может завершиться неудачно. Причиной этого могут послужить неверные значения в соответствующих полях записи. Например:

– поле обязательного заполнения (свойство Required:=True у соответствующего компонента TField) содержит пустое значение;

– для таблицы БД, у которой определен уникальный ключ, возникла ситуация дублирования ключа (Key Violation), то есть ключевое поле (группа полей) данной записи содержит значение, которое уже хранится в поле (группе полей) в другой записи;

– не соответствие типов введенных значений с указанными типами полей;

– обработчики событий типа OnValidate (компонент TField) и BeforePostRecord обнаружили, что какое-либо поле содержит неверное значение, не удовлетворяющее некоторым условиям. В этом случае, программно возбуждается исключительная ситуация, которая подавляет выполнение Post.

В лучшем случае при возникновении препятствий для выполнения Post запись переводится в состояние, в котором НД находился до выполнения метода (dslnsert или dsEdit).

Отмена сделанных изменений — метод procedure Cancel

Метод **Cancel** отменяет все изменения, сделанные в записи. Если НД находился в режиме добавления новой записи, запись в НД не добавляется. Если НД находился в режиме изменения записи, изменявшаяся запись в НД не записывается, и данные в ней остаются в том состоянии, в котором они находились до перехода в режим dsEdit. Сам НД переводится в режим dsBrowse.

Вызов Cancel зависит от способа, которым ранее был вызван метод Insert или Edit:

– программно;

– автоматически.

Cancel вызывается автоматически, если пользователь предпримет соответствующие действия, направленные на запоминание измененной записи в НД. Вид этих действий зависит от визуального компонента, связанного с НД. Например, для компонента TDBGrid, связанного с набором данных, это — нажатие клавиши Esc. Для НД, управляемого компонентом TDBNavigator, это — нажатие на соответствующей экранной клавиши компонента TDBNavigator.

Оценка изменения записи

Часто бывает необходимо знать, вносились ли в запись изменения в режимах dsInsert или dsEdit. Это актуально в тех случаях, когда внесение изменений в записи зависит от каких-либо условий, которые могут наступать или не наступать в разные моменты работы приложения.

Свойство НД

property Modified: Boolean;

автоматически устанавливается в True, если значение какого-либо поля записи НД было изменено в режимах dslnsert или dsEdit. Методы Post и Cancel переводят свойство в состояние False.

Например:

```
tPeople.Edit;
// действия
if tPeople.Modified then tPeople.Post else tPeople.Cancel;
```

Удаление записи

Удаление текущей записи в наборе данных реализуется методом **procedure Delete**. Например:

tPeople.Delete;

Удаление записи может производиться:

– программно;

– автоматически, если это предусмотрено в том или ином компоненте.

В компоненте TDBGrid нажатие комбинации клавиш Ctrl + Del влечет за собой удаление записи, которое, в соответствии с опциями настроики TDBGrid, может выполняться как с запросом подтверждения, так и без него.

Необходимо помнить об одной важной особенности. Записи в различных СУБД могут удаляться 2 способами:

– пометка записи в таблице БД как удаленной. Сама запись физически не удаляется из таблицы БД. В зависимости от СУБД новые записи могут записываться место помеченных как «удаленные» или в конец таблицы БД. В последнем случае такие таблицы БД могут «разбухать» до больших размеров, поэтому время от времени для них проводят операцию сжатия, при которой помеченные как удаленные записи физически уничтожаются, а остальные записи «сдвигаются» вверх, заполняя образовавшиеся пустоты в таблице БД;

– немедленное удаление записей из таблицы БД, вследствие чего последующие записи «сдвигаются» вверх, заполняя образовавшиеся в таблице БД пустоты.

В Delphi при работе с НД реализован второй метод. После удаления записи все оставшиеся записи «сдвигаются» наверх. При удалении одной записи это может быть несущественным, однако, если нужно удалить несколько записей это способно внести осложнения.

Например, пусть требуется удалить все записи из tPeople. Можно было бы предположить, что данную потребность можно реализовать следующим программным кодом:

```
with tPeople do begin
First;
  while not Eof do begin
    Delete;
    Next; //ошибка
  end; // while
end; // with
```

Однако в действительности этот код приведет к удалению примерно половины записей в tPeople. Причина этого лежит в

том, что когда мы удаляем запись (например, № 3), последующие записи автоматически перемещаются вверх, и поэтому запись, бывшая до удаления следующей (№ 4), становится текущей (№3). После выполнения метода Next осуществляется переход к записи №4. Таким образом, записи удаляются через одну.

Удалив ненужный вызов Next, мы сотрем все записи:

```
with tPeople do begin
  First;
  while not (RecordCount =0) do Delete;
end; // with
```

Чаще всего нужно удалять не все записи НД, а часть записей, удовлетворяющих некоторому условию.

Кроме того, желательно ожидать подтверждения для удаления записи. Например, будем удалять текущую запись по нажатию кнопки (Button1) с сообщением подтвердить удаление записи:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Application.MessageBox('Удалить текущую запись ?', 'Предупреждение', MB_YESNO) = IDYES
  then tPeople.Delete;
end;
```

Обработка ошибок смены состояний набора данных

В случае неудачи при выполнении методов Insert, Edit, Delete и Post обработку ошибки можно реализовать в соответствующих обработчиках событий OnEditError (ошибки при выполнении Insert и Edit), OnDeleteError, (ошибки при выполнении Delete) и OnPostError (ошибки при выполнении Post).

property OnEditError: TDataSetErrorEven;
property OnDeleteError: TDataSetErrorEven;
property OnPostError: TDataSetErrorEven;

где

TDataSetErrorEvent = procedure(DataSet: TDataSet; E: EDatabaseError;
var Action: TDataAction) of object;

TDataAction = (daFail, daAbort, daRetry);

назначение параметров:

DataSet — указатель на компонент, в котором произошла ошибка;

E — ссылка на объект-исключение;

Action — действие:

daFail — выполнение метода, вызвавшего ошибку, отменяется, выводит сообщение об ошибке;

daAbort — выполнение метода, вызвавшего ошибку, отменяется, сообщение об ошибке не выводится;

daRetry — метод, вызвавший ошибку, после выхода из обработчика выполняется заново; при этом в теле обработчика должна быть скорректирована при ошибке, иначе произойдет зацикливание программы.

Практическая работа

В приложении для работы с базой данных Отдела кадров (автоматизация части деятельности отдела кадров) предусмотреть работу с таблицами БД по принципу Справочников и Операционной таблицы.

В программе предусмотреть:

– меню;

– панель инструментов;

– отдельные экранные формы для работы со справочниками и операционной таблицы;

– обеспечить редактирование, добавление, удаление данных (в том числе через программную реализацию).

Особенности работы с мемо-полем и полем с графическим изображением

Работа с текстом

Элемент управления TDBMemo является потомком TMemo и является простым текстовым редактором, адаптированным для работы с текстовым представлением BLOB поля набора данных.

Поместите экземпляр на форму, назначьте ему источник данных (TDBMemo.DataSource) и имя поля (TDBMemo.DataField), в

котором предполагается хранить текст. Возможности TDBMemo при работе с текстом аналогичны его родителю TMemo — выделение и редактирование текста, работа с буфером обмена, поддержка TEditActions. Следует обратить внимание на методы TMemo.Lines.LoadFromFile() и TMemo.Lines.SaveToFile(), которые позволяют загружать из файла и сохранять в файл содержимое редактора (и соответствующего поля). Особенность — перед вызовом TMemo.Lines.LoadFromFile() нужно убедиться, что набор данных находиться в режиме редактирования.

Работа с графическим объектом

Для сохранения в таблице графического изображения Вам может пригодиться элемент управления TDBImage. Поместите его на форму, назначьте ему источник данных и имя поля, в котором предполагается хранить картинку. Естественно тип поля должен быть соответствующим, обычно это тип BLOB.

Для простого отображения картинки хранящейся в таблице этих действий достаточно. Картинка будем изменяться, когда вы будете скролировать набор данных. Для изменения картинки можно использовать два способа.

Первый заключен в использовании буфера обмена (clipboard). TDBImage имеет специальные методы для копирования картинки, которая сейчас содержится в поле назначенного набора данных (и отображается control'ом) в системный буфер обмена Windows — это методы TDBImage.CopyToClipboard и TDBImage.CutToClipboard. Второй метод отличается тем, что после сохранения картинки в буфере содержимое TDBimage и соответствующего поля стирается. Картинка сохраняется в формате Bitmap и может быть вставлена из буфера во многие Windows программы (Word, Image Editor. Paint, Excel). Также есть метод для вставки картинки, которая сейчас содержится в системном буфере в TDBImage и поле назначенное ему. Это метод PasteFromClipboard.

Все эти методы можно вызывать программно — например, назначив событие на кнопку или TAction. Также за ними уже зарезервированы комбинации клавиш, которые вы может использовать без какого либо дополнительного кодироваия (табл. 8).

Таблица 8 — **Комбинации клавиш**

Метод	Клавиши	Действие
CopyToClipboard	Ctrl+C Ctrl+Insert	Скопировать картинку в буфер обмена.
CutToClipboard	Ctrl+X Shift+Delete	«Вырезать» картинку в буфер обмена.
PasteFromClipboard	Ctrl+V Shift+Insert	Скопировать картинку из буфера обмена.

Второй способ использование методов свойства TDBImage.Picture. Используйте метод TDBImage.Picture.SaveToFile (FileName:string) для сохранения картинки в файле формата *.bmp. И метод TDBImage.Picture.LoadFromFile(FileName:string) для загрузки файла в TDBImage и текущее поле. Один нюанс при использовании последнего метода: набор данных должен быть в режиме редактирования. Проверяйте это условие перед вызовом LoadFromFile и если это не так — переводите набор данных в режим редактирования принудительно: например

TDBImage.DataSource.DataSet.Edit.

Все вышесказанное в одинаковой степени касается и простого элемента управления TImage, т.к. TDBImage является его потомком в иерархии классов VCL/CLX. Но при использовании простого TImage Вам придется самостоятельно отслеживать изменение состояния набора данных и нужного поля.

ЛАБОРАТОРНАЯ РАБОТА № 5
РАБОТА С SQL

Задание

1. Изучение компонента TQuery.

2. Создание структурированных запросов SQL.

3. Разработка формы с реализацией различных запросов к БД отдела кадров.

Компонент TQuery

Компонент TQuery предназначен для:

– работы с НД, источником данных для которого могут служить записи как одной, так и нескольких таблиц БД (TQuery, возвращающий набор данных);

– выполнения запросов к БД, не возвращающих наборов данных (добавление, изменение, удаление записей в таблицах БД и др.).

Основные отличия компонента данных TQuery, возвращающего набор данных, от выполняющего сходные функции компонента TTable:

– НД, возвращаемый TQuery, может быть составлен из записей нескольких таблиц (над которыми выполнена операция объединения);

– в общем случае НД, возвращаемый TQuery, даже если источником этого НД служит одна таблица БД, предполагает обращение к подмножеству записей и столбцов (полей), в то время как TTable ориентирован на работу со всеми записями и полями и для того, чтобы работать в нем с подмножеством строк и полей, необходимо предпринять дополнительные действия (фильтрацию записей, ограничение состава полей в редакторе полей).

Результирующий НД компонента TQuery формируется путем выполнения запроса к БД на языке SQL (Structured Query Languague, язык структурированных запросов). Такой запрос использует SQL-оператор SELECT. Текст любого запроса хранится в свойстве SQL компонента TQuery.

Запросы, выполняемые компонентом TQuery — независимо от того, возвращают они набор данных или просто производят

какие-либо действия в БД, — могут быть статическими и динамическими.

Статический запрос характерен тем, что описывающий его SQL-оператор не изменяется в процессе выполнения приложения.

SQL-оператор **динамического запроса** может частично изменяться в процессе выполнения приложения. В этом случае изменяемые части SQL-запроса оформляют в качестве параметров, значения которых могут многократно изменяться в процессе выполнения приложения. Таким образом, можно использовать один компонент TQuery для выполнения множества разнесенных во времени запросов к БД, различающихся по значению параметров. Заметим, что состав параметров также может меняться во время выполнения. Это более характерно для формируемых запросов — разновидности динамических запросов.

Формируемые запросы — такие запросы, текст SQL-оператора которых формируется программно в процессе выполнения приложения. Действия по формированию такого запроса состоят в очистке предыдущего содержимого свойства SQL и программного занесения в это свойство нового текста SQL-запроса (вид которого зависит от текущей ситуации и определяется рядом условий), а также в последующем его выполнении. Такая задача является тривиальной, поскольку свойство SQL имеет тип TString, то есть являет собой экземпляр динамического строкового списка. Таким образом, один компонент TQuery используется для выполнения таких различных запросов, как, например, SELECT и INSERT.

Характерной особенностью компонента TQuery является использование в нём специального языка для работы с реляционными БД — SQL(Structured Query Language — язык структурированных запросов). С помощью этого языка программа составляется SQL-запрос, который TQuery передаёт машине баз данных BDE. Последняя имеет встроенный интерпретатор SQL, позволяющий ей выполнять описанные в запросе действия.

Если запрос требует получения из БД нужных сведений, сформированные с помощью BDE данные помещаются в локальную таблицу в виде переменного файла в каталоге запуска программы и TQuery становится владельцем этой таблицы. Данные из временной таблицы через компонент посредник TDataSource

передаются визуальным компонентам и отображаются в них точно так же, как если бы они были получены компонентом TTable. Однако, в отличие от TTable, пользователь не может их изменять, т.к. они представляют собой лишь копию реальных данных. Для изменения хранящейся в БД информации формируются специальные SQL операторы (INSERT, UPDATE, DELETE), которые TQuery передаёт BDE. В этом случае BDE не формирует новые и никак не использует созданные ранее временные таблицы, но лишь интерпретирует запрос и уведомляет программу о том, насколько успешно произошло его выполнение.

Подводя итог, нужно сказать, что при работе с локальными или файл-серверными БД скорость доступа к данным у TQuery в общем случае меньше, чем у TTable, так как для своей работы TQuery создаёт временные таблицы. С другой стороны, мощные возможности SQL позволяют с помощью TQuery получать НД, которые невозможно получить с помощью TTable (например, объединение в одном НД данных из нескольких таблиц БД).

При работе с серверными БД TTable теряет всяческие преимущества, так как в этом случае он создаёт временную таблицу, являющуюся локальной копией всей серверной ТБД, а уже затем формирует из неё нужный НД. Затраты времени на создание больших локальных таблиц и значительно более скромные возможности TTable в отношении получения сложных НД практически исключают его использование в клиент-серверных приложениях. Здесь в основном используют TQuery.

SQL оператор запросов SELECT

Оператор Select — наиболее часто используемый оператор SQL. Позволяет производить выборки данных из ТБД и преобразовывать к нужному виду полученные результаты. С его помощью можно реализовать весьма сложные условия выбора данных из различных таблиц.

В самом общем виде он имеет формат:

Что берём:

SELECT [DISTINCT | ALL] {* | <значение1> [, <значение2>...]}

Откуда:

FROM <таблица1> [, <таблица2>…]

Каким условиям отвечает:

[WHERE <условие отбора>]

Группировка по колонкам:

[GROUP BY столбец [COLLATE collation]]

[, столбец1 [COLLATE collation]…]

Группировка по строкам:

[HAVING <условия поиска>]

Порядок вывода:

[ORDER BY <список_столбцов>]

Всё что находится в [] скобках не является обязательным, т.е. для создания простейшего запроса выводящего всю таблицу, или выбранные из неё поля достаточно первых двух строк.

После ключевого слова **Select** приводится список значений, каждое из которых определяет столбец возвращаемого оператором результирующего набора данных. Звёздочка «*» указывает, что в результат запроса надо включить все столбцы той или иной таблицы.

После **From** указывается список ТБД, из которых будет происходить выборка данных.

Использование предложения WHERE

В набор данных, возвращаемый оператором Select, будут включаться только те записи, которые удовлетворяют условию поиска, указанному после Where.

Варианты формирования условий поиска:

1. Сравнение значения столбца с константой или столбцом другой таблицы.

<имя столбца таблицы1> < оператор> < константа или имя столбца таблицы2 >

= равно | !< не меньше (больше или равно)
<меньше | !> не больше (меньше или равно)
> больше | <> не равно
<= меньше или равно | != не равно
>= больше или равно |

WHERE Last_Name = ‘Петров’

2. Использование логических выражений.

Строятся при помощи логических выражений AND, OR и NOT. Важно: операции отношения в них имеют меньший приоритет, чем логические операции, что избавляет от необходимости расстановки многочисленных скобок.

WHERE Last_Name = ‘Петров’ AND First_Name = ‘Иван’

3. Сравнение столбца с результатом вычисления.

<выражение> <оператор> <столбец> или <столбец> <оператор> <выражение>

Однако этот способ чаще применяется при использовании механизма вложенных подзапросов (вложенных операторов Select), речь о которых пойдёт ниже.

4. Использование BETWEEN.

В условии поиска можно указать, что некоторое значение (столбец или вычисленное значение) должно находиться в интервале между *значением 1* и *значением 2*

<значение> [NOT] BETWEEN < значение 1> AND < значение 2>

WHERE Oklad BETWEEN 2000 AND 3000

5. Использование IN.

Если нужно, чтобы значение какого либо столбца (или результат вычисление некоторого выражения) совпадало с одним из дискретных значений, в условии поиска указывается предложение

< значение > [NOT] IN (<значение 1> [, < значение 2>…])

WHERE Last_Name IN (‘Петров’, ‘Иванов’, ‘Сидоров’)

6. Использование функции UPPER.

UPPER (<значение>)

Преобразует все буквы аргумента <значение> (содержимого столбца, результата вычисления выражения) к заглавным. Обычно эта функция используется в условиях поиска, когда необходимо игнорировать возможную разницу в высоте букв. Функция

UPPER может фигурировать как в списке столбцов результирующего набора данных (после SELECT), так и в условии поиска в предложении WHERE.

WHERE UPPER(Last_Name) = ‘петров’

7. Использование LIKE.

Предложение LIKE определяет шаблоны сравнения строковых значений. Если необходимо, чтобы сравниваемое значение (значение столбца или результат вычисления строкового выражения) удовлетворяло шаблону, в условии поиска необходимо указать
<значение> [NOT] LIKE <шаблон> [ESCAPE <подшаблон>]

В шаблоне используются специальные символы — «%» и «_». Символ «%» означает, что на его месте может быть строка любой длины, а символ «_» используется для указания любого единичного символа. Например, LIKE «%USD» требуется, чтобы строковое значение оканчивалось символом «USD», независимо от того, какие символы (и сколько их) предшествует этому окончанию. LIKE «__94» указывает, что сравниваемое значение может содержать четыре символа, из которых первые два — любые, а два последних — «94». Ещё в предложении LIKE символы «%» или «_» должны использоваться в шаблоне подобия как обычные символы (без учёта их специальных функций), в запрос включается предложение ESCAPE <символ>. ESCAPE определяет символ <символ>, появление которого в шаблоне отменяет специальные функции следующего за ним символа: WHERE STOLBEZ LIKE «%!%» ESCAPE «%» (строки STOLBEZ должны содержать два символа и заканчиваться символом «%»).

Select * from people where Last_Name like «И%»

8. Использование функции CAST.

Иногда возникает потребность трактовать значение одного типа, как значение другого типа. Например, использовать числовое значение как символьную строку или наоборот. В этом случае применяют функцию

CAST (<значение> AS <тип данных>)

Функция CAST делает копию *значения,* преобразуя его к указанному *типу данных*. При этом не следует забывать о множестве типов данных, в которое может быть преобразовано значение:

Тип данных	Можно привести к типам
NUMERIC	CHARACTER, DATE
CHARACTER	NUMERIC, DATE
DATE	CHARACTER, NUMERIC

Использование псевдонимов таблиц

Каждой таблице используемой в запросе можно присвоить краткое имя (например, просто букву), а затем обращаться по псевдониму.

```
SELECT ....
FROM <таблица1 псевдоним1> [, <таблица2 псевдоним2>…]
WHERE ....
```

Внутреннее соединение таблиц

Необходимо когда вы используете в запросе более чем одну связанную таблицу, т.е. для правильной работы запроса необходимо связать таблицы. Эта процедура похожа на создание реляционных отношений.

<Имя столбца 1 таблицы> = <Имя столбца 2 таблицы>

Это условие указывается в предложении WHERE и далее через логический оператор AND указываются условия запроса.

Например: Кто работает бухгалтером?

