

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ  
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ»

## **ИСПОЛЬЗОВАНИЕ СРЕДЫ C++ BUILDER ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ**

Учебно-методическое пособие для вузов

Составитель  
Ю.А. Крыжановская

Издательско-полиграфический центр  
Воронежского государственного университета  
2009

## Содержание

Введение.....	4
Основные компоненты для работы с базами данных .....	5
Модули данных .....	12
Динамическое создание базы данных .....	12
Добавление полей в таблицу .....	15
Создание базы данных из приложения.....	16
Обработка событий базы данных.....	17
Проверка данных .....	20
SQL-ориентированный доступ.....	22
Создание статического запроса.....	22
Генератор запросов .....	23
Динамические запросы .....	25
Хранимые процедуры .....	27
Утилиты.....	27
Задания .....	28
Список литературы .....	29

Компонент **TTable** предоставляет доступ к одной таблице. События этого компонента позволяют строить и контролировать поведение приложений. Для использования компонента TTable следует:

- разместить TTable на форме или в модуле данных (см. ниже);
- задать значение свойств DatabaseName и TableName;
- разместить на форме или в модуле данных компонент DataSource и установить значение свойства DataSet равным имени компонента TTable;
- компоненты со страницы *Data Controls* расположить на форме и связать с компонентом DataSource для отображения данных.

Так как базы данных предназначены не только для хранения, но и для выбора и обработки информации, одним из важнейших аспектов их использования является создание запросов к ним. Для создания запроса используется компонент **TQuery**. Он позволяет использовать операторы SQL для определения или создания наборов данных, которые можно отображать, вставлять, удалять и редактировать. Отметим, что язык запросов SQL (Structured Query Language), обычно применяемый при работе с серверными СУБД, может быть использован и при работе с таблицами формата dBase и Paradox. Пример использования компонента TQuery будет рассмотрен далее.

Объекты класса **TField** являются свойством объекта TDataSet. Свойство Fields объекта типа TDataSet позволяет обращаться к отдельным полям набора данных и является массивом объектов TField, динамически создающимся во время выполнения приложения. Элементы массива соответствуют колонкам таблицы. Объект TField не делает никаких предположений относительно типов данных, с которыми он связан, и имеет несколько свойств, позволяющих установить или вернуть обратно значения поля, например AsString, AsBoolean, AsFloat, AsInteger. Fields Editor позволяет создать статический список полей таблицы, добавляемых к описанию класса формы на этапе проектирования приложения. При внесении колонок с использованием Fields Editor для каждого из полей, добавленных к TDataSet, возникают объекты TField, после чего можно увидеть эти поля в инспекторе объектов и использовать в приложениях их свойства, события и методы, а ссылки на них появятся в h-файле формы. Fields Editor используется следующим образом:

- разместить TTable или TQuery на форме, указать DatabaseName и имя таблицы или свойство SQL для TQuery;
- для компонента TDataSet вызвать контекстное меню, в котором выбрать Fields Editor;
- снова вызвать контекстное меню и выбрать опцию Add Fields (имена всех колонок таблицы или запроса появятся в диалоговой панели Add Fields), выбрать поля для внесения в список объектов, нажать ОК;
- для создания вычисляемого поля на основе имеющихся полей из контекстного меню выбрать New Field для создания нового поля на основе

существующего или для создания вычисляемого поля (в дальнейшем следует создать код обработчика события OnCalcFields компонента TTable, где и производятся необходимые вычисления);

– для удаления статического поля из списка нужно в контекстном меню выбрать Delete.

Если применить операцию drag-and-drop к выделенным в Fields Editor полям, перенеся их на форму, можно получить готовую форму с необходимым набором интерфейсных элементов.