```
select p.Last_name, j.Job_Name
from people p, job j , contract c
where j.Job_Name='Бухгалтер' and j.Id_Job=c.Id_Job and c.Id_People = p.Id_People
```

Предложение ORDER BY — определение сортировки

Результирующий НД можно отсортировать с помощью предложения:

ORDER BY <список_столбцов> [asc или desc]

Список столбцов содержит имена столбцов, по которым будет производиться сортировка. Если указаны два или более столбцов, первый столбец будет использован для глобальной сортировки, второй столбец — для сортировки внутри группы, определяемой единым значением первого столбца, и т.д. При указании asc (принято по умолчанию) сортировка происходит по возрастанию, desc — сортировка производится по убыванию.

Устранение повторяющихся значений

Часто в результирующий НД необходимо включать не все записи с одинаковым значением какого-либо столбца (комбинаций столбцов), а только одну из них. В этом случае после SELECT указывают ключевое слово DISTINCT, с помощью которого устраняются повторяющиеся записи:

SELECT [DISTINCT | ALL] {* | <значение1> [, <значение2>...]}
FROM <таблица1> [, <таблица2>…]

Повторяющимися считаются записи, содержащие идентичные значения во всех столбцах результирующего НД. Наоборот, если в результирующий запрос нужно включить все записи, после SELECT указывают слово ALL (принято по умолчанию во многих СУБД).

Расчёт значений вычисляемых столбцов

Для расчёта значений вычисляемых столбцов результирующего НД используются арифметические значения. При этом в списке возвращаемых столбцов после SELECT вместо имени вычисляемого столбца указывается выражение:

SELECT [DISTINCT | ALL] {* | <столбец1> [, <выражение1>...]}
FROM <таблица1> [, <таблица2>…]

Если вы хотите задать новое временное имя столбца

Это особенно актуально для вычисляемых столбцов и столбцов при слиянии со строкой, после имени столбца или выражения нужно указать ключевое слово AS и новое имя столбца:

SELECT ... {* | <значение1> [, <выражение1 [AS <имя столбца>]>...]}

SELECT Last_Name AS Фамилия

Агрегатные функции

Агрегатные функции предназначены для вычисления итоговых значений операций над всеми записями НД.

К агрегирующим относятся следующие функции:

- COUNT (<выражение>) — подсчитывает число вхождений значения выражения во все записи результирующего НД;
- SUM (<выражение>) — суммирует значения выражения;
- AVG (<выражение>) — находит среднее значение;
- MAX (<выражение>) — определяет максимальное значение;
- MIN (<выражение>) — определяет минимальное значение.

Если из группы одинаковых записей нужно учитывать только одну, перед выражением в скобках включают слово DISTINCT:

COUNT (DISTINCT <выражение>)

Чаще всего в качестве выражения выступают имена столбцов, при этом выражение может вычисляться и по значениям нескольких таблиц.

Группировка записей

Иногда требуется получить агрегированные значения (минимум, максимум, среднее) не по всему результирующему НД, а по каждой из входящих в него групп записей, характеризующихся одинаковым значением какого-либо столбца. Например, выдать общее количество сотрудников в каждом отделе. В этом случае в оператор SELECT вводится предложение

GROUP BY столбец [,столбец1 …]

При этом необходимо, чтобы один из столбцов результирующего НД был представлен агрегатной функцией.

Предложение HAVING — наложение ограничений на группировку записей

Если нужно в результирующем НД выдавать агрегацию не по всем группам, а только по тем из них, которые отвечают некоторому условию, после предложения GROUP BY указывают предложение:

HAVING <агрегатная функция> <отношение> <значение>

где

- агрегатная функция — одна из функций MIN, MAX, AVG и SUM;
- отношение — одна из операций отношения =, <>, <, <=, >, >=;
- значение — константа, результат вычисления выражения или единичное значение, возвращаемое вложенным оператором SELECT.

Таким образом, после HAVING указываются условия, которые отличаются от условий, определяемых в предложении WHERE, одним важным обстоятельством: в HAVING обязательно должна быть указана одна из агрегатных функций, в то время как в предложении WHERE такие функции указывать нельзя.

Использование подзапросов

Часто невозможно решить поставленную задачу путём использования единственного запроса. Например, в тех случаях, когда при использовании условия поиска <сравниваемое значение> <оператор> <значение, с которым сравнивать> в предложении WHERE параметр <значение, с которым сравнивать> заранее не определён и должен вычисляться в момент выполнения оператора SELECT или представляет собой не одно, а несколько значений. В такого рода случаях используется подзапросы (вложенные запросы). Оператор SELECT с подзапросом имеет такой вид:

SELECT …
FROM …
WHERE <сравниваемое значение> <оператор> (SELECT …)

Синтаксис вложенного запроса ни чем не отличается от синтаксиса основного запроса и, следовательно, вложенный запрос может в свою очередь содержать подзапрос. Заметим, что SQL — синтаксис требует заключать подзапрос в круглые скобки.

Замечание: распространённой ошибкой является использование вложенного оператора SELECT, который вместо единичного значения способен возвращать список значений.

Использование подзапросов, возвращающих множество значений

- **Использование ALL, SOME.**

Если в условии поиска необходимо указать, что сравниваемое значение (значение столбца, результат вычисления выражения) должно находиться в определённых отношениях со всеми или некоторыми значениями из множества значений, возвращаемых подзапросом, применяют предложение:

<сравниваемое значение> {[NOT] <оператор> | ALL | SOME | ANY}
(<подзапрос>)

Отношение сравниваемого значения и значений, возвращаемых подзапросом, устанавливаются словами ALL и SOME (ANY):

- ALL — указывает, что условие поиска будет истинно только тогда, когда сравниваемое значение находится в нужном отношении со всеми значениями, возвращаемыми подзапросом;
- SOME (вместо него можно указать ANY) — условие поиска истинно только тогда, когда сравниваемое значение находится в нужном отношении хотя бы с одним значением, возвращаемым подзапросом.

Использование HAVING и агрегатных функций для вложенных подзапросов

Если в условиях поиска для вложенного запроса нужно указать агрегатную функцию, используется предложение NAVING.

UNION — объединение результатов выполнения нескольких операторов SELECT

Иногда бывает полезным объединять два или более результирующих НД, возвращаемых в результате выполнения операторов SELECT. Такое объединение производится при помощи оператора UNION. Результирующие НД должны иметь одинаковую структуру, то есть одинаковый состав возвращаемых столбцов. Если в результирующих НД имеется одна и та же запись, в свободном НД она не дублируется.

Например:

```
SELECT * FROM People
WHERE Last_Name Like 'П%'
UNION
SELECT * FROM People
WHERE Last_Name Like 'М%'
```

Использование IS NULL

Если требуется выдать все записи, в которых некоторый столбец (или результат вычисления выражения) имеет значение NULL (т.е. не имеет никакого значения), достаточно в условии поиска указать предложение:

<значение> IS [NOT] NULL

Использование операции сцепления строк

Операция + соединяет два строковых значения, которые могут быть представлены выражениями:

<строковое выражение1> + <строковое выражение2>

Эту операцию можно использовать как после слова SELECT для указания возвращаемых значений, так и в предложении WHERE.

```
SELECT Last_Name + ' '+ First_Name +' ' + Second_Name as ФИО
FROM People
WHERE Last_Name = 'Макаров'
```

3 Основные свойства компонента TQuery

В таблице 9 приведем основные свойства компонента TQuery.

Таблица 9 — **Основные свойства компонента TQuery**

Свойства TQuery	Назначение
Property Constrained: Boolean;	Если содержит True в «Живом» НД на записи, вводимые или изменяемые пользователем, накладываются ограничения секции WHERE и оператора SELECT.
Property DataSource: TDataSource;	Содержит ссылку на компонент TDataSource, используемый для формирования параметрического запроса.
Property Local: Boolean;	Содержит True, если TQuery работает с локальной или файл-серверной БД.
Property ParamCheck: Boolean;	Если содержит True, список параметров будет автоматически обновляться при изменение запроса на этапе прогона программы.
Property Params[Index: Word]: TParams;	Содержит массив объектов-параметров класса TParams.
Property Prepared: Boolean;	Содержит True, если запрос был подготовлен к выполнению методом Prepare.
Property RequestLive: Boolean;	Содержит True, если TQuery должен возвращать «Живой» НД.
Property RowsAffected: Integer;	Содержит количество записей, которые будут сменены или удалены в результате выполнения запроса.
Property SQL: TStrings;	Содержит текст SQL — запроса.
Property Text: PChaf;	Содержит текст SQL — запроса, который был в действительности передан BDE.
Property UniDirectional: Boolean;	Если содержит True, курсор НД может перемещаться только вперёд. Такие НД требуют меньше памяти и быстрее обрабатываются.

Соединение компонента TQuery с базой данных

Для случаев работы с локальными и удаленными БД имеются некоторые различия в способе соединения компонента TQuery с базой данных, для которой и будет выполняться SQL-оператор из свойства SQL этого компонента. Свойство

property Database: TDatabase;

возвращает указатель на компонент TDatabase, выполняющий соединение данного TQuery с базой данных. Если компонент TDatabase явно в приложении не создан (что характерно при работе с локальными БД), на период сеанса автоматически создается временный компонент TDatabase. Свойство

property DatabaseName: TFileName;

позволяет указать, с какой БД будет работать компонент TQuery.

При работе с локальными БД в свойстве DatabaseName указывается:

– псевдоним БД, ранее определенный, например, при помощи утилиты BDE Administrator;

– переопределенный псевдоним из свойства DatabaseName явно определенного компонента TDatabase, если он используется;

– путь на диске к конкретному каталогу (если свойство DatabaseName хранит пустое значение, подразумевается, что указан текущий каталог).

Если свойство DatabaseName хранит пустое значение (подразумевается путь в текущий каталог) или явно указан путь на диске к вполне конкретному каталогу, БД будут искаться именно там (для случая работы с локальными СУБД).

Например, при запросе «выбрать все записи из таблицы People» имя таблицы в операторе SELECT можно указать следующим образом:

– если свойство DatabaseName хранит пустое значение (подразумевается путь к текущему каталогу)

SELECT * FROM «People.DB»

– или, если в установках BDE указано, что в случае отсутствия расширения для файла локальной таблицы по умолчанию берутся таблицы Paradox, то тогда просто записывается:

SELECT * FROM People

– если свойство DatabaseName хранит пустое значение (подразумевается путь к конкретному каталогу), этот каталог можно указать в составе имени файла таблицы БД:

SELECT * FROM «C:\Temp\BD\PEOPLE.DB»

где «C:\Temp\DB\» указывает конкретный каталог, в котором следует искать файл «PEOPLE.DB»;

– если свойство DatabaseName хранит псевдоним БД, переустановленный псевдоним БД или путь к конкретному каталогу

SELECT * FROM PEOPLE

Соединение компонента TQuery и визуальных компонентов для работы с данными

Соединение компонента TQuery с визуальными компонентами происходит через промежуточный компонент TDataSource. Для этого в компоненте TDataSource в свойстве DataSet необходимо указать имя компонента TQuery. В визуальных компонентах (TDBGrid, TDBEdit и т.д.) в свойстве DataSource указывается имя компонента TDataSource и, если необходимо, в свойстве DataField выбирается имя интересующего поля.

Выполнение статических запросов

Для формирования статического запроса необходимо:

1. Выбрать для существующего компонента TQuery в инспекторе объектов свойство SQL и нажать кнопку в правой части строки.

2. В появившемся окне текстового редактора набрать текст SQL-запроса (рис. 3.1).

3. Установить свойство Active компонента TQuery в True, если НД должен быть открыт в момент начала работы приложения или оставить свойство Active в состоянии False, если открытие НД будет производиться в программе в некоторый момент работы приложения.

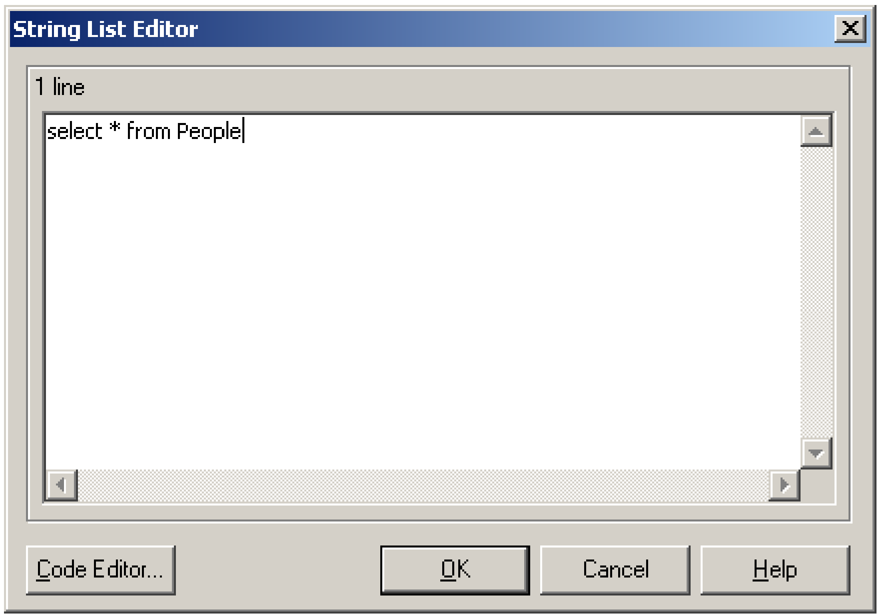


Рис. 10 — Текст SQL-запроса в текстовом редакторе свойства SQL

Условия выборки записей, единожды реализованные по статическому запросу, изменить нельзя, поскольку текст SQL-оператора данного запроса в программе не изменяется. Например, в компоненте TQuery, использующем оператор SELECT, показанный выше, закрытие и повторное открытие компонента приведет к выдаче результирующего НД, в который будут включены все записи из таблицы PEOPLE, присутствующие в данной таблице на момент повторного открытия компонента TQuery.

Формирование текста SQL-оператора SELECT может осуществляться не вручную, а при помощи встроенного в Delphi средства Visual Query Builder, действующего по принципу QBE (Query By Example, запрос по образцу). Для его запуска нужно сделать компонент TQuery текущим, нажать правую кнопку мыши и выбрать режим Query Builder. Заметим, что в TQuery на момент запуска Visual Query Builder должно быть установлено значение свойства DatabaseName.

Вид оператора SELECT для данного запроса (рис. 11):

```
SELECT * FROM PEOPLE WHERE Last_Name='Петров'
```

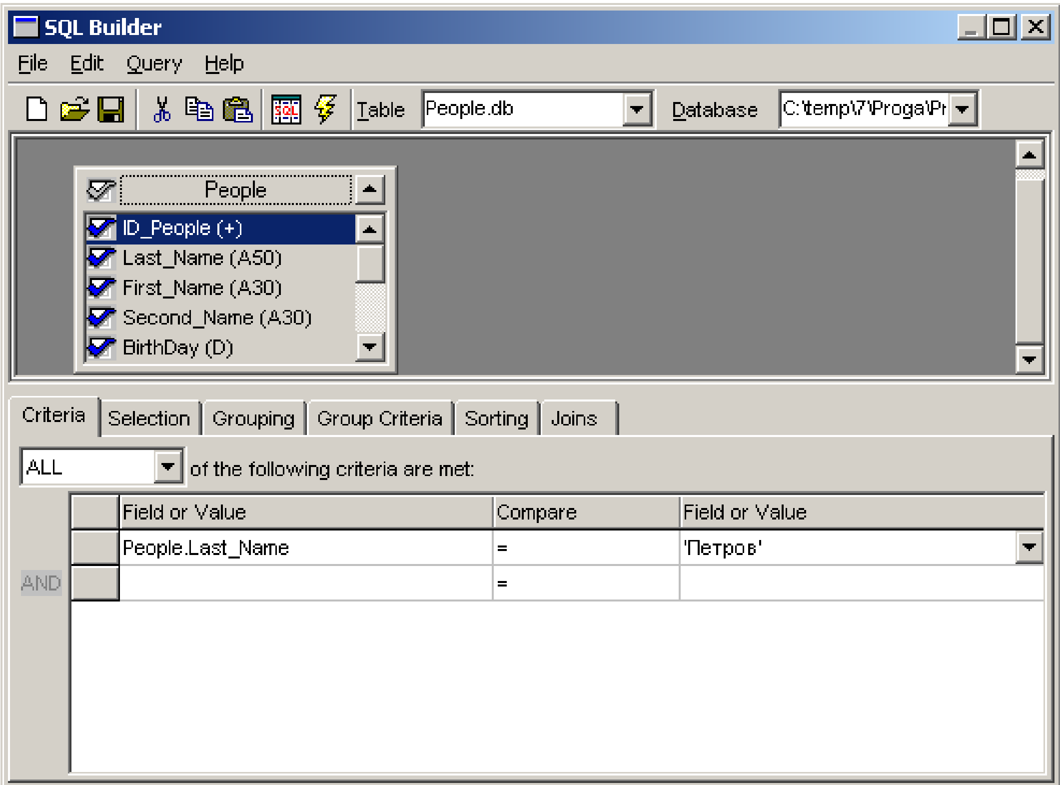


Рис. 11 — Окно Visual Query Builder для выбора таблиц, участвующих в запросе

Для запроса: в каком отделе работает Петров получаем (рис. 12).

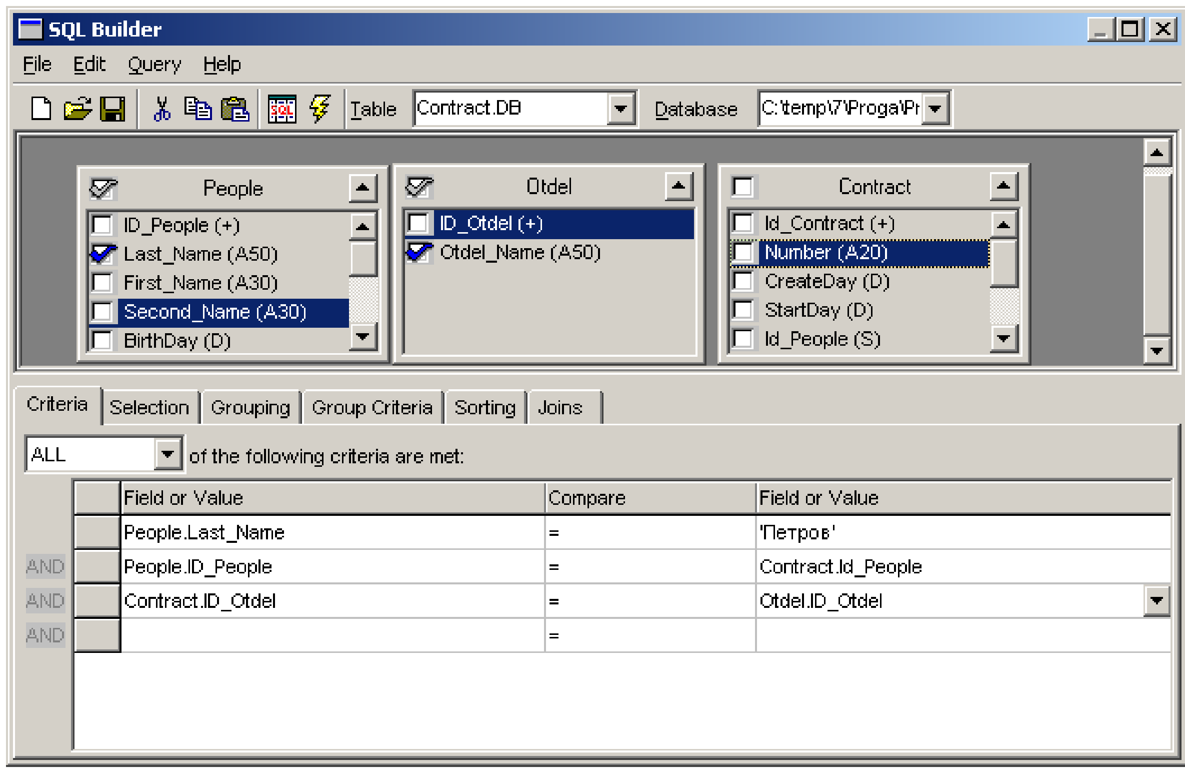


Рис. 12 — Окно Visual Query Builder для запроса в каком отделе работает Петров

Сформированный запрос может быть просмотрен (рис. 13) и выполнен (рис. 14). После закрытия Visual Query Builder текст сформированного запроса помещается в свойство SQL данного компонента TQuery.

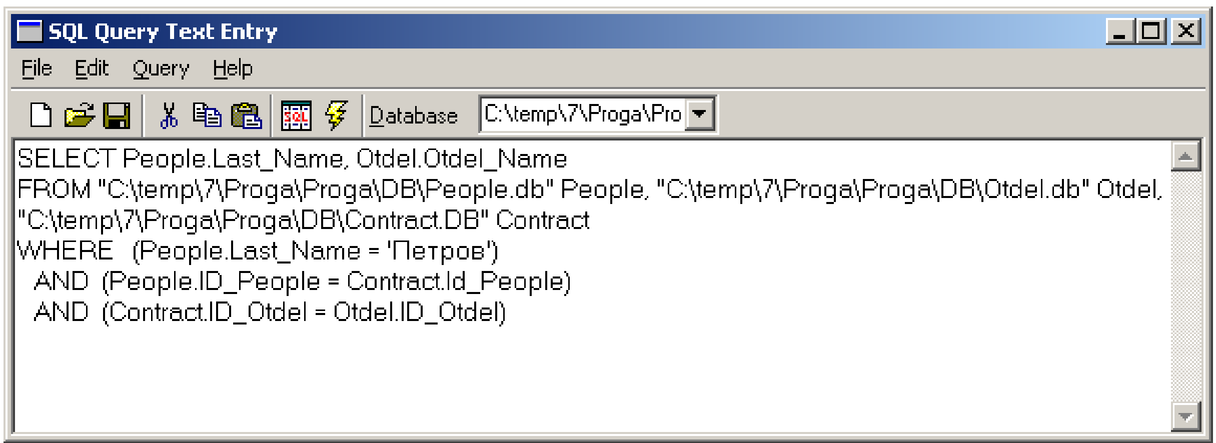


Рис. 13 — Окно Visual Query Builder в режиме просмотра формированного запроса: в каком отделе работает Петров

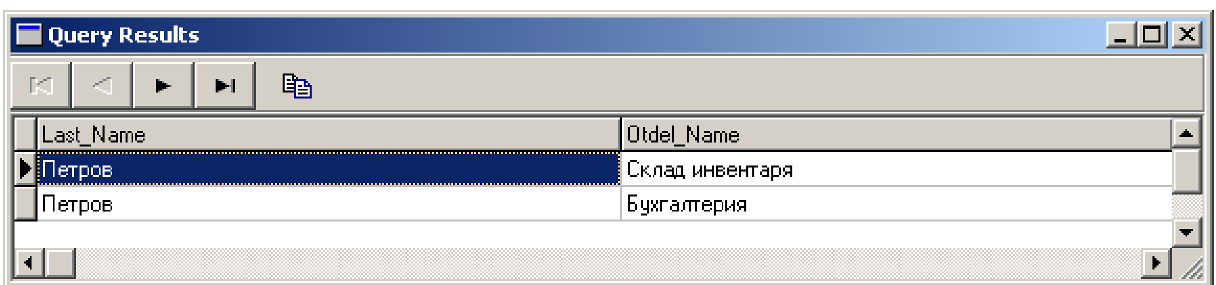


Рис. 14 — Окно Visual Query Builder результата выполнения запроса в каком отделе работает Петров

Visual Query Builder обычно используется для создания черновых вариантов оператора SELECT. Впоследствии текст запроса трансформируется разработчиком к нужному виду.

Кроме того, формирование текста SQL-оператора SELECT может осуществляться и программно.

Методы открытия и закрытия компонента TQuery

Компонент TQuery может возвращать НД (если компонент использует оператор SELECT, то есть осуществляет выборку из одной или более таблиц БД) и выполнять действие над одной или более таблицей БД (SQL-операторы INSERT, UPDATE, DELETE).

В случае использования оператора SELECT после открытия компонента TQuery возвращается НД, в котором указатель текущей записи всегда установлен на первую запись (если она имеется). Такой компонент TQuery следует открывать:

– установкой свойства Active в значение True,
– или выполнением метода

procedure Open;

Например,