Компоненты **DBLookup** используются при наличии связанных таблиц, когда необходимо вывести на экран описательную информацию вместо поля, содержащего ее код. Таких компонентов четыре. **DBLookupList** и **DBLookupListBox** позволяют вывести на экран набор вариантов, основанных на значении в другой таблице. Эти компоненты отличаются от компонента DBListBox тем, что позволяют согласовать выбранное значение из списка с текущей строкой другой таблицы БД, тогда как для DBListBox список значений для выбора определен заранее и не имеет отношения к таблицам БД. Похожи на них и компоненты **DBLookupCombo** и **DBLookupComboBox**, за исключением возможности пользователя выбирать значение в списке либо вводить новое. Компоненты DBLookupCombo и DBLookupComboBox аналогичны компоненту ComboBox, но позволяют согласовать выбранное значение с текущей строкой другой таблицы БД.

При рассмотрении различных примеров работы с БД будем пользоваться таблицами базы данных BCDEMOS, содержащейся в комплекте поставки CBuilder.

## Лабораторная работа № 1

Рассмотрим простой пример для просмотра полей БД [3]. Создадим форму, показанную на рис. 1, добавив на пустую форму объекты TStringGrid, TMainMenu, TOpenDialog и TTable. В данном случае напрямую устанавливать свойства объектов grid, dialog или table не будем. Объект main menu должен содержать пункт «Файл» с подпунктами «Открыть» и «Выход». Пункт меню «Открыть» будет использоваться для выбора таблицы базы данных для просмотра, а пункт «Выход» — для закрытия приложения.

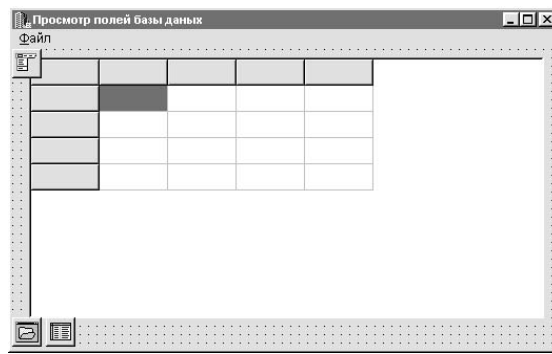


Рис. 1. Форма программы просмотра полей базы данных

Создадим таблицы типов полей. Типы полей хранятся в БД как числовые значения типа **TFldType**. Для отображения этих данных в более удобном виде следует в начало исходного файла `Unit1.h` добавить код:

```
typedef struct
{
    int nCode;
    char *strDesc;
} DbFieldType;
DbFieldType sFieldTypes[] =
{
    {ftUnknown, "Неизвестно или не определено"},
    {ftString, "Символьное или строковое поле"},
    {ftSmallint, "16-битное целое поле"},
    {ftInteger, "32-битное целое поле"},
    {ftWord, "16-битное беззнаковое целое поле"},
    {ftBoolean, "Логическое поле"},
    {ftFloat, "Поле чисел с плавающей точкой"},
    {ftCurrency, "Денежное поле"},
    {ftBCD, "Двоично-кодированное десятичное поле"},
    {ftDate, "Поле даты"},
    {ftTime, "Поле времени"},
    {ftDateTime, "Поле даты и времени"},
    {ftBytes, "Фиксированное количество байт (двоичное представление)"},
    {ftVarBytes, "Переменное количество байт (двоичное представление)"},
    {ftAutoInc, "Автоматически увеличивающееся 32-битное целое поле счетчика"},
    {ftBlob, "Поле Binary Large Object (большой двоичный объект)"},
    {ftMemo, "Поле мемо (строка неограниченной длины)"},
    {ftGraphic, "Поле растрового рисунка"},
    {ftFmtMemo, "Поле форматированного мемо"},
    {ftParadoxOle, "Поле Paradox OLE"},
    {ftDBaseOle, "Поле dBase OLE"},
    {ftTypedBinary, "Типизированное двоичное поле"},
    {-1, ""}
};
```

Эта таблица будет ставить в соответствие типы данных (символы `ftxxx`) и строки, которые характеризуют тип поля.