```
Query1.Active:= True;
Query1.Open;
```

В случае использования операторов INSERT, UPDATE, DELETE набор данных не возвращается. Такой компонент TQuery следует открывать, выполняя метод

procedure ExecSQL;

Например,

```
InsertQuery.ExecSQL;
```

Метод ExecSQL посылает серверу для выполнения SQL-оператор и свойства SQL данного компонента TQuery.

Закрытие компонента TQuery осуществляется методом

procedure Close;

или установкой в False свойства Active, например:

```
Query1.Active := False;
Query1.Open;
```

При этом следует помнить, что для компонента TQuery, не возвращающего набор данных, выполнение метода Close не имеет последствий, поскольку с данным компонентом не связан открытый НД. Для динамических запросов, особенно для отсылаемых к удаленной БД, полезно использовать методы, осуществляющие «связывание» параметров с их фактическими значениями (Prepare) и отменяющие такое «связывание» (UnPrepare). Более подробно о них будет рассказано далее в подразделах, посвященных выполнению динамических запросов.

Изменяемые TQuery

Записи НД, возвращаемые компонентом TQuery, могут изменяться, подобно тому, как это происходит в компоненте TTable. Записи можно редактировать в компоненте TDBGrid, связанном с TQuery (метод автоматического перевода НД в состояния dslnsert, dsEdit, автоматического выполнения методов Insert, Edit, Delete, Post и Cancel). Также может применяться метод программного формирования значения полей записи, ввода таких значений с использованием компонента TDBEdit, TDBCheckBox и других. В этом случае методы Insert, Edit, Delete, Post и Cancel вызываются в программе явно.

Возможность изменения НД, возвращаемого после выполнения оператора SELECT, определяется свойством

property CanModify: Boolean;

Значение этого свойства устанавливается автоматически, исходя из определенных факторов. Если в процессе выполнения свойство CanModify установлено в True, набор данных доступен для изменения. При значении False записи НД не могут быть добавлены, изменены или удалены.

Свойство CanModify всегда устанавливается в False, если в False установлено свойство RequestLive компонента TQuery:

property RequestLive: Boolean;

Значение данного свойства может быть установлено как во время проектирования, так и во время разработки приложения. По умолчанию всегда устанавливается False (НД доступен только для чтения). Однако установка свойства в значение True (НД может быть изменен) вовсе не означает, что действительно будет позволено изменяться и что свойство CanModify будет установлено в True. Это произойдет только в том случае, если синтаксис SELECT при выполнении запроса будет признан «верным».

Синтаксис оператора SELECT будет признан «неверным», если:

– НД формируется более чем из одной ТБД;

– присутствует предложение принудительной сортировки результирующего набора данных ORDER BY;

– значения хотя бы одного столбца результирующего НД сформировано с использованием агрегатных функций (SUM, COUNT, AVG, MIN, MAX);

– при доступе к СУБД Sybase в таблице отсутствует уникальный индекс.

В том случае, если свойство RequestLive установлено в True, а синтаксис оператора SELECT признан «неверным»:

– возвращается НД, доступный только для чтения — при доступе к таблицам локальных СУБД (Paradox, dBase);

– выдается ошибка — при доступе к серверным СУБД (InterBase, Oracle) и т.д.

Если изменения, внесенные в НД методами Post, Delete, не отображаются в НД, его содержимое можно обновить методом

procedure Refresh;

Однако в этом случае оператор SELECT должен быть выполнен для таблицы локальной СУБД и эта таблица должна иметь уникальный индекс. Для НД, возвращенных в результате выполнения запроса к удаленной СУБД, исполнение метода Refresh не влечет за собой никаких последствий.

Для случая работы с удаленными БД следует избегать выполнения изменений при помощи методов Insert, Edit, Delete записей в НД, полученного при помощи TQuery. В этом случае вся нагрузка на модификацию удаленных таблиц ложится на клиентское приложение, в то время как в соответствии с одним из краеугольных камней идеологии «клиент-сервер» — вся тяжесть операций по физическому доступу к БД должна ложиться на сервер БД.

Предотвратить ввод записей, не удовлетворяющих условиям, перечисленным в предложении WHERE оператора SELECT, можно путем установки в True значения свойства

property Constrained: Boolean;

Например, для НД, полученного по запросу

SELECT * FROM JOB WHERE OKLAD > 1000

при Constrained, установленном в True, будут блокироваться попытки запоминания записей со значением поля OKLAD, меньшим 1000.

В том случае, если НД доступен только для чтения, его записи могут быть изменены при помощи SQL-операторов

INSERT, UPDATE, DELETE. При этом изменения в НД не отображаются. Для отображения изменений НД следует переоткрыть, выполнив метод Close и повторно Open. Повторное открытие НД устанавливает указатель текущей записи на первую строку, что удобно далеко не всегда. В этом случае можно:

1) вносить изменения в НД пакетом, визуализируя их путем его повторного открытия после равных порций изменений — например, после каждых 10 изменений; этот метод больше подходит для случаев значительного количества изменений в НД;

2) реализовать в форме локатор — механизм поиска записи, удовлетворяющей некоторому условию (значение условия можно вводить, например, и компонент TEdit или группу компонентов TEdit или автоматически помещать туда значения полей, однозначно идентифицирующих запись для последней добавленной или измененной записи). Поиск реализуется при нажатии экранной кнопки. Для поиска используются различные методы, например метод Locate.

3) перед повторным открытием НД запоминать значение поля (полей), однозначно идентифицирующего запись, и после открытия восстанавливать местоположение указателя текущей записи при помощи метода Locate.

Например, пусть в набор данных (компонент QueryPeople) добавляются записи при помощи SQL-оператора INSERT (компонент InsertQuery). Однозначно идентифицирует запись в НД Query поле Id_People. Тогда восстановление указателя текущей записи после повторного открытия НД QueryPeople может быть реализован так:

```
…….
TmpId_People :=
QueryPeople.FieldByName(‘Id_People’).AsInteger;
InsertQuery.ExecSQL;
QueryPeople.Close;
QueryPeople.Open;
QueryPeople.Locate(‘Id_People’,TmpID_People, []);
………
```

Выполнение динамических запросов

Понятие динамического запроса

Динамическим (параметрическим) является запрос, в SQL-операторе которого в процессе выполнения приложения могут изменяться отдельные его составляющие. В этом случае изменяемая часть оператора оформляется как Параметры.

Например, пусть в процессе выполнения приложения может быть выдан запрос: выдать все записи из таблицы Contract, составленные 01 марта 2004 г:

```
SELECT * FROM CONTRACT
WHERE CreateDay = «01.03.04»
```

И запрос: выдать все записи из таблицы Contract, составленные 10 марта 2004 г:

```
SELECT * FROM CONTRACT
WHERE CreateDay = «10.03.04»
```

Также аналогичный запрос по договорам за другие даты.

Конечно, каждый такой запрос можно реализовать в отдельном компоненте TQuery, но только теоретически — при достаточно большом числе договоров и дат число компонентов TQuery должно стремиться к бесконечности. Поэтому разумнее применить только один компонент TQuery, указав в его свойстве SQL оператор, в котором изменяющиеся части заменены на параметры:

```
SELECT * FROM CONTRACT
WHERE CreateDay = :DayCreate
```

Под параметром понимается имя, предваренное кавычками «:».

В динамических запросах параметры всегда заменяют значения, которые могут изменяться в процессе выполнения. Имена параметров произвольны и могут не совпадать со значениями полей таблицы, которым они обычно ставятся в соответствие.

Заметим, что такой подход не годится для случая удаления записей при помощи SQL-оператора DELETE, поскольку после удаления в НД не существует записи с запомненным значением уникального поля (полей).

Формирование динамического запроса

Для формирования динамического запроса необходимо:

1) выбрать для существующего компонента TQuery в инспекторе объекте свойство SQL и нажать кнопку текстового редактора;

2) в появившемся окне текстового редактора набрать текст SQL-запроса с параметрами;

3) выбрать в инспекторе объектов свойство Params и нажать кнопку в строке данного свойства; в появившемся окне будут показаны имена всех параметров, введенных в текст динамического SQL-оператора на шаге 2; список параметров отслеживается автоматически всякий раз при изменении содержимого свойства SQL;

4) каждому параметру из списка необходимо поставить через Инспектор объектов в соответствие определенный тип и, если нужно, стартовое значение в поле Value. Переключатель Null Value позволяет указать в качестве стартового значения NULL. Стартовые значения присваивать необязательно, однако каждому параметру необходимо поставить в соответствие определенный тип данных, иначе попытка открытия компонента TQuery может привести к возбуждению исключения;

5) компонент TQuery можно сделать активным (установить свойство Active = True) на стадии разработки приложения только в том случае, если каждому из параметров присвоено стартовое значение.

Компонент TQuery, содержащий динамический запрос, если его свойство Active установлено на этапе разработки в значение True, открывается при создании формы, содержащей данный компонент TQuery. При этом он использует значения параметров, установленные по умолчанию (стартовые значения). Если хотя бы одному из параметров не назначено стартовое значение, то выдается ошибка.

Если компонент TQuery, содержащий динамический запрос, не открыт в момент создания формы, его можно открыть в некоторый момент времени, программно установив значения параметров и выполнив метод Open или установив свойство Active в True.

Впоследствии всякий раз, когда необходимо изменить значения параметров запроса (что приведет к выдаче другого НД), нужно закрыть компонент TQuery, программно присвоить значения параметрам и повторно открыть компонент.

Установка значений параметров динамического запроса во время выполнения

Самым распространенным способом указания текущих значений параметров является их ввод пользователем в поля ввода (компоненты TEdit и другие) и последующее программное назначение параметров.

Параметры компонента TQuery доступны через его свойство

property Params[Index: Word]:TParams;

Это свойство является набором параметров, где каждый параметр определяется индексом в диапазоне O...ParamCount-l, где ParamCount есть число параметров, которое можно получить с помощью свойства ParamCount компонента TQuery.

property ParamCount: Word;

Обратиться к конкретному параметру можно:

1) указав индекс параметра в свойстве Params компонента TQuery, например, Params[0]. Порядок следования параметров аналогичен показываемому в окне редактора параметров (активизирующегося после нажатия кнопки в строке свойства Params инспектора объектов);

2) через метод компонента TQuery

function ParamByName(const Value: string): TParam;

где Value определяет имя параметра.

Для установки значения конкретного параметра используется одно из свойств компонента TParam AsNNN (AsString, Aslnteger и т.д.) или более общее свойство

property Value: Variant;

Например,

QueryPeople.Params[0].AsDate:= StrToDate(Editl.Text);

QueryPeople.ParamByName(‘CreateDay’).Value:= TmpDat;

По разным причинам попытка открыть НД может быть неуспешной. Поэтому рекомендуется помещать программный код, реализующий открытие НД, внутрь оператора TRY, например:

```
TRY
QueryPeople.Open;
EXCEPT
…..
END;//try
```

Например, будем вводить значения параметров в компонентеTEdit. Тип параметра будем задавать через выбор в ComboBox1: TComboBox. Открытие НД (компонента Query1), будем производить по нажатию кнопки Выполнить запрос. Параметризированный запрос будем писать в Memo1:TMemo

Вид формы данного приложения приведен на рисунке 15.

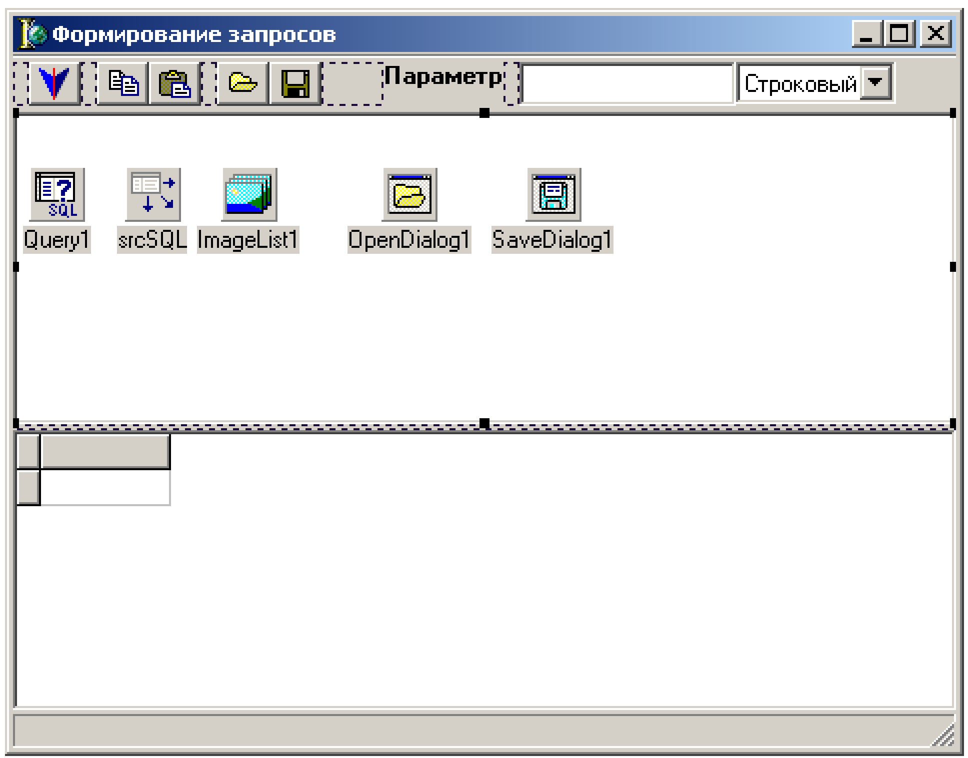


Рис. 15 — Вид формы приложения для работы с запросами

Обработчик события нажатия кнопки Выполнить запрос имеет вид:

```
procedure TfmSQL.ToolButton2Click(Sender: TObject);
begin
  if Query1.Active then Query1.Close;
Query1.DatabaseName:=ExtractFilePath(Application.ExeName)+'DB';
  Query1.SQL.Text:=Memo1.Lines.Text;
```

```
    if Query1.ParamCount>0 then begin
      case ComboBox1.ItemIndex of
        0: Query1.Params[0].AsString:=Edit1.Text;
        1: Query1.Params[0].AsInteger:=StrToInt(Edit1.Text);
        2:
Query1.Params[0].AsDateTime:=StrToDateTime(Edit1.Text);
      end;
    end;
    Query1.Active:=True;
  if Query1.RecordCount=0 then StatusBar1.SimpleText:='Таких
значений нет!!!'
    else StatusBar1.SimpleText:='Значения найдены! Всего: '
    +IntToStr(Query1.RecordCount);
  end;
```

Методы Prepare и Unprepare

Синтаксис SQL-операторов проверяется только при их выполнении.

Проверка синтаксиса требует времени, что особенно актуально для больших запросов при обращении к удаленным БД. Однако в динамических запросах изменяются только значения параметров, а сам синтаксис хотя бы единожды исполненного SQL-оператора является верным. Поэтому каждый раз выполнять проверку синтаксиса такого оператора не следует. Вместо этого запрос компилируется в исполняемый код и, если текст его не изменялся, а изменялись только значения параметров, проверка синтаксиса не производится и происходит немедленное выполнение уже скомпилированного запроса.

Для того чтобы «подготовить» запрос к многократному использованию, следует хотя бы один раз выполнить метод Prepare компонента TQuery:

procedure Prepare;

Выполнение этого метода для динамических запросов с неизменным синтаксисом лучше делать в обработчике события формы OnCreate.

В дальнейшем можно многократно присваивать параметрам различные значения и выполнять запрос методами Open и ExecSQL.

Выполнение метода Prepare необязательно в том смысле, что и без этого запрос будет выполнен. Однако выполнение метода Prepare для динамически запросов строго рекомендуется. В противном случае «подготовка» будет производиться всякий раз при выполнении запроса.

Свойство компонента TQuery

property Prepared: Boolean;

возвращает True, если НД был «подготовлен» методом Prepare.

Метод

procedure UnPrepare;

позволяет освободить ресурсы, выделенные для «подготовленного» запроса. Этот метод неявно вызывается всякий раз, когда изменяется текст SQl оператора запроса, что также ведет к немедленному закрытию НД.

Формируемые запросы

Часто один компонент TQuery используют для выполнения различных отстоящих друг от друга во времени запросов. Такой подход уменьшает число используемых компонентов, но может привести к возрастанию программного кода.

Свойство SQL компонента TQuery имеет тип TStrings:

property SQL: TStrings;

и потому содержимое свойства SQL может формироваться программно методами Add (добавить элемент), Delete (удалить элемент), Clear (очистить список) и прочими для TStrings.

Например:

```
with QueryPeople do begin
  Close;
  SQL.Clear;
  SQL.Add(‘Select * ’);
  SQL.Add(‘From People ’);
  SQL.Add(‘Where Last_Name=”Петров”’);
  Open;
end; // with
```

Кроме того, можно обраться к свойству SQL указав номер строки текста запроса, начиная от нуля.

Например:

```
with QueryPeople do begin
  Close;
  SQL[0]:= ‘Select * From People ’;
  SQL[1]:= ‘Where Last_Name=”Петров”’;
  SQL[2]:= ‘Order by BirthDay’;
  Open;
end; // with
```

В данном случае к каждой строке запроса можно обраться отдельно по ее индексу. При этом строка с определенным индексов уже должна существовать в свойстве SQL.

Свойство TQuery.SQL.Text: String позволяет одной строкой кода задать весь текст запроса, затирая предыдущее значение свойства SQL.

Практика

Создание формы для работы с SQL

Откроем новую, пустую форму.

Поместим на неё компоненты TQuery, TDataSource и DBGrid.

Настроим эти компоненты.

Данные компоненты — минимум для визуальной работы с SQL, при желании вы можете добавить на форму другие компоненты (рис. 16).

В свойстве SQL компонента TQuery напишите простейший запрос, например:

Select * From People

Данный запрос выведет целиком таблицу People.db.

В свойстве Active этого же компонента укажите True, если всё правильно компонент должен отобразить в окне результат запроса.

Чтобы формировать запросы через запущенное приложение положим компонент TMemo (в нем будем писать запросы).

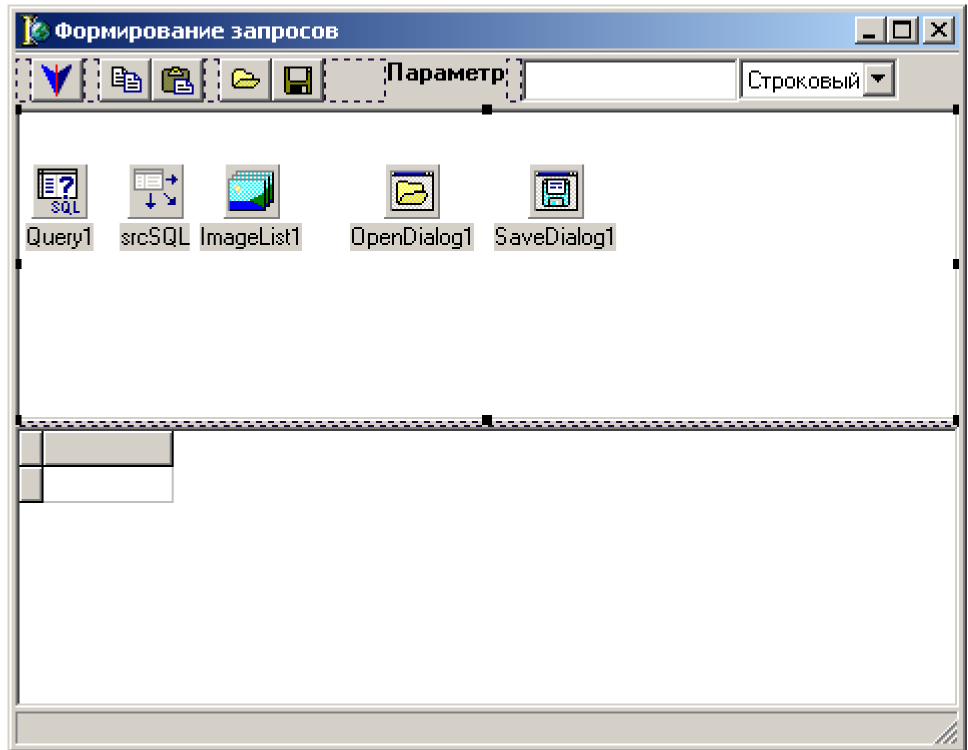


Рис. 16 — Вид формы для формирования
запросов к БД отдела кадров

Выполнять запросы будем по нажатию кнопки на панели инструментов. В строке статуса StatusBar1 будем выводить количество полученный в результате запросов значений.

Через ComboBox1: TComboBox (добавив через его свойство Items значения: Строковый, Целочисленный, Дата) будем выбирать тип параметра для параметризированного запроса.

Для этого на обработчик события нажатия кнопки на панели инструментов запишем (как уже было сказано):

```
     if Query1.Active then Query1.Close;

Query1.DatabaseName:=ExtractFilePath(Application.ExeName)+'db';
      Query1.SQL.Text:=Memo1.Lines.Text;
      if Query1.ParamCount>0 then begin
        case ComboBox1.ItemIndex of
          0: Query1.Params[0].AsString:=Edit1.Text;
          1: Query1.Params[0].AsInteger:=StrToInt(Edit1.Text);
          2:
Query1.Params[0].AsDateTime:=StrToDateTime(Edit1.Text);
```

```
  end;
 end;
 Query1.Active:=True;

if Query1.RecordCount=0 then StatusBar1.SimpleText:='Таких значений нет!!!'
 else StatusBar1.SimpleText:='Значения найдены! Всего: '
 +IntToStr(Query1.RecordCount);
end;
```

Помимо непосредственного написания текста запроса в компоненте Memo1:TMemo

на панели инструментов предусмотрим кнопки:

– копирование содержимого Memo1:TMemo в буфер:

```
procedure TfmSQL.ToolButton4Click(Sender: TObject);
begin
 Memo1.CopyToClipboard;
end;
```

– вставка содержимого из буфера в Memo1:TMemo:

```
procedure TfmSQL.ToolButton5Click(Sender: TObject);
begin
 Memo1.PasteFromClipboard;
end;
```

– вставка содержимого открытого файла через стандартный диалог открытия файла в Memo1:TMemo. Для этого с палитры компонент Dialogs положим компонент OpenDialog1: TOpenDialog — для открытия стандартного диалога открытия файла и в обработчике нажатия кнопки на панели инструментов запишем:

```
procedure TfmSQL.ToolButton7Click(Sender: TObject);
begin
 with OpenDialog1 do
   if Execute then try
     Memo1.Lines.LoadFromFile(FileName);
   except
```

```
      ShowMessage('Ошибка !!!');
    end;
end;
```

– сохранение содержимого Memo1:TMemo в файл через стандартный диалог сохранения файла. Для этого с палитры компонент Dialogs положим компонент SaveDialog1: TSaveDialog — для открытия стандартного диалога сохранения файла и в обработчике нажатия кнопки на панели инструментов запишем:

```
procedure TfmSQL.ToolButton8Click(Sender: TObject);
begin
  with SaveDialog1 do
    if Execute then Memo1.Lines.SaveToFile(FileName);
end;
```

Создание структурированных запросов SQL

Выше было приведёно описание SQL-оператора SELECT. Задача — изучение оператора SELECT на практике.

Запросы будем формировать для разработанной БД отдела кадров.

1. Внимательно изучите примеры, приведённые выше.
2. Изучите примеры приведённые ниже.
3. В свойстве SQL компонента TQuery ранее созданной формы задайте рассмотренные запросы. Запросы можно задавать после запуска созданного приложения в Memo1:TMemo и нажатия кнопки Выполнить запрос.
4. Внимательно изучите выдаваемые результаты, внимательно посмотрите по структуре базы данных их получение.

Примеры

Привести данные о сотрудниках подразделения бухгалтерия:

```
select p.Last_name, p.First_Name, p.Second_Name,
j.Job_Name, o.Otdel_Name
from people p, job j , contract c, otdel o
```

where (o.Otdel_Name='Бухгалтерия') and (o.Id_Otdel=c.Id_Otdel) and (c.Id_People = p.Id_People) and (c.Id_Job=j.Id_Job)

или

```
select p.Last_name+' '+p.First_Name+' '+p.Second_Name as ФИО, j.Job_Name, o.Otdel_Name
from people p, job j , contract c, otdel o
where (o.Otdel_Name='Бухгалтерия') and (o.Id_Otdel=c.Id_Otdel) and (c.Id_People = p.Id_People) and (c.Id_Job=j.Id_Job)
```

Показать количество сотрудников в подразделениях:

```
select o.Otdel_name as Подразделение, count(c.id_people) as Kolvo from contract c, otdel o
where c.id_otdel=o.id_otdel
group by o.Otdel_name
```

Показать сотрудников, для которых оклад выше среднего:

```
select p.Last_name, p.First_Name, p.Second_Name, j.Job_Name, o.Otdel_Name
from people p, job j , contract c, otdel o
where (o.Id_Otdel=c.Id_Otdel) and (c.Id_People = p.Id_People) and (c.Id_Job=j.Id_Job) and (j.Oklad > (select avg(Oklad) from Job))
```

ЛАБОРАТОРНАЯ РАБОТА № 6
ПОИСК, ФИЛЬТРАЦИЯ И СОРТИРОВКА ЗАПИСЕЙ В НД

Задание

1. Изучение различных методов поиска, фильтрации и сортировок записей в НД TTable и TQuery.

2. Применение изученных методов в приложении.

Обзор методов поиска записей в НД

Существует множество способов поиска записей в наборах данных. Рассмотрим некоторые общие для наборов данных TTable и TQuery средства поиска записей в НД. Кроме перечисленных для компонента TTable можно воспользоваться методами Find; GoToKey, FindNearest, GoToNearest.

Метод Locate

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

Метод Locate ищет первую запись, удовлетворяющую критерию поиска, и ели такая запись найдена, делает ее текущей. В этом случае в качестве результата возвращается True. Если поиск был неуспешен, возвращается False.

Параметры:

Список *KeyFields* указывает поле или несколько полей, по которым ведется поиск, в виде строкового выражения. В случае нескольких поисковых полей их названия разделяются точкой с запятой.

Критерии поиска задаются в вариантном массиве *Key Values* так, что i-е значение в Key Values ставится в соответствие i-му полю в KeyFields. В случае поиска по одному полю в Key Values указывается одно значение.

Options позволяет указать необязательные значения режимов поиска:

loCaseInsensitive — поиск ведется без учета высоты букв, т.е. если в Key Values указано 'принтер', а в некоторой записи в данном поле встретилось 'Принтер' или 'ПРИНТЕР', запись считается удовлетворяющей условию поиска;

loPartialKey — запись считается удовлетворяющей условию поиска, если она содержит часть поискового контекста; например, удовлетворяющими контексту «Ма» будут признаны записи с значениями в искомом поле «Машин», «Макаров» т.д.

Locate отличается от методов *FindKey, FindNearest, GoToKey, GoToNearest* (компонент TTable) следующим:

– FindKey, FindNearest, GoToKey, GoToNearest производят поиск только по полям, входящим в состав текущего индекса TTable; в случае, когда условию поиска удовлетворяет несколько записей, текущей станет логически самая первая из них (в порядке сортировки записей в НД, определяемом текущим индексом);

– Locate производит поиск по любому полю. Поле или поля, по которым производится поиск, могут не только не входить в текущий индексе, но и не быть индексными вообще.

В случае, если поля поиска входят в какой-либо индекс, Locate использует этот индекс при поиске. Если искомые поля входят в несколько индексов, то трудно сказать, какой из них будет использован. Соответственно, трудно предсказать, какая запись из множества записей, удовлетворяющих критерию поиска, будет сделана текущей — особенно в случае, если поиск ведется не по текущему индексу.

При поиске по полям, не входящим ни в один индекс, применяются фильтры BDE.

Например:

```
procedure TFormX.LocateButtonClick (Sender : TObject);
begin
Tablel.Locate(‘Last_Name; First_Name’, VarArrayOf([Edit1.text, Edit2.text]), [loPartialKey]);
end;

procedure TFormX.LocateButtonClick (Sender : TObject);
begin
if not Queryl.Locate(‘Last_Name’, Edit1.text, [loCaseInsensitive, loPartialKey]) then
ShowMessage(‘Такой фамилии нет!’);
end;
```

Использование методов FindFirst, FindLast, FindNext, FindPrior

Известно, что набор данных может быть отфильтрован с использованием свойства Filtered (см. далее). Условие фильтрации задается свойством Filter или описывается в обработчике события OnFilterRecord. Свойство Filtered указывает, выполнять ли фильтрацию (значение True) или нет (значение False). В этом случае в НД показываются все записи, а не только удовлетворяющие условию фильтрации.

Для НД, в котором определены условия фильтрации, но сама фильтрация в текущий момент не включена, Delphi предоставляет интересную возможность. Она заключается в том, что в неотфильтрованном в данный момент НД можно обеспечить навигацию только между теми записями, которые удовлетворяют условию фильтрации (оно в текущий момент, когда свойство Filtered — False, не действует).

Для этой цели используются методы *FindFirst, FindLast, FindNext, FindPrior.*

Условие фильтрации можно сделать совпадающим с условием поиска, указанным в параметре KeyValues метода Locate. При этом поиск с помощью указанных методов имеет довольно большое преимущество перед поиском с помощью Locate: если в Locate можно указывать только значения, то в условии фильтрации можно указывать логические условия.

В случае, если искомая запись найдена, данные методы возвращают True, в противном случае — False.

function FindFirst: Boolean; — переходит на первую запись, удовлетворяющую фильтру;

function FindLast: Boolean; — переходит на последнюю запись, удовлетворяющую фильтру;

function FindNext: Boolean; — переходит на следующую запись, удовлетворяющую фильтру;

function FindPrior: Boolean; — переходит на предыдущую запись, удовлетворяющую фильтру;

property Found: Boolean; — возвращает True, если последнее обращение к одному из методов FindFirst, FindLast, FindNext, FindPrior привело к нахождению нужной записи.

Например, предоставим пользователю возможность перемещаться на первую, последнюю, следующую, предыдущую запись, удовлетворяющую условию. Содержимое Edit1 входит как часть фамилии сотрудника. Заметим, что свойство Table1.Filtered = False, т.е. хотя в обработчике события Tablel.OnFilterRecord и указано условие фильтрации, в НД показываются все записи, и он остается в не отфильтрованном состоянии.

```
//условие фильтрации
procedure TFormX.TablelFilterRecord(DataSet: TDataSet; var
Accept: Boolean);
begin
Accept := POS(Editl.Text, DataSet['Last_Name']) > 0;
end;
// нажата кнопка "Первая"
procedure TFormX.FindFirstButtonClick(Sender: TObject);
begin
Label1.Caption := ‘’;
IF not Tablel.FindFirst THEN
Labell.Caption := 'Нет такой записи';
end;
// нажата кнопка "Последняя"
procedure TFormX.FindLastButtonClick(Sender: TObject);
begin
Labell.Caption := ‘’;
IF not Tablel.FindLast THEN
Labell.Caption := ‘Нет такой записи’;
end;
// нажата кнопка "Следующая"
procedure TFormX.FindNextButtonClick(Sender: TObject);
begin
Labell.Caption := ‘’;
IF not Tablel.FindNext THEN
Labell.Caption := ‘Нет такой записи’;
end;
// нажата кнопка "Предыдущая"
procedure TFormX.FindNextButtonClick(Sender: TObject);
begin
Labell.Caption := ‘’;
```

```
IF not Tablel.FindPrior THEN
Labell.Caption := 'Нет такой записи';
end;
```

Заметим, что поскольку фильтрация записей с использованием события OnFilterRecord или (и) свойства Filter может применяться только на небольших объемах записей (из-за того, что при этом используется последовательный метод доступа к записям в таблице БД), аналогичные ограничения накладываются и на поиск записей с использованием методов FindFirst, FindLast, FindNext, FindPrior.

Метод Lookup

function Lookup(const KeyFields: string; const Key Values: Variant; const ResultFields: string): Variant;

Метод Lookup находит запись, удовлетворяющую условию, но не делает ее текущей, а возвращает значения некоторых полей этой записи. Тип результата — Variant или вариантный массив. Независимо от успеха поиска записи, указатель текущей записи в НД не изменятся.

Lookup осуществляет поиск только на точное соответствие критерия поиска и значения полей записи. Такой режим, как loPartialKey метода Locate (поиск по частичному соответствию значений), отсутствует.

Параметры:

В *KeyFields* указывается список полей, по которым необходимо осуществить поиск. При наличии в этом списке более чем одного поля, соседние поля разделяются точкой с запятой.

KeyValues указывает поисковые значения полей, список которых содержится в KeyFields. Если имеется несколько поисковых полей каждому i-му полю в списке KeyFields ставится в соответствие i-ое значение в списке KeyValues. При наличии одного поля, его поисковое значение можно указывать в качестве KeyValues непосредственно; в случае нескольких полей — их необходимо приводить к типу вариантного массива при помощи *VarArrayOf.*

В качестве поисковых полей можно указывать поля как входящие в какой-либо индекс, так и не входящие в него; тип теку-

щего индекса не имеет значения. Если поисковые поля входят в какие-либо индексы, их использование производится автоматически; в противном случае используются фильтры BDE.

Если запись в результате поиска не найдена, метод Lookup возвращает Null, что выявляется при помощи предложения

```
IF VarType(LookupResults) = varNull THEN ...
```

В противном случае Lookup возвращает из этой записи значения полей, список которых указан в ResultFields. При этом размерность результата зависит от того, сколько результирующих полей указано в ResultFields:

– указано одно поле — результатом будет значение соответствующего типа или Null, если поле в найденной записи содержит пустое значение;

– указано несколько полей — результатом будет вариантный массив, число элементов в котором меньше или равно числу результирующих полей; меньше потому, что некоторые поля найденной записи могут содержать пустые значения.

Фильтрация записей в НД

Существует множество способов фильтрации записей в наборах данных.

Рассмотрим некоторые общие для наборов данных TTable и TQuery средства фильтрации записей в НД.

Помимо описываемых ниже средств, для фильтрации данных могут использоваться:

– методы SetRange или ApplyRange (и сопутствующие им методы) — в компоненте TTable;

– секция WHERE оператора SELECT языка SQL — в компоненте TQuery.

Свойство Filtered

property Filtered: Boolean;

Свойство Filtered, установленное в True, инициирует фильтрацию, условие которой записано или в обработчике события OnFilterRecord, или содержится как строковое значение в свойстве Filter.

Если установлены разные условия фильтрации и в событии OnFilterRecord, и в свойстве Filter, выполняются оба.

Например, если в НД одновременно установлены фильтры

Tablel.Filter = ‘Last_Name = ’’Петров’’’;

и

procedure TFormX.TablelFilterRecord(DataSet: TDataSet; var Accept: Boolean);

begin

Accept := DataSet['First_Name'] = 'Иван';

end;

то установка Tablel.Filtered в True приведет к двум фильтрациям; в результирующем наборе данных будут показаны только записи, у которых поле Last_Name содержит значение 'Петров' и поле 'First_Name' содержит значение 'Иван'.

Установка Filtered в False приведет к отмене фильтрации, условия которой указаны в событии OnFilterRecord или (и) свойстве Filter. При этом фильтрация, наложенная на НД методом *SetRange* или *ApplyRange* и ему сопутствующими методами, не нарушается.

Последовательность установки фильтров произвольна — SetRange может применяться после Filtered := True, и наоборот.

Отмена одного из этих условий фильтрации не приводит к отмене другого способа фильтрации.

Событие OnFilterRecord

property OnFilterRecord: TFilterRecordEvent;

Событие OnFilterRecord возникает, когда свойство Filtered устанавливается в True.

Обработчик события OnFilterRecord имеет два параметра: имя фильтруемого набора данных и *var Accept*, указывающий условия фильтрации записей в НД.

В отфильтрованный НД включаются только те записи, для которых параметр Accept имеет значение True.

В условие фильтрации могут входить любые поля НД, в том числе не входящие в текущий индекс, а также не входящие ни в один индекс. Возможность фильтрации НД по не индексным полям, а также полям, не входящим в текущий индекс, выгодно отличает способ фильтрации с использованием события

OnFilterRecord и свойства Filtered от способов фильтрации с использованием методов SetRange, ApplyRange и им сопутствующих методов (компонент TTable). Последние позволяют производить фильтрацию НД только по индексным полям, входящим к тому же в состав индекса, текущего на момент фильтрации. Кроме этого, второй способ часто не позволяет реализовывать сложные логические конструкции при указании условий фильтрации.

Однако следует помнить о том, что при указании условий фильтрации НД в обработчике OnFilterRecord, в нем последовательно перебираются все записи таблицы БД при анализе их на предмет соответствия условию фильтрации, в то время как методы SetRange, ApplyRange и им сопутствующие методы используют индексно-последовательный метод доступа, т.е. работают с частью записей в физической таблице БД. Это делает использование OnFilterRecord предпочтительным для небольших объемов записей и сильно ограничивает применение данного способа фильтрации при больших объемах данных.

Всякий раз, когда приложение обрабатывает событие OnFilterRecord, НД переводится из состояния dsBrowse в состояние dsFilter. Это предотвращает модификацию НД во время фильтрации. После завершения текущего вызова обработчика события OnFilterRecord, НД переводится в состояние dsBrowse.

Пример: отфильтровать таблицу БД «Сотрудники» согласно условию «Показать всех Сидоровых»:

```
procedure TForml.TablelFilterRecord(DataSet: TDataSet; var Accept: Boolean);
begin
  Accept := DataSet[‘Last_Name’] = ‘Сидоров’;
end;
```

Пример: отфильтровать таблицу БД «Сотрудники» по условию «Показать всех сотрудников с фамилией, вводимой в Editl, и с вхождением в имя символов, вводимых пользователем в Edit2»:

```
procedure TForml.TablelFilterRecord(DataSet: TDataSet; var Accept: Boolean);
begin
  Accept := (DataSet['Last_Name'] = Editl.Text)
            AND (Pos(Edit2.Text, DataSet[‘First_Name’]) > 0);
end;
```

Методы FindFirst, FindLast, FindNext, FindPrior также используют свойство OnFilterRecord, когда выполняют навигацию по НД.

Свойство Filter

property Filter: string;

Свойство Filter позволяет указать условия фильтрации. В этом случае НД будет отфильтрован, как только его свойство Filtered станет равным True.

Синтаксис похож на синтаксис предложения WHERE SQL-оператора SELECT с тем исключением, что: имена переменных программы указывать нельзя, можно указывать имена полей и литералы (явно заданные значения).

Можно применять операторы отношения:

<	Меньше чем
>	Больше чем
>=	Больше или равно
<=	Меньше или равно
=	Равно
<>	Не равно

а также использовать логические операторы AND, NOT и OR.

Строку фильтрации можно ввести во время выполнения.

```
//когда проставляется галка в поле компонента CheckBoxl (то есть
//когда  CheckBoxl.Checked =True), пользователь включает
//фильтрацию; когда пользователь снимает отметку (то есть когда
// CheckBoxl.Checked =True), пользователь выключает фильтрацию
procedure TForml.CheckBoxlClick(Sender: TObject); begin
  Tablel.Filter := Editl.Text;
  Tablel.Filtered := CheckBoxl.Checked;
end;
```

Однако при этом нужно следить, чтобы введенная строка соответствовала требованиям, предъявляемым к синтаксису строки Filter.

Другим способом мог бы быть обработчик, считывающий значения фильтрации и преобразующий их к формату строки Filter.

Символ ‘*’ позволяет заменять несколько символом в условии фильтрации.

Свойство FilterOptions

property FilterOptions: TFilterOptions;

TFilterOption = (foCaseInsensitive, foNoPartialCompare);

Свойство FilterOptions позволяет установить режимы фильтрации с использованием свойства Filter. По умолчанию FilterOptions = [ ].

foCaseInsensitive — Фильтрация производится без учета разницы в высоте букв.

foNoPartialCompare — поиск производится на точное соответствие В противном случае, при фильтре Фамилия = 'Ма' в отфильтрованный НД будут включены записи, у которых в поле Фамилия частично входит 'Ма', например (если используется опция foCaseInsensitive), '**Ма**нуйлова' и 'Ко**ма**рова'.

Навигация в неотфильтрованном НД между записями, удовлетворяющими фильтру

Методы FindFirst, FindLast, FindNext, FindPrior позволяют перемещаться в не отфильтрованном НД (у которого Filtered = False) между записями удовлетворяющими условию фильтрации. Условие фильтрации задается событием OnFilterRecord или (и) свойством Filter. Действие данных методов таково: они кратковременно переводят НД в отфильтрованное состояние (Filtered — True) без визуализации этой фильтрации в TDBGrid или другом подобном компоненте, находят соответствующую запись и переводят НД в не отфильтрованное состояние (Filtered — False).

Если искомая запись найдена, данные методы возвращают True в противном случае — False. Аналогичный результат возвращает свойство Found (см. ранее).

Сортировка записей в НД

Существует множество способов сортировки записей в наборах данных.

Для сортировки данных могут использоваться:

– сортировка определяется соответствующим индексом;

– секция ORDER BY оператора SELECT языка SQL — в компоненте TQuery.

Для TTable последовательность расположения записей в наборе данных определяется используемым индексом, обуславливающим сортировку. Для TQuery порядок следования записей либо случаен, особенно при использовании в качестве источника более одной таблицы БД, либо упорядочен в порядке перечисления полей, в разделе ORDER BY.

Начиная работать с НД, имеющим одну сортировку записей, мы можем затем переопределить сортировку, и записи "перестроятся» в соответствии с новой сортировкой, т.е. логический порядок их следования изменится.

Использование методов поиска, фильтрации и сортировки данных в приложении

Рассмотрим на примере поиска, фильтрации и сортировок данных БД отдела кадров.

Организация поиска и фильтрации данных в форме для работы со справочниками

В строке ввода Edit1: TEdit будем задавать условия поиска и фильтрации данных.

Фильтрацию будем проводить по нажатию галочки в CheckBox1: TCheckBox, специально добавленным для фильтрации.

Поиск будем осуществлять по нажатию специально добавленной кнопки btnFind: TToolButton на панели инструментов или по нажатию Enter на строке ввода.

Фильтрацию и поиск будем производить по выбранному в DBGrid1: TDBGrid столбцу.

Результат поиска и фильтрации отображается в DBGrid1: TDBGrid.

На обработчик события нажатия мышкой на кнопке панели инструментов btnFind запишем:

```
procedure TfmList.btnFindClick(Sender: TObject);
begin
```

```
    if not btnFind.Enabled then exit;
    try
    if DataSource1.DataSet.Locate(DBGrid1.SelectedField.FieldName,Edit1.Text,[loCaseInsensitive, loPartialKey])
    then DBGrid1.SetFocus
     else begin
     ShowMessage('Значение "'+Edit1.Text+'"'#13'в поле "'+DBGrid1.SelectedField.DisplayLabel+'"'#13'Не найдено!');
     Edit1.Text:='';
     Edit1.SetFocus;
     end;
    except
     ShowMessage('Ошибка поиска!');
     Edit1.Text:='';
    end;
   end;
```

Запишем для отработки поиска по нажатию Enter на строке ввода в событии OnKeyPress для Edit1:

```
procedure TfmList.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
 if Key=#13 then btnFindClick(Sender);
end;
```

Для фильтрации данных по нажатию галочки в CheckBox1:

```
procedure TfmList.CheckBox1Click(Sender: TObject);
var s:string;
begin
if CheckBox1.Checked then begin
   // Нажата галочка на CheckBox1
   s:=DBGrid1.SelectedField.FieldName+' = '''+Edit1.Text;
   if DBGrid1.SelectedField.DataType=ftString then
      s:=s+'*'''
     else s:=s+'''';
```

```
    try
      DataSource1.DataSet.Filter:=s;
    except
      ShowMessage('Ошибка фильтрации!!!');
      CheckBox1.Checked:=False;
      DataSource1.DataSet.Filtered:=False;
      Edit1.Text:='';
      exit;
    end;
  end;
    DataSource1.DataSet.FilterOptions:=[foCaseInsensitive];
    DataSource1.DataSet.Filtered:=CheckBox1.Checked;

  If DataSource1.DataSet.RecordCount=0 then begin
    ShowMessage('Нет значений');
    CheckBox1.Checked:=False;
    DataSource1.DataSet.Filtered:=False;
    Edit1.Text:='';
  end;
end;
```

Будем реагировать по изменению значений в сроке ввода:

```
procedure TfmList.Edit1Change(Sender: TObject);
begin
  CheckBox1.Checked:=False;
  CheckBox1Click(Sender);
  btnFind.Enabled:=(Length(Trim(Edit1.Text))>0);
  // Length(Trim(Edit1.Text) - Определяет длину строки
  // без концевых пробелов
end;
```

Организация поиска и сортировки данных через запрос

Будем осуществлять поиск по фамилии сотрудника.

Через запрос при этом по указанной фамилии предоставим информацию о должности, подразделении работы и информации о договоре для указанного сотрудника.

Результат поиска будем показывать через отдельную форму.

Для этого добавим новую форму в проект File/New/Form и назовем ее (свойство Name) fmFind. Таким образом мы создали новый класс TfmFind = class(TForm).

На форме разместим:

– набор данных qFind: TQuery;
– источник данных srcqFind: TDataSource;
– DBGrid1: TDBGrid;
– ToolBar1: TToolBar (по желанию);
– GroupBox1: TGroupBox;
– DBImage1: TDBImage;
– ImageList1: TImageList (по желанию).

Свяжем qFind: TQuery с источником данных srcqFind: TDataSource. DBGrid1: TDBGrid с srcqFind: TDataSource.

На GroupBox1: TGroupBox положим DBImage1: TDBImage и укажем для него через Инспектор объектов свойства DBImage1.DataSource:=srcqFind и DBImage1.DataField:='Photo'.

Установим для GroupBox1 свойство Align в alRight.

Установим для DBGrid1 DBGrid1.Align:=alClient.

На панели инструментов добавим кнопку для сортировки данных.

Укажем ей пиктограмму через ImageList1: TImageList (по желанию).

Вид формы приведен на рис. 17.

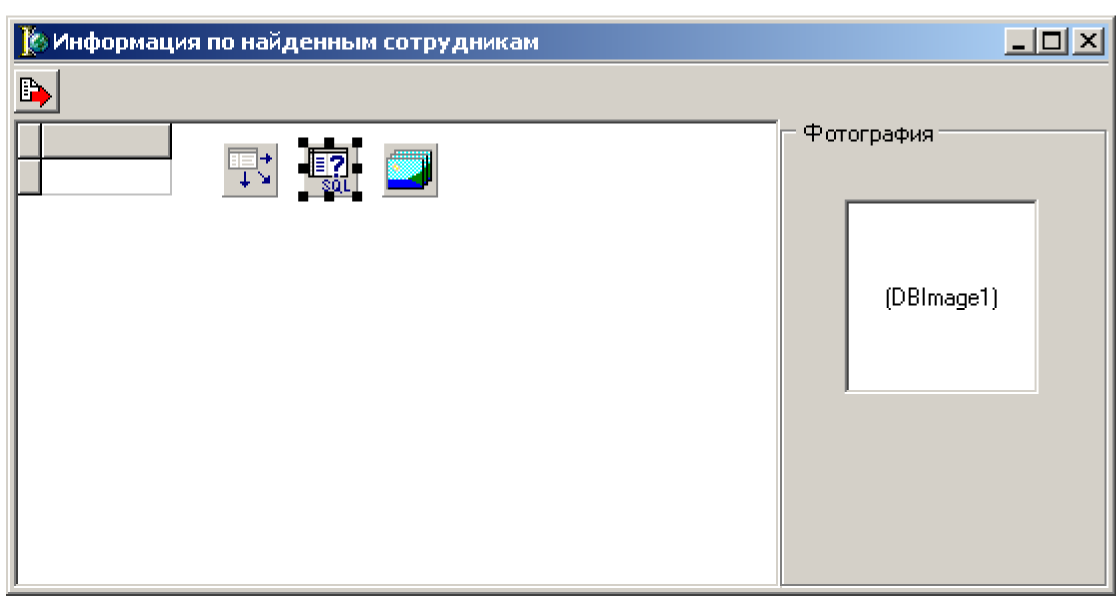


Рис. 17 — Вид формы для показа результата поиска сотрудников

Укажем путь к БД через свойство qFind.DatabaseName в путь к Вашей БД.

Свойство SQL для qFind: TQuery сформируем программно. Для этого объявим в блоке implementation переменную sSQL:string, для которой запишем наш запрос для выдачи информации о сотруднике:

```
var sSQL:string = 'select p.Last_Name, p.First_Name, p.Second_Name, '#13+
'p.BirthDay, p.Photo, p.Notes, j.Job_Name, j.Oklad, o.Otdel_name, '#13+
'c.Number, c.StartDay from people p, job j , contract c, otdel o '#13+
'where (c.Id_People = p.Id_People) and (c.Id_Job=j.Id_Job) '#13+
'and (c.Id_Otdel=o.Id_Otdel) and (p.Last_Name LIKE "%s") '#13;
```

Для ввода параметра поиска будем использовать диалог InputBox. Введенное значение (тип string) будем записывать в переменную sFind:string. Объявим ее в блоке public формы fmFind. Для этого запишем:

```
public
  { Public declarations }
  sFind:string;
end;
```

Будем вызывать диалог поиска через, например, добавленную акцию actFindPeople компонента ActionList1: TActionList или по нажатию кнопки или через другой управляющий элемент. Добавим акции и для OnExecute запишем:

```
procedure TfmMain.actFindPeopleExecute(Sender: TObject);
begin
 fmFind.sFind:=InputBox('Поиск по фамилии сотрудника','Введите фамилию','');
 if fmFind.sFind<>'' then fmFind.SHowModal
```

else ShowMessage('Вы ничего не ввели!');
end;

Сформированный при этом диалог InputBox('Поиск по фамилии сотрудника','Введите фамилию','') будет иметь вид (рис. 18).

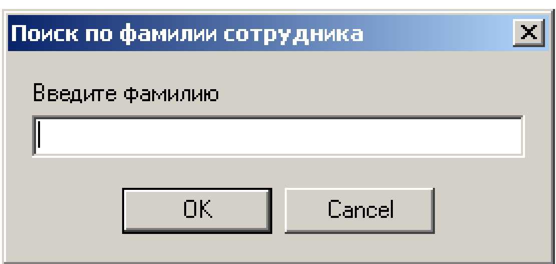


Рисунок 18 Вид заданного диалога InputBox

На момент показа формы fmFind запишем:

```
procedure TfmFind.FormShow(Sender: TObject);
begin
  qFind.Close;
  // формируем текст запроса динамически подставляя значение строки
  // фильтра поиска
  qFind.SQL.Text:=Format(sSQL,[sFind+'%']);
  // Функция форматирования строки
  // в качестве первого параметра идет строка форматирования
  // вторым параметром идет массив переменной длины
  //который позволяет указать через "," различное количество параметров
  // различного типа в []
  // Возвращает исходную строку, подставляя вместо %s
  // строковый параметр из массива аргументов
  qFind.Open;
  if qFind.RecordCount = 0 then ShowMessage('Таких значений нет!');
 end;
```

Недостаток такого поиска по фамилии — важно с большой или маленькой букв указана заданная фамилия (ограничения Where). Для избежания этого недостатка можно использовать свойство Filter и Filtered с указанием FilterOptions:=[foCaseInsensitive].

Сортировка данных

Будем сортировать данные по выбранной пользователем колонке и нажатию кнопки на панели инструментов и сразу сортировать (без нажатия кнопок) по нажатию соответствующего заголовка компонента DBGrid1.

Будем сортировать значения в выбранном пользователем заголовке компонента DBGrid1. Для этого в обработчике события OnTitleClick у DBGrid1 запишем:

```
procedure TfmFind.DBGrid1TitleClick(Column: TColumn);
var s:string;
begin
  s:=Column.FieldName;
  qFind.Close;
  // формируем текст запроса динамически подставляя значение строки
  // фильтра поиска
  qFind.SQL.Text:=Format(sSQL+'ORDER BY '+s,[sFind+'%']);
  qFind.Open;
end;
```

Для сортировки по выбранной колонке и нажатию кнопки на панели инструментов запишем:

```
procedure TfmFind.ToolButton1Click(Sender: TObject);
var s:string;
begin
  s:=DBGrid1.SelectedField.FieldName;
  qFind.Close;
  // формируем текст запроса динамически подставляя значение строки
  // фильтра поиска
  qFind.SQL.Text:=Format(sSQL+'ORDER BY '+s,[sFind+'%']);
  qFind.Open;
end;
```

Можно организовать поиск, фильтрацию и сортировку данных и другими способами.

ЛАБОРАТОРНАЯ РАБОТА № 7
ПОСТРОЕНИЕ ОТЧЕТНЫХ ФОРМ

Задание

1. Изучение компонент с палитры QReport.

2. Разработка варианта отчетной формы по информации из БД отдела кадров.

Компоненты для построения отчетов

В Delphi на странице палитры компонентов QReport расположено около двух десятков компонентов, применяемых для построения отчетов. «Главным» компонентом, является **TQuickRep**, определяющий поведение отчета в целом. Другие компоненты определяют составные части отчета. Перечислим основные:

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

– область заголовка отчета;

– область заголовка страницы;

– область заголовка группы;

– область названий столбцов отчета;

– область детальных данных, предназначенную для отображения данных самого нижнего уровня детализации;

– область подвала группы;

– область подвала страницы;

– область подвала отчета.

– **TQRSubDetail** — определяет область, в которой располагаются данные подчиненной таблицы при реализации в отчете связи Master-Detail на основе существующей связи между ТБД;

– **TQRGroup** — применяется для группировок данных в отчете;

– **TQRLabel** — позволяет разместить в отчете статический текст;

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

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

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

– **TQRMemo** — служит для вывода в отчете содержимого полей комментариев;

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

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

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

– **TQRDBImage** — служит для вывода в отчете графической информации, источником которой является поле набора данных;

– **TQRChart** — служит для встраивания в отчет графиков.

Компонент TQuickRep

Компонент TQuickRep определяет поведение и характеристики отчета в целом.

При размещении этого компонента в форме в ней появляется сетка отчета (рис. 19).

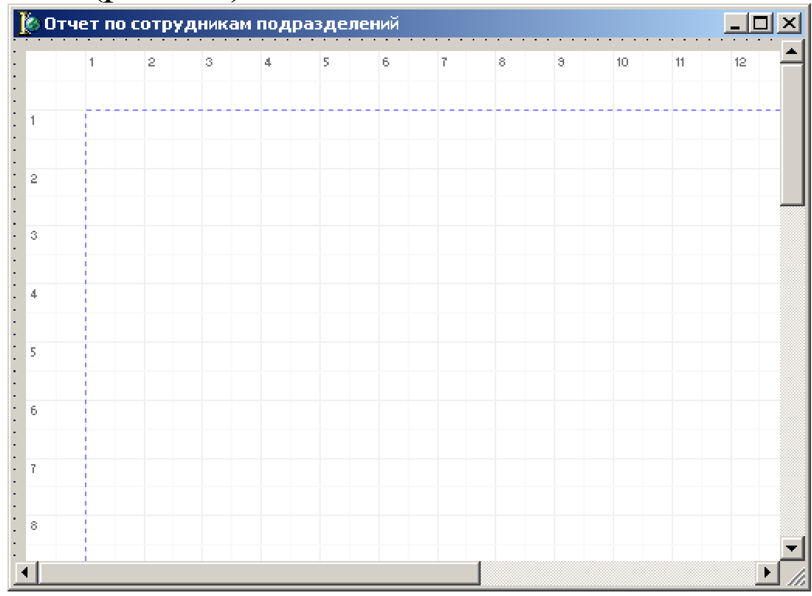


Рис. 19 — Пустая сетка отчета

Образуется после размещенияв форме компонента TQuickRep.

В дальнейшем в этой сетке располагаются составные части отчета, например, группы TQRBand и др. (рис. 20).

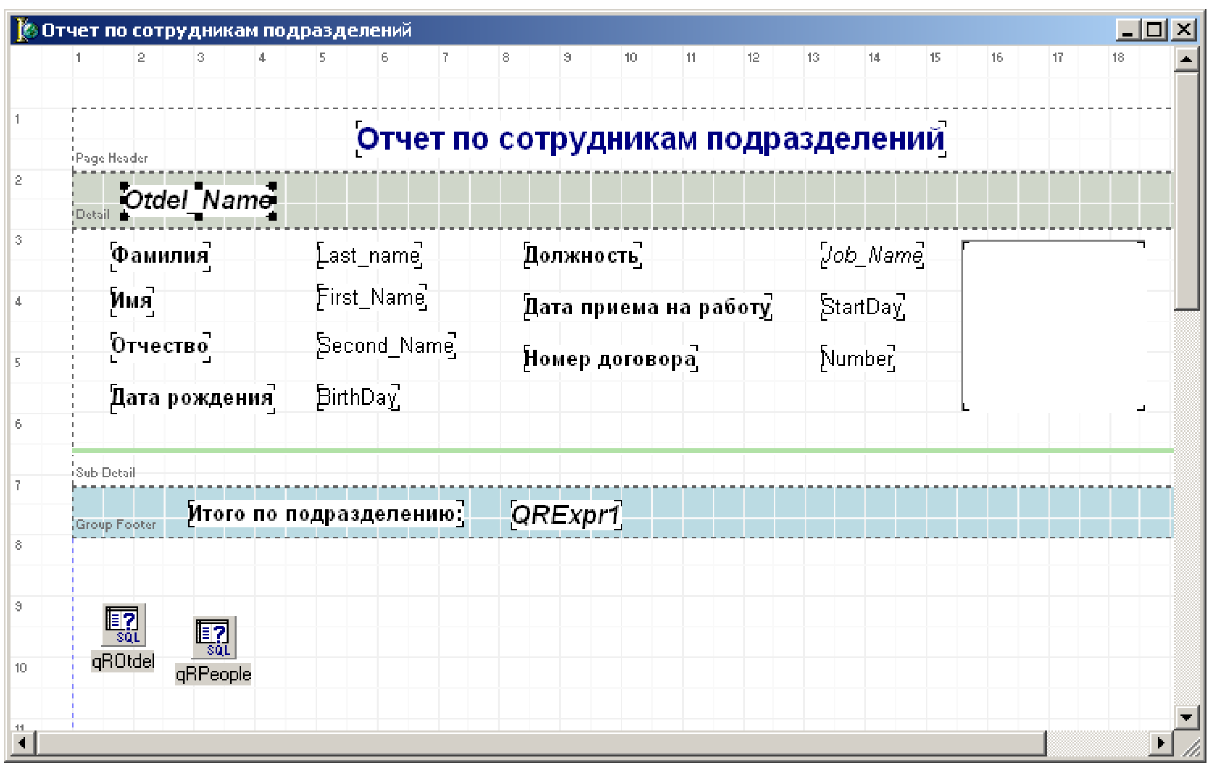


Рис. 20 — Сетка отчета с размещенными в ней компонентами отчета

Перечислим важнейшие свойства, методы и события компонента TQuickRep.

Свойство

property Bands : TQuickRepBands;

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

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

property DataSet: TDataSet;
указывает на набор данных, на основе которого и создается отчет. Обычно для выдачи отчета используется один НД.

Если нужно вывести связанную информацию из нескольких таблиц БД, ее объединяют в одном НД при помощи оператора SELECT. В этом случае в качестве НД для отчета может использоваться компонент TQuery. Информацию из нескольких связанных НД можно включать в отчет, если эти наборы данных связаны в приложении отношением Master-Detail. В этом случае в качестве НД отчета указывается Master-набор, а ссылка на соответствующие Detail-наборы осуществляется в компонентах TQRSubDetail.

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

property Frame : TQRFrame;
определяет параметры рамки отчета:

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

property Page: TQRPage;
определяет параметры страницы.

property PrinterSettings: TQuickRepPrinterSettings;
определяет параметры принтера.

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

Методы:

procedure NewPage;
выполняет переход на новую страницу. Может использоваться в обработчиках событий компонентов отчета BeforePrint или AfterPrint и не может — в обработчиках событий OnPrint, OnStartPage и OnEndPage.

procedure Preview;
выводит отчет в окно предварительного просмотра (рис. 21).

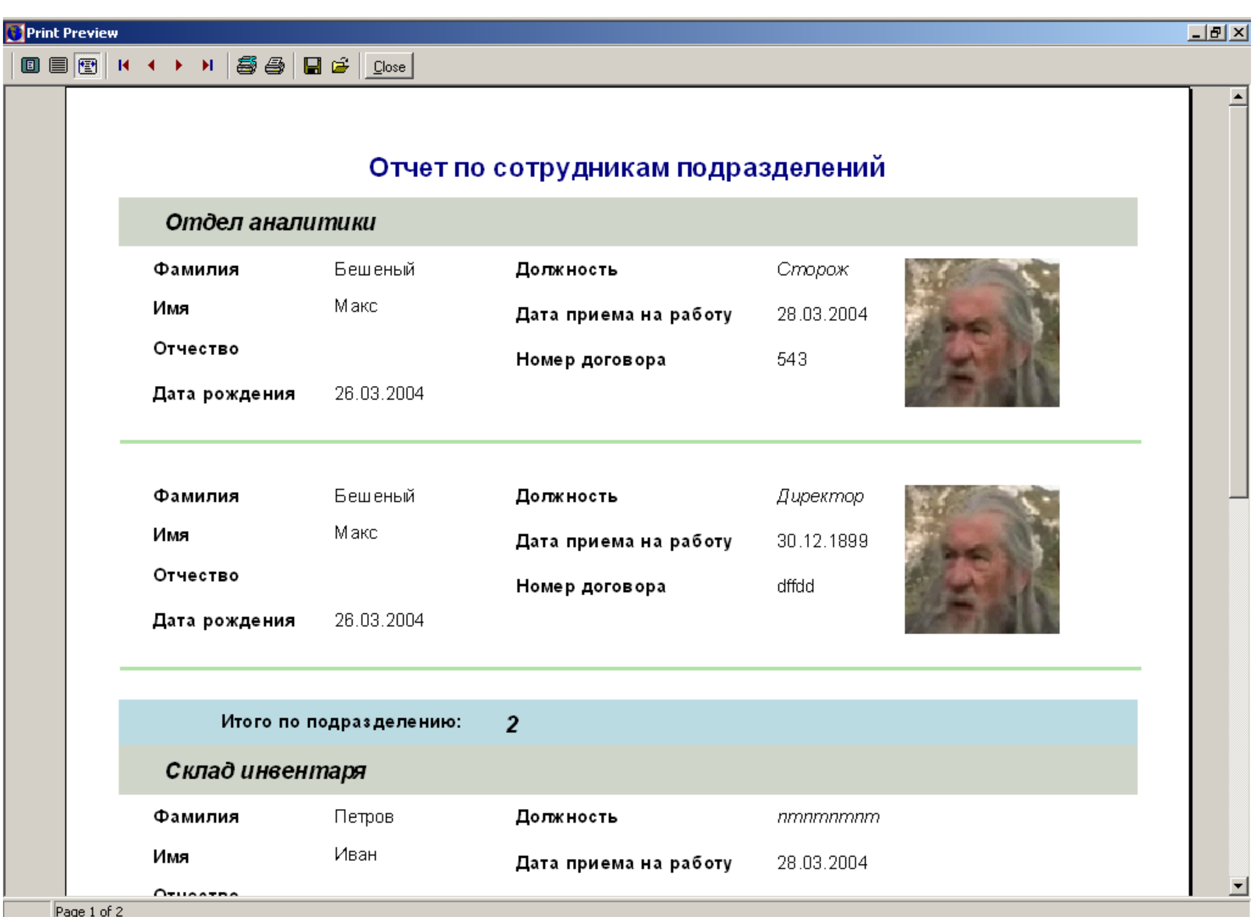


Рис. 21 — Окно предварительного просмотра отчета

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

– выбрать отчет при помощи мыши;

– нажать правую кнопку мыши;

– во всплывающем меню выбрать элемент Preview.

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

procedure Print;
печатает отчет на принтере.

procedure PrinterSetup;
обеспечивает установки параметров принтера.

События

property AfterPreview : TQRAfterPreviewEvent;
наступает после закрытия окна предварительного просмотра отчета.

property AfterPrint: TQRAfterPrintEvent;
наступает после вывода отчета на печать.

property BeforePrint: TQRBeforePrintEvent;
наступает в момент генерации отчета, до выдачи окна предварительного просмотра отчета и до вывода отчета на печать.

property OnEndPage : procedure(Sender : TObject);
наступает в момент подготовки к генерации последней страницы отчета.

property OnStartPage : procedure(Sender : TObject);
наступает в момент подготовки к генерации первой страницы отчета.

Компонент TQRBand

Компоненты TQRBand являются основными составными частями отчета и используются для размещения в них статического текста и данных. Месторасположение компонента в отчете и его поведение определяются свойством

property BandType : TQRBandType;

Ниже перечислены возможные значения этого свойства:

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

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

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

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

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

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

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

– *rbSubDetail* — компонент для выдачи детальной информации из подчиненного набора данных, при выводе в отчете информации из двух или более наборов данных, связанных в приложении при помощи механизма Master-Detail. Это значение присваивается компоненту автоматически, когда генерируется компонент TQRBand при размещении в форме компонента

TQRSubDetail. Программа не должна устанавливать это значение в свойство BandType;

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

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

Свойство

property Enabled : **Boolean;**

указывает, печатается в отчете (True) или нет (False) информация, содержащаяся в компоненте TQRBand.

Свойство

property ForceNewPage : **Boolean;**

указывает, должна ли информация в составе TQRBand всегда печататься с новой страницы (True) или нет (False).

Событие

property BeforePrint: **TQRBeforePrintEvent;**

наступает перед печатью информации, размещенной в области компонента TQRBand.

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

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

Компонент TQRBand, у которого в свойство BandType установлено значение rbPageHeader, используется для показа заголовка страницы. Он выводится для каждой новой страницы перед выводом другой информации.

Компонент TQRBand, у которого в свойство BandType установлено значение rbPageFooter, используется для показа подвала

страницы. Он выводится для каждой страницы после вывода любой иной информации.

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

Для компонента TQRBand для заголовка страницы (*rbPageHeader*) чтобы отчеркнуть линию вверху страницы необходимо установить в свойство компонента заголовка страницы Frame.DrawTop значение True, что обеспечивает вывод линии по верхнему краю области, занимаемой компонентом.

Аналогичным образом для компонента подвала страницы установив его свойство Frame.DrawBottom значение True, обеспечивается вывод линии по нижнему краю области, занимаемой компонентом.

Таким образом, вверху и внизу каждой страницы отчета выводятся линии.

Использование компонента TQRSysData для показа вспомогательной и системной информации

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

property Data : **TQRSysDataType;**

Ниже указаны возможные значения этого свойства:

– *qrsColumnNo* — номер текущей колонки отчета (для одноколоночного отчета всегда 1);

– *qrsDate* — текущая дата;

– *qrsDateTime* — текущие дата и время;

– *qrsDetailCount* — число записей в НД; при использовании нескольких НД — число записей в master-наборе. Для случая, когда НД представлен компонентом TQuery, эта возможность может быть недоступной, что связано с характером работы компонента TQuery, который возвращает столько записей, сколько необходимо для использования в текущий момент, а остальные предоставляет по мере надобности;

– *qrsDetailNo* — номер текущей записи в ИД. При наличии нескольких наборов — номер текущей записи в master-наборе;

- *qrsPageNumber* — номер текущей страницы отчета;
- *qrsPageCount* — общее число страниц отчета;
- *qrsReportTitle* — заголовок отчета;
- *qrsTime* — текущее время.

Группировки данных в отчете

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

Свойство FooterBand компонента TQRGroup содержит ссылку на компонент подвала группы.

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

Поскольку свойство Expression не визуализирует значения выражения, необходимо разместить в группе компонент TQRExr и определить значение его свойства Expression.

Часто внутри группы должны содержаться другие группы. В этом случае внутри одной группы выделяют другую группу посредством дополнительных компонентов TQRGroup.

Использование компонента TQRExpr для определения выражений

Выражение в отчетах формируется при помощи компонента TQRExpr.

Для того чтобы войти в редактор формул данного компонента, необходимо в инспекторе объектов выбрать свойство Expression и нажать кнопку в поле данных этого свойства.

Появится окно редактора формул (рис. 22).

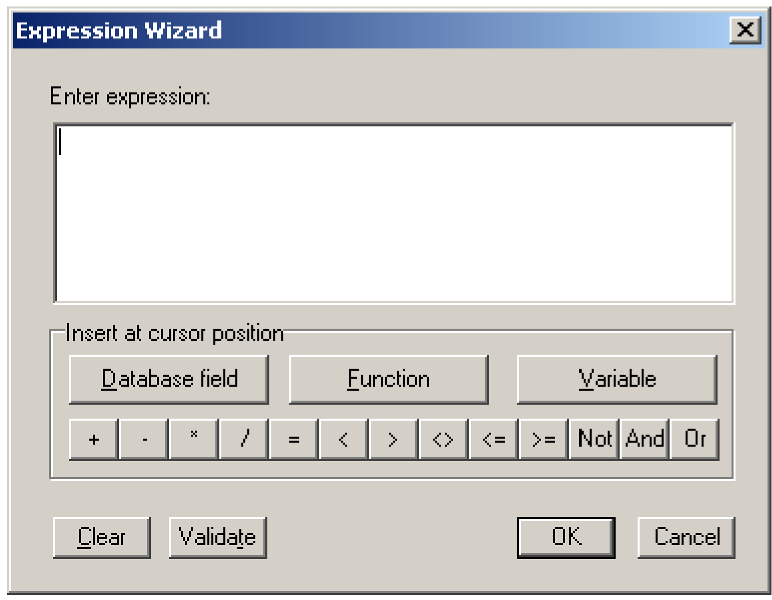


Рис. 22 — Окно редактора формул
компонента TQRExpr

Группа Function позволяет включить в выражение функцию, при этом выпадающий список Category определяет категорию функции (по умолчанию показываются все функции, значение ALL).

Группа Database Field позволяет включить в выражение поля набора данных, причем список Select Dataset определяет НД, а в списке Available Fields перечисляются поля текущего НД.

Группа Variable позволяет работать с различными переменными.

В группе без названия находятся кнопки, которые позволяют добавить константу, выражение, функцию или поле непосредственно в текст выражения.

Построение отчета на основе нескольких наборов данных, связанных в приложении как Master — Detail

Если необходимо выдавать отчет на основе более чем одной таблицы БД, можно поступить двумя способами:

1. В рамках компонента TQuery произвести соединение данных из нескольких таблиц БД в один НД, после чего определить в отчете нужные группировки.

2. Создать в приложении по одному НД на каждую таблицу БД, соединить эти наборы между собой связью Master-Detail (используя свойства MasterSource, MasterFields набора данных) и применить в отчете компонент (или несколько компонентов) TQRSubDetail для вывода информации из подчиненного (Detail) НД (или группы подчиненных НД); для вывода информации из основного (Master) НД, как и в обычных отчетах, применяется компонент TQRBand, у которого в свойстве BandType установлено значение rbDetail.

Компонент TQRSubDetail предназначен для показа информации в отчете из подчиненного НД. Его свойство

– property DataSet: TDataSet;

– указывает имя подчиненного НД, информация из которого будет выводиться в пространстве компонента TQRSubDetail. В остальном использование данного компонента аналогично использованию компонента TQRBand, у которого значение в свойство BandType установлено значение rbDetail.

Если необходимо определить заголовок и подвал для информации, группируемой в компоненте TQRSubDetail, следует воспользоваться свойством этого компонента.

property Bands : TQRSubDetailGroupBands;

Данное свойство обладает двумя подсвойствами, указывающими на наличие или отсутствие заголовка и подвала. Это :

property HasHeader : Boolean;
property HasFooter: Boolean;

Композитный отчет

Композитный (составной, сложный) отчет объединяет в себе несколько простых отчетов. При выдаче композитного отчета, входящие в его состав простые отчеты выводятся друг за другом.

Композитный отчет реализуется при помощи компонента TQRCompositeReport. В обработчике события OnAddReport ранее определенные простые отчеты добавляются в списковое свойство Report:

procedure TCompozitnyjOtchet.QRCompositeReportlAddReports(Sender: TObject);

```
begin
  WITH  QRCompositeReportl do begin
    Reports.Add(ManyGroup.QuickRepl);
    Reports.Add(Prostoj.QuickRepl);
  END;//with
end;
```

В приведенном выше обработчике композитный отчет составляется из двух отчетов: QuickRepl (определенный в форме ManyGroup) и QuickRepl (определенный в форме Prostoj). Печать композитного отчета или его предварительный просмотр осуществляется так же, как для простых отчетов, например

```
QRCompositeReportl.Preview;
```

Разработка варианта отчетной формы по информации из БД отдела кадров

Создадим отчет, состоящий из информации по сотрудникам подразделений.

Будем создавать отчет на основе нескольких наборов данных, связанных в режиме «главный»-«подчиненный». 1 отдел — много сотрудников.

1. Разместим в форме компоненты qROtdel: TQuery и qRPeople: TQuery.

Набор данных qROtdel: TQuery формируется следующим оператором SELECT:

```
select * from Otdel
```

Набор данных qRPeople : TQuery формируется следующим оператором SELECT:

```
select p.Last_name, p.First_Name, p.Second_Name, p.BirthDay,
 j.Job_Name, j.Oklad, c.Number, c.StartDay, p.Photo
from people p, job j , contract c
where (c.Id_People = p.Id_People) and (c.Id_Job=j.Id_Job) and
(c.Id_Otdel=:idotd)
```

Будем показывать данные в отчете в режиме «главный»-«подчиненный». В нашем случае будем устанавливать связь между подразделениями «главный» и их сотрудниками «подчиненный».

В нашем случае мы в качестве наборов данных используем компоненты TQuery, а следовательно, не можем использовать свойства НД TTable MasterSourse и MasterFields.

Для установки режима «главный»-«подчиненный» проведем подстановку соответствующих полей связи Id_Otdel в qROtdel и Id_Otdel в qRPeople (значение по параметру). И в событии главного набора данных qROtdelAfterScroll пропишем:

```
procedure TfmReport.qROtdelAfterScroll(DataSet: TDataSet);
begin
  if qRPeople.Active then qRPeople.Close;

qRPeople.Params[0].AsInteger:=qROtdel.FieldByName('id_otdel').AsIn
teger;
  qRPeople.Open;
end;
```

Добавим в редактор полей НД qRPeople все поля, чтобы не возникало проблем с подстановкой параметра в запрос. Затем в свойстве qRPeople.Params[0] для параметра idotd установим свойство qRPeople.Params[0].DataField в ftInteger.

2. Разместим в форме компонент TQuickRep (имя QuickRepl).

Установим в свойство DataSet отчета значение qROtdel, назначив таким образом отчету главный НД.

3. Добавим в отчет компонент TQRBand (имя QRBandl).

Свойство BandType компонента QRBandl по умолчанию установлено в значение rbTitle, то есть компонент QRBand1 определяет заголовок отчета. Установим свойство BandType компонента QRBandl в rbPageHeader.

Разместим в пространстве отчета, занимаемом компонентом QRBand1, компонент TQRLabel (статический текст) с именем QRLabel1. Установим в свойство Caption этого компонента зна-

чение 'Отчет по сотрудникам подразделений' и установим в свойстве Font жирный шрифт ([fsBold]) высотой 14 (Size) пунктов цвета clNavy.

4. Добавим в отчет компонент TQRBand (имя QRBand2).

Свойство BandType компонента QRBand2 установим в rbDetail. Установим при этом свойство QRBand2.Color := $00C9D6CF (добавленный предварительно в набор цветов цвет, можно выбрать из стандартной палитры).

Разместим в пространстве отчета, занимаемом компонентом QRBand2, компонент TQRDBText (статический текст из поля набора данных) с именем QRDBText2. Установим в свойство Dataset этого компонента значение qROtdel, а в свойство DataField Otdel_Name. Установим при этом в свойстве Font жирный шрифт с курсивом ([fsBold,fsItalic]) высотой 12 (Size) пунктов без изменения цвета. Кроме того, установим свойство Transparent (прозрачность фона надписи) в True.

5. Добавим в отчет компонент TQRSubDetail (имя QRSubDetail1).

Установим свойство QRSubDetail1.DataSet в qRPeople.

Разместим в пространстве отчета, занимаемом компонентом QRSubDetail1 компоненты TQRDBText (статический текст из поля набора данных) рядом с которыми разметим соответствующие компоненты TQRLabel (статический текст):

– QRDBText2 (свойство Dataset этого компонента в значение qRPeople, а свойство DataField в Last_name) и QRLabel2 (свойство Caption этого компонента в значение «Фамилия»);

– QRDBText3 (свойство Dataset этого компонента в значение qRPeople, а свойство DataField в First_Name) и QRLabel3 (свойство Caption этого компонента в значение «Имя»);

– QRDBText4 (свойство Dataset этого компонента в значение qRPeople, а свойство DataField в Second_Name) и QRLabel4 (свойство Caption этого компонента в значение «Отчество»);

– QRDBText5 (свойство Dataset этого компонента в значение qRPeople, а свойство DataField в BirthDay) и QRLabel5 (свойство Caption этого компонента в значение «Дата рождения»);

– QRDBText6 (свойство Dataset этого компонента в значение qRPeople, а свойство DataField в Job_Name) и QRLabel6 (свойство Caption этого компонента в значение «Должность»);

– QRDBText7 (свойство Dataset этого компонента в значение qRPeople, а в свойство DataField в StartDay) и QRLabel7 (свойство Caption этого компонента в значение «Дата приема на работу»;

– QRDBText8 (свойство Dataset этого компонента в значение qRPeople, а свойство DataField в Number) и QRLabel8 (свойство Caption этого компонента в значение «Номер договора»).

Для компонентов TQRDBText установки шрифтов не меняем.

Для QRDBText6 установим в свойстве Font курсив ([fsItalic]) высотой 10 (Size) пунктов без изменения цвета.

Для компонентов TQRLabel установим в свойстве Font жирный шрифт ([fsBold]) высотой 10 (Size) пунктов без изменения цвета.

Разместим в пространстве отчета, занимаемом компонентом QRSubDetail1 компонент TQRDBImage (имя QRDBImage1) для показа фотографий сотрудников. Для QRDBImage1 установим в свойство Dataset этого компонента значение qRPeople, а свойство DataField в Photo.

Чтобы записи о сотрудниках отдела были разделены линией положим на пространстве отчета, занимаемом компонентом QRSubDetail1, компонент TQRShape (имя QRShape1). Зададим свойство Color := $00A6E1B1 (добавленный предварительно в набор цветов цвет, можно выбрать из стандартной палитры), свойство Height в 4.

6. Добавим в отчет компонент TQRBand (имя QRBand3).

Свойство BandType компонента QRBand3 установим в rbGroupFooter.

Установим при этом свойство QRBand3.Color := $00E3DBBB (добавленный предварительно в набор цветов цвет, можно выбрать из стандартной палитры).

Разместим в пространстве отчета, занимаемом компонентом QRBand3, компонент TQRLabel (статический текст) с именем QRLabel9. Установим в свойство Caption этого компонента зна-

чение 'Итого по подразделению:' и установим в свойстве Font жирный шрифт ([fsBold]) высотой 10 (Size) без изменения цвета. Кроме того, установим свойство Transparent (прозрачность фона надписи) в True.

Разместим в пространстве отчета, занимаемом компонентом QRBand3, компонент TQRExpr (имя QRExpr1) для выдачи вычисляемого значения Количество сотрудников в отделе. Установим в свойство Font жирный курсив ([fsBold,fsItalic]) высотой 12 (Size) без изменения цвета. Кроме того, установим свойство Transparent (прозрачность фона надписи) в True.

Так как в QRBand3 со свойством (rbGroupFooter) будем выводить информацию по подразделениям, то установим для компонента QRSubDetail1 свойство FooterBand в QRBand3.

Для вычисления значения количества сотрудников в отделе через компонент QRExpr1 на событие QRExpr1Print запишем:

```
procedure TfmReport.QRExpr1Print(sender: TObject; var Value: String);
begin
  Value:=IntToStr(qRPeople.RecordCount);
end;
```

Отчет сформирован (см. рис. 20).

Покажем сформированный отчет в режиме предварительно просмотра (правая кнопка мыши, нажатие кнопки в приложении, на акцию и пр.):

```
fmReport.QuickRep1.Preview;
```

См. рис. 21.

Например, для акции actReport в момент ее выполнения (событие Execute) можно записать:

```
procedure TForm1.actReportExecute(Sender: TObject);
begin
  if fmReport.qRPeople.Active then fmReport.qRPeople.Close;
fmReport.qRPeople.DatabaseName:=ExtractFilePath(Application.
ExeName)+'db';
```

```
fmReport.qRPeople.Open;
if fmReport.qROtdel.Active then fmReport.qROtdel.Close;
fmReport.qROtdel.DatabaseName:=ExtractFilePath(Application. Ex-
eName)+'db';
fmReport.qROtdel.Open;

fmReport.QuickRep1.Preview;
end;
```

По аналогии формируются и другие отчеты.

ЛАБОРАТОРНАЯ РАБОТА № 8
ПОСТРОЕНИЕ ГРАФИКОВ ПО ДАННЫМ БД

Задание

1. Изучение компонента TDBChart.
2. Создание различных графиков по информации из БД.
3. Разработка формы с реализацией построения графиков для БД отдела кадров.

Создание графика. Компонент TDBChart

Компонент TDBChart предназначен для построения графиков по информации из БД.

Для того чтобы создать график, необходимо поместить на форму компонент TDBChart. В форме будет создана заготовка (рис. 23). Затем необходимо щелкнуть по этой заготовке 2 раза.

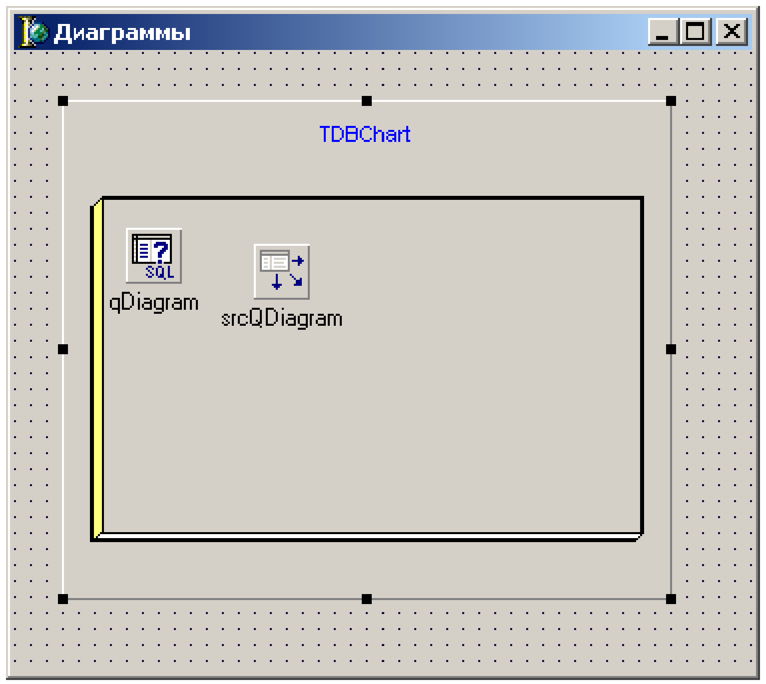


Рис. 23 — Заготовка графика в форме

Будет произведен переход в редактор графика. В среде этого редактора; можно установить свойства графика и его серий. Содержимое редактора графика представляет собой табулированный

блокнот. Для нового графика первой всегда показывается закладка Chart и для страницы Chart — закладка Series (рис. 1.2).

Каждая из закладок на странице Chart предназначена для установки параметров того или иного компонента графика.

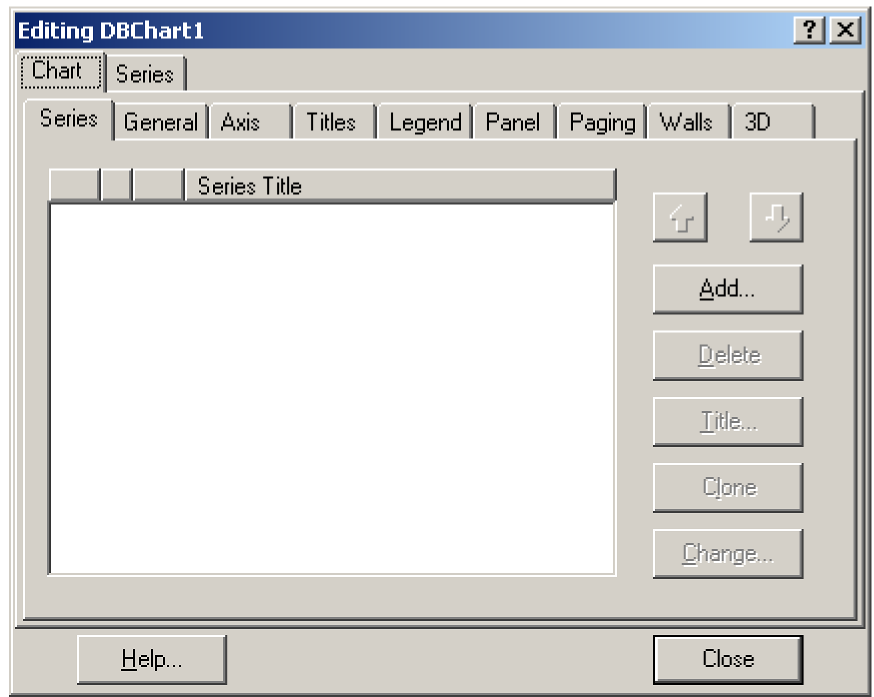


Рис. 24 — Редактор графика

Series — содержит серии графика. Серией называется набор точек графика. На графике серии соответствует отдельная линия или ряд столбцов. Если в графике несколько серий, будет визуализировано несколько линий или рядов столбцов.

General — устанавливает общие параметры графика, такие как объемность графика, отступы от краев, возможность увеличения (Zoom) и др.

Axis — устанавливает свойства осей (рис. 25).

В области Show Axis определяется, для какой оси устанавливаются параметры — левой, правой, верхней или нижней. На странице, определяемой закладкой Scales, устанавливаются свойства масштаба значений по оси. Automatic устанавливает автоматическое масштабирование данных по оси — минимум и максимум вычисляются динамически, исходя из текущих значений серии. При отмене автоматического масштабирования можно установить автоматическое масштабирование минимального (Mini-

mum) или максимального (Maximum) значения (отметка Auto). Для установки значения максимума и (или) минимума вручную следует нажать соответствующую кнопку Change. Шаг масштаба по оси выбирается автоматически, если в Desired Increment установлено значение 0. Установить фиксированное значение шага можно, нажав кнопку Change. Закладка Title позволяет установить текст заголовка по оси, угол расположения заголовка и шрифт, которым заголовок выводится. Закладка **Labels** задает параметры меток для оси. Закладка **Tiks** устанавливает параметры самой линии оси.

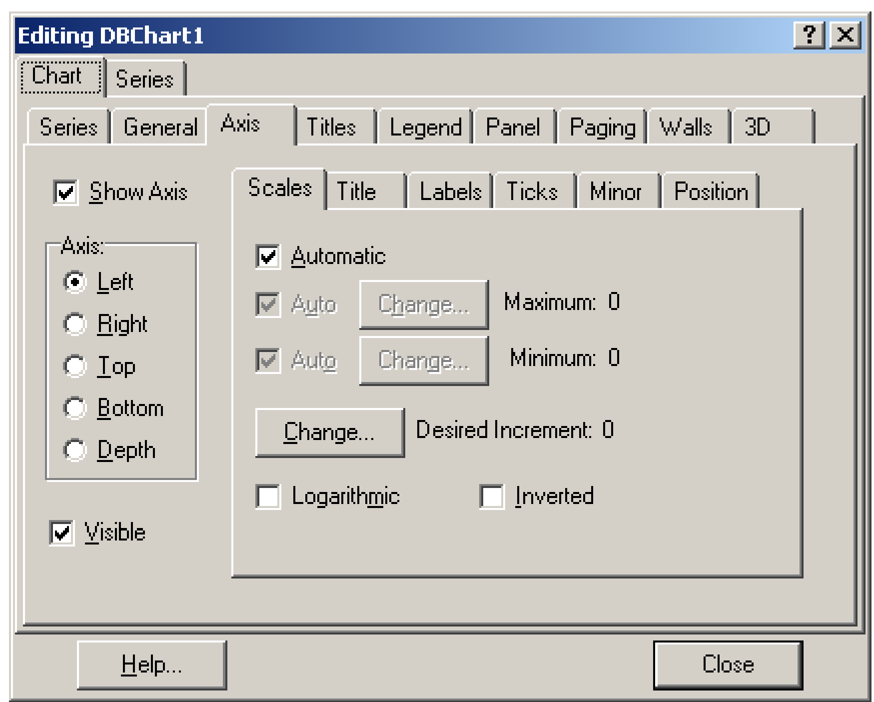


Рис. 25 — Редактор графика — окно установки свойств осей

Titles — определяет заголовок графика, шрифт, выравнивание и др.

Legend — задает параметры легенды. Легенда — область графика, где приводится информация о графике и служит для пояснения графика.

Panel — определяет параметры панели, на которой располагается график.

Paging — устанавливает параметры многостраничного графика.

Walls — задает «стенку» графика.

Добавление серии в график и установка свойств серии в редакторе графика

На графике одновременно может располагаться несколько серий. В большинстве случаев их значения строятся по одинаковому закону и две и более серий одновременно показываются в графике для сравнения.

Чтобы добавить в график серию, следует на странице Chart, (закладка Series) нажать кнопку Add. После этого появится окно выбора типа серии (рис. 26).

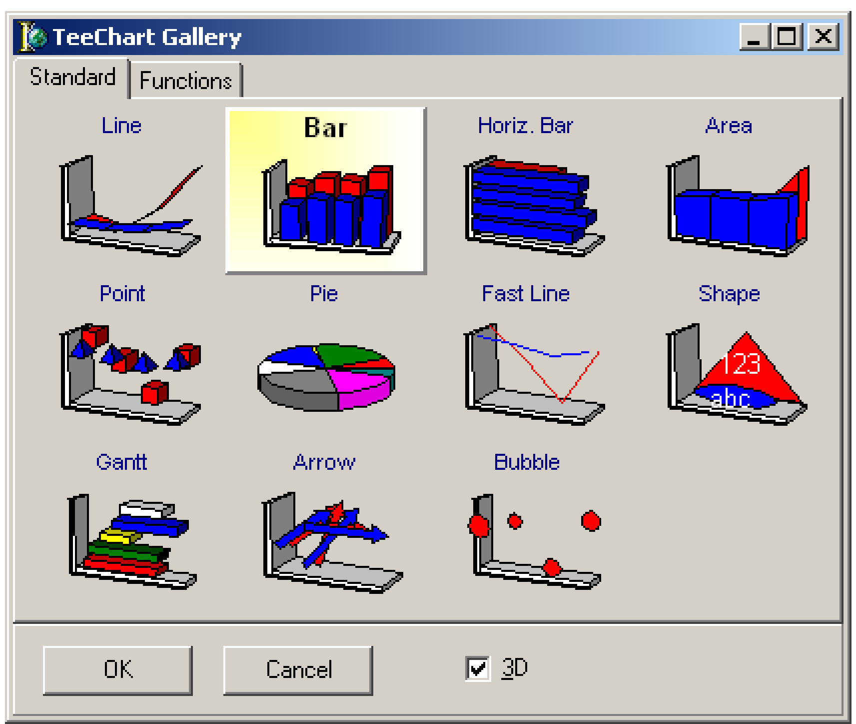


Рис. 26 — Редактор графика — окно выбора типа серий

После выбора типа серии в график добавляется компонент, дочерний от базового типа TChartSeries — TLineSeries, TBarSeries, TPieSeries и т.д. Выберем серию типа Pie и нажмем Ок. В окне страницы Chart (закладка Series) будет показана серия (рис. 27).

Кнопка Add может использоваться для добавления других серий, кнопка Delete — для удаления текущей серии. После нажатия кнопки Title можно определить заголовок серии, кнопки

Clone — создать новый экземпляр такой же серии в этом же графике, кнопки Change — изменить тип текущей серии.

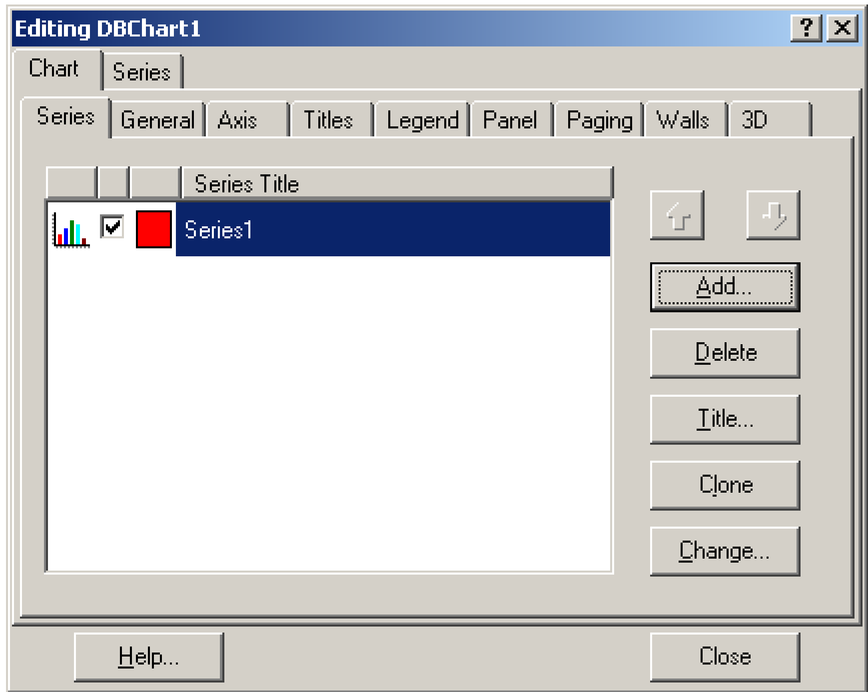


Рис. 27 — Редактор графика — список серий графика

Перейдем с закладки Chart на закладку Series. На этой странице представлен блокнот с закладками Format, General, Marks, DataSource. Рассмотрим свойства серии, которые можно установить на страницах, соответствующих этим закладкам.

Выбор источника данных

Несомненно, главные свойства серии можно определить на странице DataSource. На ней определяется источник данных для серии. Выпадающий список ниже закладки позволяет определить тип источника данных для серии:

No Data — серии не назначается источник данных. Далее мы собираемся сделать, что программно. Кроме того, заготовленный шаблон серии может в разное время использоваться для показа данных из разных источников, которые мы также собираемся переключать программно во время выполнения.

Random Values — набор случайных чисел. Бывает полезен при формировании заготовки серии, источник данных которой мы собираемся установить позднее.

Function — функция (Copy, Average, Low, High, Divide, Multiply, Subtract, Add) — служит для построения графиков на основании данных в двух или более сериях.

DataSet — позволяет указать НД, значения полей (столбцов) которого будут использоваться для формирования точек серии. В качестве НД могут выступать компоненты TTable, TQuery, TClientDataSet.

Выберем DataSet и из выпадающего списка выберем компонент qDiagram, ранее расположенный на нашей форме. qDiagram — набор данных, который формируется на основании запроса «Выдать информацию о количестве сотрудников в подразделениях».

Вид оператора SELECT для данного запроса имеет вид:

select o.Otdel_name as Подразделение, count(c.id_people) as Kolvo

from contract c, otdel o

where c.id_otdel=o.id_otdel

group by c.id_otdel, o.Otdel_name

Будем строить распределение количества сотрудников по подразделениям. Для круговой диаграммы укажем (рис. 28).

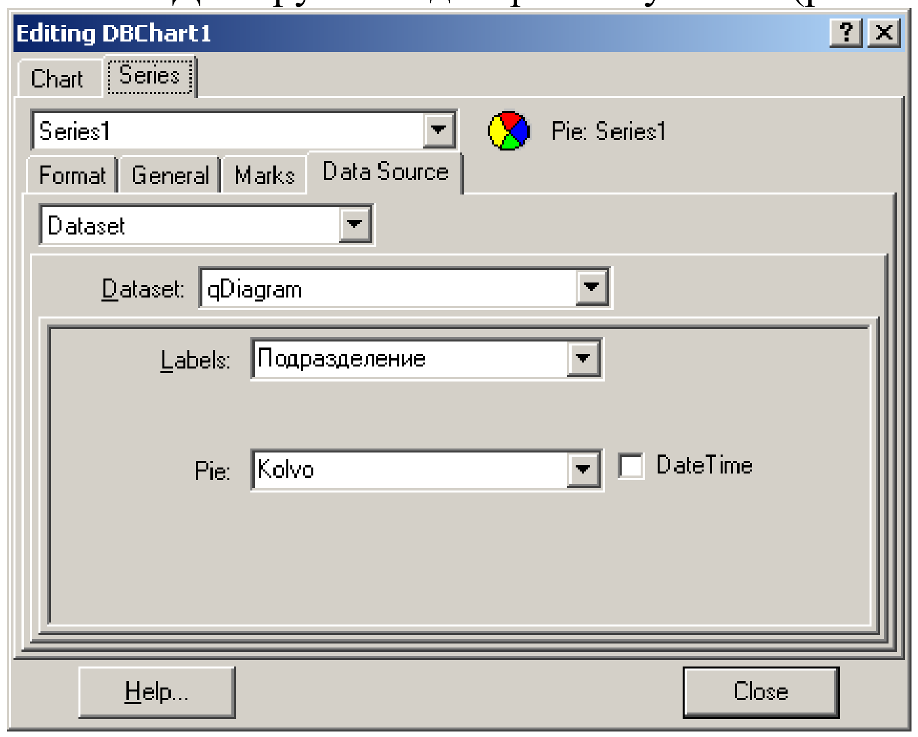


Рис. 28 — Редактор графика — определение источников для формирования координат графика серии типа Pie (круговая)

Для построения блочной диаграммы укажем (рис. 29).

Рис. 29 — Редактор графика — определение источников для формирования координат графика серии типа Bar (блочная)

Отметим, что не все типы серии требуют значений по осям Y или X. Для серий типа **Pie, Bar** можно указывать значения по **одной** из осей и значения меток Labels. В качестве меток могут использоваться символьные поля и поля типа даты и времени. Для примера приведем графики распределения сотрудников по подразделениям (рис. 28 и 29).

Серия типа Bar может содержать точки, сформированные как по осям X, Y, так и по оси Y (Bar) и меткам Labels (рис. 29).

После того, как мы указали источник данных и поля для формирования значений по осям X и Y, нажмем кнопку Close и выйдем из редактора графика. На ранее пустой панели будет построен график (рис. 30, 31).

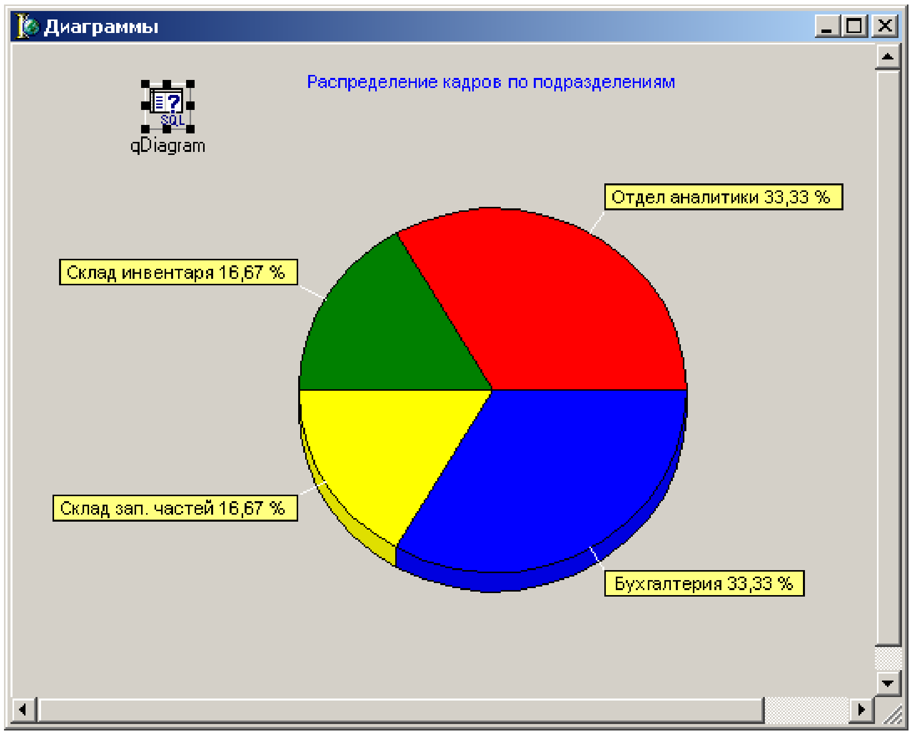


Рис. 30 — График серии типа Pie

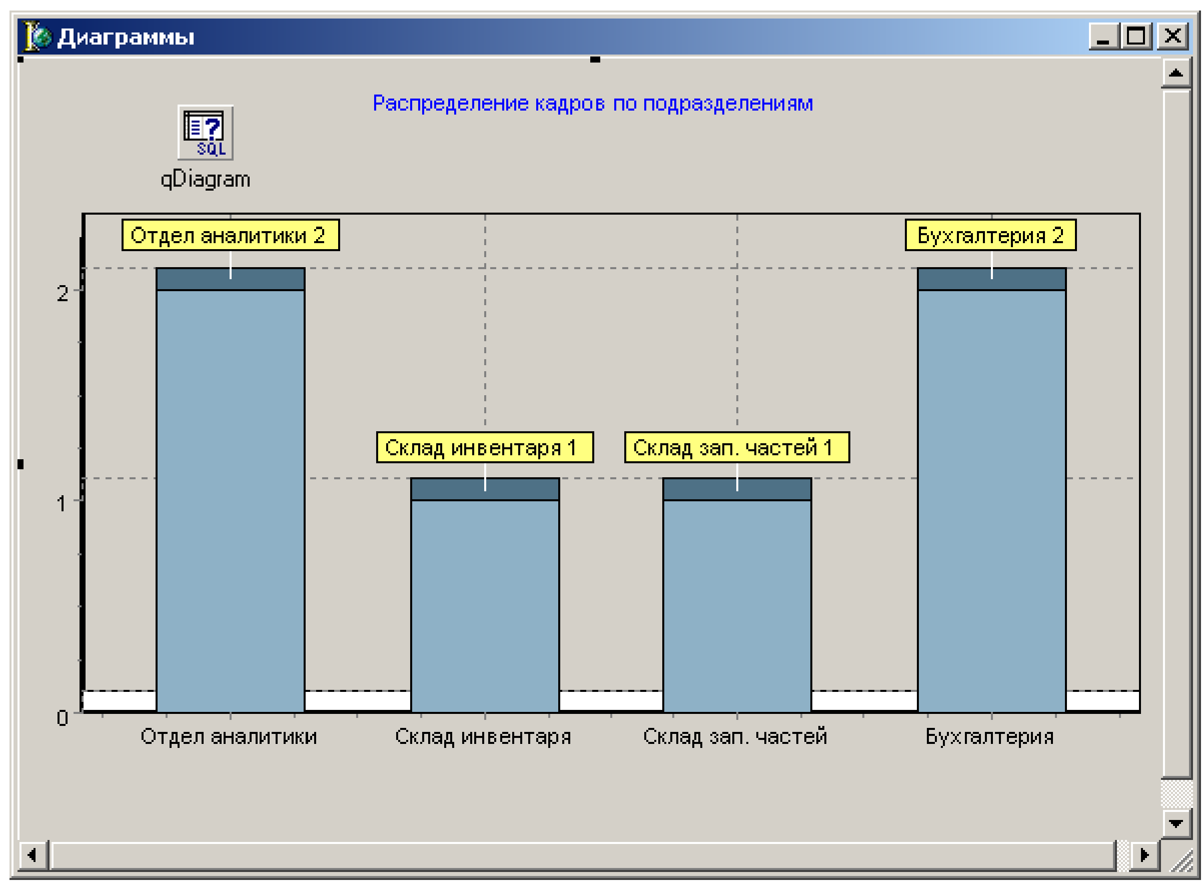


Рис. 31 — График серии типа Bar

Если вернуться в редактор графика, на странице Series можно, помимо DateSource, увидеть закладки Format, General, Marks.

Их назначение:

Format — определяет свойства палитры, линий графика и т.д.

General — задает форматы данных.

Marks — устанавливает марки — значения в рамке над точками серии.

На странице Series (закладка Data Source) в качестве источника данных можно определить функцию. Функция обычно используется для показа отношений между другими сериями.

Например, пусть для графика определены две серии. Требуется определить третью серию, которая показывала бы разницу межу первыми двумя. Для этого добавим в график новую серию и на странице Series в закладке Data Source выберем Function. В диалоговом окне определения функций выберем тип функции — Subtract (вычитание). В области Source Series Available показаны серии, присутствующие в графике. Используя кнопку >>, переместим две упомянутые выше серии в область Selected.

Добавление серии во время выполнения

Число серий, присутствующих в текущий момент в графике, во время выполнения можно определить при помощи метода компонента TDBChart

function SeriesCount: Longint;

Список серий графика содержится в его свойстве

property Series[Index:Longint]:TChartSeries;

где Index лежит в диапазоне 0…SeriesCount-1, поскольку отсчет серий идет с нуля.

Метод

procedure RefreshData;

обновляет данные в серии из наборов данных, которые служат их источником.

Покажем, как можно добавить серию **в** график во время выполнения:

```
var
MySeries : TBarSeries;
…………………………..
```

```
MySeries := TBarSeries.Create(Self);
MySeries.ParentChart := DBChartl;
MySeries.SeriesColor := clGreen;
MySeries.DataSource := Queryl;
MySeries.XLabelsSource := 'MES';
MySeries.YValues.ValueSource := 'S';
MySeries.Active := True;
MySeries.Title := Tablel.Value;
```

В приведенном примере создается серия типа Ваг (вертикальная столбчатая диаграмма), в качестве родительского графика ей назначается DBChartl. В качестве источника данных назначается Queryl. Для формирования значений серии используются поля компонента Queryl: S (по оси Y) и MES (в качестве Labels). Затем серия активизируется (после этого она становится видна в графике) и ее заголовку присваивается наименование из Table1.

Работа с сериями. Компонент TChartSeries

Компонент TChartSeries является родительским типом для серий графика (для TLineSeries, TAreaSeries, TPointSeries, TBarSeries, THorizBarSeries, TPieSeries, TChartShape, TFastLineSeries, TArrowSeries, TGanttSeries, TBubbleSeries).

Свойства компонента TChartSeries

property Active : Boolean; — активизирует (показывает) серию в графике (значение True) и дезактивизирует (скрывает) серию (False). Например:

DBChartl.Series [0] .Active :=True;

property DataSource : TComponent; — ссылается на компонент типа НД (TTable, TQuery, TClientDataSet) или на другую серию, откуда берутся данные для показа в серии. Например:

DBChartl.Series[0].DataSource := Query2;

property HorizAxis: **THorizAxis;** — указывает, какая горизонтальная ось будет использована для серии. Значения: aTopAxis — верхняя горизонтальная ось; aBottomAxis — нижняя горизонтальная ось.

property Marks : TSeriesMarks; — описывает свойства марок серии. Т.е. значений в прямоугольниках, рисуемых для каждого значения серии. Свойства объекта Marks:

– **property Arrow : TChartPen;** — задает свойства пера, рисующего марку. Свойства объекта Arrow:

– **property Color : TColor;** — цвет линий;

– **property Mode: TPenMode;** — способ рисования линий;

– **property Style: TPenStyle;** — стиль линий;

– **property Visible: Boolean;** — видимость линий;

– **property Width : Integer;** — задает ширину линий;

– **property ArrowLength : Integer;** — длина в пикселях линии, соединяющей марку с соответствующим изображением элемента серии. По умолчанию 16;

– **property BackColor: TColor;** — определяет цвет фона марки. По умолчанию S80FFFF (желтый);

– **property Clip: Boolean;** — если содержит True, марки не могут накладываться на другие элементы графика (на легенду, метки осей и т.д.);

– **property Font : TFont;** — определяет шрифт, которым выводится информация внутри марки;

– **property ParentSeries : TChartSeries;** — содержит указатель на серию, к которой принадлежат марки;

– **property Style : TSeriesMarksStyle;** — определяет содержимое марки. По умолчанию smsLabel. В обработчике события TChartSeries.OnGetMarkText можно переопределить значения, принятые по умолчанию. Например:

DBChartl.Series[0].Style := smsLabelValue;

Возможные значения свойства **Style**:

– **smsValue** — значения по осиУ (YValue), за исключением THorizBarSeries (XValue). Например, «9087»;

– **smsPercent** — процентное значение, например «44%»; для форматирования процентного значения также используется свойство TChartSeries.PercentFormat;

– **smsLabel** — показывает метку, ассоциированную с точкой графика, например «Сахарный песок» (при построении графика продаж по товарам); в том случае, если метки со значениями не ассоциированы, в марках выводятся сами значения;

– **smsLabelPercent** — показывает метку и процентное значение, например «Сахарный песок 44%». ***См. рисунок 1.8***;

– **smsLabelValue** — показывает метку и значение, например «Сахарный песок 9087». ***См. рисунок 1.9***;

– **smsLegend** — показывает один из элементов легенды графика, список возможных значений доступен через свойство TChartLegend.TextStyle;

– **sms Percent Total** — показывает процентное значение и общую сумму, от которой оно взято, например «44% от 20563».

– **smsLabelPercentTotal** — показывает метку, процентное число и общую сумму, например «Сахарный песок 44% от 20563»;

– **smsXValue** — показывает значение по оси X (XValue), например «01.02.1997»;

– **property Transparent: Boolean;** — значение True определяет, что цвет фона марки не используется (в качестве фона используется «прозрачный цвет»); по умолчанию False;

– **property Visible : Boolean;** — определяет, видимы ли (True) или нет (False) марки на графике;

– **property ParentChart : TCustomChart;** — указывает компонент TDBChart, к которому принадлежит серия. Изменение этого свойства позволяет во время выполнения добавлять в график новые серии, показывать серии в других графиках. Например:

```
var
MySeries : TBarSeries;
……………………………
MySeries := TBarSeries.Create(Self);
MySeries.ParentChart := DBChartl;
```

– **property PercentFormat : String;** — определяет формат показа процентных значений;

– **property RecalcOptions: TSeriesRecalcOptions;** — указывает перечень событий, приводящих к пересчету значений серии (учитывается только для серий, свойство DataSource которых указывает на другую серию) по умолчанию [rOnDelete, rOnModify, rOnlnsert, rOnClear];

– **property SeriesColor**: **TColor;** — определяет цвет, которым выводятся значения серии в графике. Например:

DBChartl.Series[0].SeriesColor := clBlue;

– **property ShowInLegend: Boolean;** — определяет, показывать ли (True) легенду или нет (False) по умолчанию True;

– **property Title: String;** — определяет заголовок серии; по умолчанию заголовок отсутствует, но он может быть назначен в редакторе графика (кнопка Title в окне Series). Например:

DBChartl.Series[0].Title := Editl.Text;

– **property ValueColor[Index:LongInt]:TColor;** — массив, определяет цвет элемента серии с номером Index, например,

DBChartl.Series[0].ValueColor[2] := clAqua;

– **property ValueFormat : String;** — определяет формат показа значений серии; при прорисовке осей используется для форматирования меток, при прорисовке серии используется для форматирования значений, показываемых в марках;

– **property ValueMarkText[Index:Longint]:String;** — массив значений, выводимых в марках серии;

– **property VertAxis** : **TVertAxis;** — определяет местоположение вертикальной оси — слева на графике (aLeftAxis) или справа (aRightAxis);

– **property XLabel[Index:Longlnt]: String; —** массив, хранящий метки серии по оси X**;** Index должен находиться в диапазоне 0…Count -1;

DBChartl.Series[0].XLabel[2] := Edit2.Text;

- property XLabelsSource: String; — имя поля НД (или иного источника значений для серии), определяемого в свойстве DataSource. Содержимое этого поля служит для отображения значений по оси X. Поле должно быть типа, к которому применяется метод AsString. Если значение свойства опущено, значения по оси X не выводятся. Например:

DBChartl.Series[0].XLabelsSource := 'MES';

– **property XValue[Index:LongInt] : Double;** — возвращает значение в списке XValues (см. ниже) с индексом Index (значение в диапазоне 0… Count -1);

– **property XValues:TChartValueList;** — хранит значения серии по оси X. Значения из этого списка НЕЛЬЗЯ удалять, добавлять и т.д. напрямую. Для этого следует воспользоваться со-

ответствующими методами компонента TChartSeries. Могут быть полезны следующие свойства ***TChartValueList***:

– **property Value[Index:LongInt]: Double;** — обеспечивает доступ к элементу серии с индексом Index (значение в диапазоне 0… Count -1). Например:

DBChartl.Series[0].YValues.Value[2] := StrToFloat (Edit3.Text);

DBChartl.Series[0].Repaint;

– **property ValueSource : String;** — указывает источник данных для формирования значений по оси X. В зависимости от того, каков источник данных для серии (свойство DataSource компонента TChartSeries), может содержать:

1) имя поля — числового типа, типа даты, времени, даты и времени; в этом случае свойство серии DataSource должно ссылаться на НД (TTable, TQuery, TClientDataSet), например:

DBChartl.Series[0].DataSource := Query2;

DBChartl.Series[0].XValues.ValueSource := 'Pole1'

при этом необходимо помнить, что данные будут взяты в серию только из открытого НД; если НД закрыт, то получение данных будет отложено до открытия НД;

2) имя существующего TChart ValueList из другой серии; в этом случае свойство DataSource серии должно ссылаться на другую серию, например:

DBChartl.Series[0].DataSource := DBChart2.Series[4]; DB-Chartl.Series[0].XValues.ValueSource := ‘X';

Свойства

property YValue[Index:LongInt] : Double;

property YValues: TChartValueList;

аналогичны свойствам XValue и XValues и используются для вертикальной оси.

Методы компонента TchartSeries

function AddXY(Const AXValue, AYValue: Double; Const AXLabel: String; AColor: TColor): Longlnt;

Добавляет новую точку в серию. Параметры AXValue и AYValue содержат соответственно значения по осям X и Y. Параметр AXLabel содержит метку для добавляемой точки серии.

Параметр AColor определяет цвет. Функция возвращает позицию новой точки в серии. Например:

DbChartl.Series[0].AddXY(TmpX,TmpY,TmpLabel,clAqua);

function AddY(Const AYValue: Double; Const AXLabel: String; AColor: TColor):

Longlnt;

Добавляет в серию новое значение по оси X. Применяется для тех серий, в которых график строится по X и меткам значений по X (например, Pie, Bar). Назначение параметров такое же, как у метода AddXY.

procedure AssignValues(Source: TChartSeries); — копирует все точки из серии Source в текущую серию.

procedure CheckDataSource; — обновляет точки в серии, независимо от того, какой компонент является источником данных — набор данных или другая серия. Обновление производится по текущим данным источника. Метод рекомендуется вызывать в случае изменений данных в источнике.

procedure Clear; — удаляет все значения из серии; если вслед за этим не занести новых точек, будет показываться пустой график.

procedure ColorRange(AValueList: TChartValueList ;Const FromValue, ToValue:

Double; AColor: TColor);

Изменяет цвет указанного диапазона точек серии. AValueList — либо XValues, либо YValues. FromValue указывает начальное, а ToValue конечное значение в списке AValueList. AColor — новый цвет. Например:

```
WITH DbChartl.Series[0] do begin
  ColorRange(XValues,XValues.Value[2], XVa-
lues.Value[2],clAqua);
END;//with
```

function Count : Longlnt; — возвращает число точек в серии. Например, поместить все значения по X и Y точек серии в ListBoxl:

```
ListBoxl.Items.Clear;
WITH DbChartl.Series[0] do begin
  FOR i := 0 TO Count - 1 do
    ListBoxl. Items.Add (FloatToStr(XValues.Value[i]) + ‘ ‘ +
                                    FloatToStr(YValues.Value[i]));
END;//with
```

procedure Delete(ValueIndex : Longlnt); — удаляет из серии точку с номером Valuelndex. График, к которому принадлежит серия, автоматически перерисовывается. Например:

DBChartl.Series[0].Delete(4);

procedure DoSeriesClick(ValueIndex:LongInt; Button: TMouseButton; Shift: TShiftState; X, Y: Integer); virtual; — инициирует наступление события OnClick.

function GetCursorValuelndex : Longlnt; — возвращает индекс точки серии в

TChartValueList, ближе всего к которой расположен курсор мыши. Если такую точку определить не удается, возвращается — 1. Например, в следующем фрагменте Label1.Caption будет содержать индекс ближайшей точки к курсору мыши или '???', если такая точка не определена:

```
procedure TForml.DBChartlDblClick(Sender: TObject);
var Tmp : Integer;
begin
  Tmp := DBChartl.Series[0].GetCursorValuelndex;
  IF Tmp >= 0 THEN Label1.Caption :=  IntToStr(Tmp)
  ELSE                              /
  Label1.Caption := "???';
end;
```

procedure GetCursorValues(Var x, y: Double); — возвращает значения по X и Y точки графика (а не только серии), ближе всего к которой расположен курсор мыши. Например, Label2.Caption и Label3.Caption в следующем фрагменте содержат соответственно значения координат X и Y графика, соответствующие точке, на которой находится курсор мыши:

```
var TmpX, TmpY : Double;
……………………………………..
DBChartl.Series[0].GetCursorValues(TmpX, TmpY);
Label2.Caption := Format('%10.2f', [TmpX]);
Label3.Caption := Format('%10.2f', [TmpY]);
```

function GetHorizAxis: TChartAxis; — возвращает указатель на назначенную серии горизонтальную ось. Используя данный указатель, можно вызывать методы оси, обращаться к ее свойствам.

function GetVertAxis:TChartAxis; — возвращает указатель на вертикальную ось.

function MaxXValue: Double; virtual; — возвращает максимальное значение по X.

function MinXValue: Double; virtual; — возвращает минимальное значение по X.

function MaxYValue: Double; virtual; — возвращает максимальное значение по У.

function MinYValue: Double; virtual; — возвращает минимальное значение по Y.

procedure RefreshSeries; — обновляет значения серии из источника данных указанного в свойстве DataSource.

procedure Repaint; — приводит к полной перерисовке всего графика Рекомендуется вызывать этот метод в случае изменения хотя бы одного из основополагающих свойств серии (например, при изменении значения и DataSource и др.).

function ValuesListCount:LongInt; — возвращает число списков значении точки, используемых в серии. Обычно это 2 (XValues и YValues), но некоторые серии используют 3 (BubbleSeries — XValues, YValues, Radius; GanttSeries — Y, Start,End).

function VisibleCount: Longlnt; — возвращает число точек серии, видимых на графике.

События компонента TChartSeries

property OnBeforeAdd: TSeriesOnBeforeAdd;

TSeriesOnBeforeAdd = Function(Sender: TChartSeries): Boolean of object;

Наступает перед добавлением точки в серию. В обработчике данного события может производиться анализ корректности добавляемых в серию точек. Наступает также при соединении серии с источником данных (TTable или TQuery).

property OnAfterAdd: TSeriesOnAfterAdd;

TSeriesOnAfterAdd = procedure(Sender:TChartSeries; ValueIndex:Longint) of object;

Происходит после добавления точки в серию.

property OnClearValues: TSeriesOnClear;

TSeriesOnClear = procedure(Sender: TChartSeries) of object;

Происходит при очистке серии от точек.

property OnClick : TSeriesClick;

TSeriesClick = procedure(Sender:TChartSeries; Valuelndex: Longlnt; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

Происходит при щелчке мышью на серии.

property OnGetMarkText: **TSeriesOnGetMarkText;**

TSeriesOnGetMarkText = procedure (Sender : TChartSeries ; Valuelndex : Longlnt; Var MarkText: String)

Происходит при формировании марки для точки в серии. Обработчик может использоваться для изменения содержимого марки.

Разработка формы с реализацией построения графиков для БД отдела кадров

Откроем новую, пустую форму.

Поместим на неё компоненты TQuery и TDBChart.

Построим графики распределения кадрового состава по подразделениям.

Сформируем набор данных qDiagram: TQuery. В его свойстве SQL запишем:

```
select o.Otdel_name as Подразделение, count(c.id_people) as Kolvo
from contract c, otdel o
where c.id_otdel=o.id_otdel
group by o.Otdel_name
```

Сформируем серии типа Pie и Bar согласно пунктам 1, 1.1, 1.2.

В результате необходимо получить графики как на рис. 30 и 31.

Будем менять вид показываемых графиков по нажатию радио-кнопки в группе радиокнопок. Для этого разместим на форме компонент с палитры Standart RadioGroup1: TRadioGroup.

Вид формы разработанного приложения приведен на рис. 32.

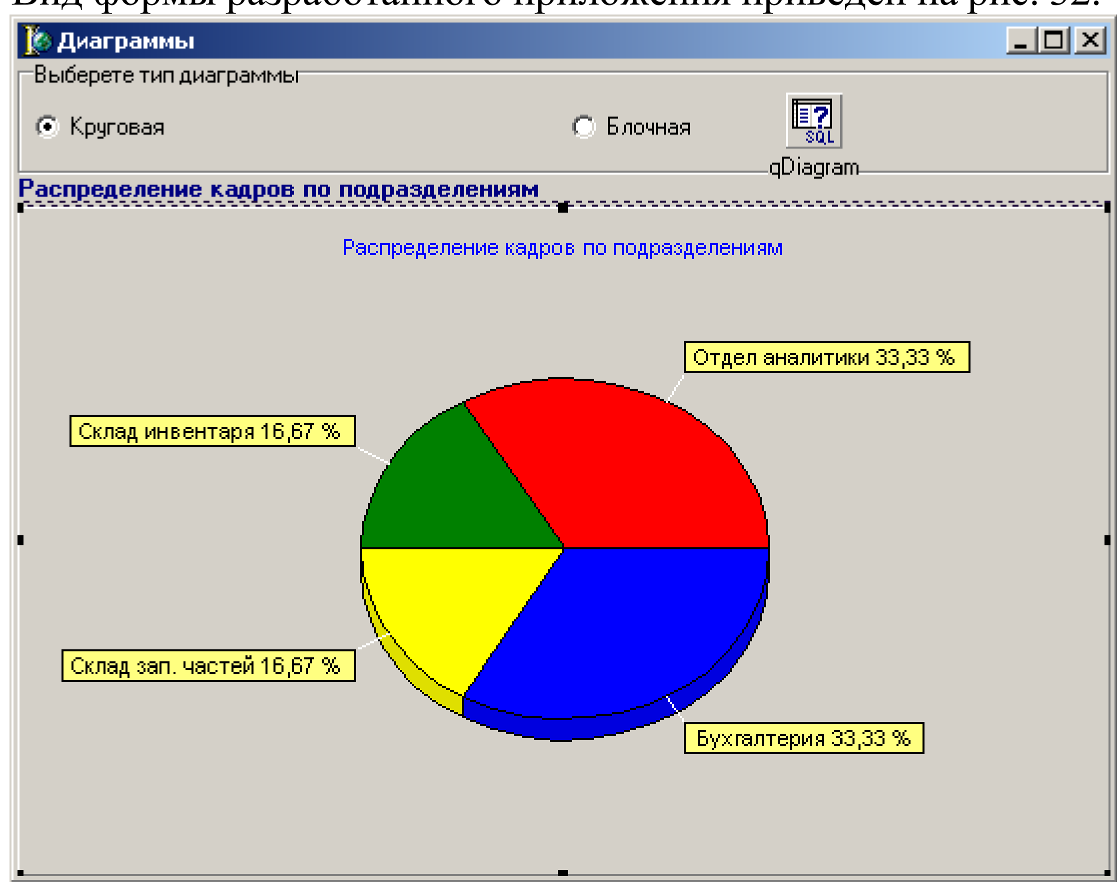


Рис. 32 — Вид формы разрабатываемого приложения

Какая радиокнопка сейчас нажата содержит свойство RadioGroup1.ItemIndex. Его значения и будем отслеживать. Установим по умолчанию RadioGroup1.ItemIndex:=0 — нажата первая кнопка. По нажатию первой кнопки будем показывать круговую диаграмму. Для этого в DBChart1 установим активной сформированную серию типа Pie.

На событие показа формы запишем:

```
procedure TfmDiagram.FormShow(Sender: TObject);
begin
  // Устанавливаем программно путь к БД
  if qDiagram.Active then qDiagram.Close;
```

```
    qDiagram.DatabaseName:=ExtractFilePath(Application.ExeName)+'db';
    qDiagram.Open;
  // Отслеживаем нажатие кнопок для показа графиков
    case RadioGroup1.ItemIndex of
      0:begin
         DBChart1.Series[0].Active:=True;
         DBChart1.Series[1].Active:=False;
        end;
      1:begin
         DBChart1.Series[1].Active:=True;
         DBChart1.Series[0].Active:=False;
        end;
    end;
  end;
```

Запустим приложение.

Можете построить и другие зависимости по информации в БД отдела кадров.

СПИСОК ЛИТЕРАТУРЫ

1. Кагаловский М.Р. Технология баз данных на персональной ЭВМ. — М.: Финансы и статистика, 1992. — 224 с.

2. Ревунков Г.И., Самохвалов Э.М., Чистов В.В. Базы и банки данных и знаний. — М.: Высшая школа, 1992 — 368 с.

3. Дейт К. Введение в системы баз данных. — М.: Наука, 1980. — 463 с.

4. Гэри Хансен, Джеймс Хансен Базы данных: разработка и управление: Пер. с англ. — М.: ЗАО «Издательство БИНОМ», 1999. — 704 с.

5. Боуман Джудит, Эмерсон Сандра, Дарновски Марси Практическое руководство по SQL / 3-е издание.: Пер. с англ. — К.: Диалектика, 1997. — 320с.

6. Саймон А.Р. Стратегические технологии баз данных: менеджмент на 2000 год: Пер. с англ./ Под ред. И с предисл. М.Р. Кагаловского. — М.: Финансы и статистика, 1999.— 479 с.

7. Базы даны: модели, разработка реализация / Т.С. Карпова.— СПб.: Питер, 2001. — 304 с.