ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ И КОММУНИКАЦИИ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫЙ ИНЖИНИРИНГ

КУРСОВАЯ РАБОТА

по предмету “Программирование на C++ ”

На тему: «Работа с базами данных в Borland C++ Builder»

Выполнил: Базарбаев Манас
Принял: Кадыров Азизбек

Оглавление

Введение

В данной курсовой работе я хочу создать не большую программу для использования базы данных в новой версии C++ Builder XE8. Я буду создать несложную программу “Расписание группы”, для этого я использую новую компоненты из вкладки FireDAC для работы с базами данных и язык Sqlite. Причина использования языка Sqlite это то что этот язык сейчас очень популярен, потому что оно используется в популярных мобильных системах IOS и Android.

C++ Builder отличная среда для разработки приложении для платформы Windows OS. Но последние версии этого продукта от компании Embarcadero позволяют создать кроссплатформенные приложения для десктопных и мобильных платформ как Linux OS, OS X, Windows OS, Android OS и IOS.

Набор данных в C++ Builder - это объект, состоящий из набора записей, каждая из которых, в свою очередь, состоит из полей, и указателя текущей записи. Набор данных может иметь полное соответствие с реально существующей таблицей или быть результатом запроса, он может быть частью таблицы или объединять между собой несколько таблиц.

На сегодняшний день все носимые устройства и смартфоны работают с помощью базы данных. А также базы данных играет огромную роль в клиент-серверных приложениях. Кстати о клиент серверных приложениях, программа которую я буду создать в данной курсовой работе можно расширить и сделать клиент-серверным приложением чтобы обновить базу данных с воздуха(с интернета). Это будет отличным приложением для студентов, школьников и преподавателей.

C++ Builder обладает широкими возможностями доступа к базам 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. Поэтому сегодня мы рассмотрим, как в C++ Builder решаются проблемы построения запросов.
Запрос в C++ Builder - это объект, представляющий собой набор данных. Обычно для создания запроса используется компонент TQuery - потомок абстрактного класса TDataSet.

I. Теоретическая часть

1.1 Компоненты для работы с базами данных

Компонент TFDConnection

Компонент TFDConnection обеспечивает подключение к различным типам баз данных. Будем указывать экземпляр этого компонента в свойствах Connection остальных компонентов FireDac. К какому именно типу баз данных будет происходить подключение, зависит от значения свойства DriverName. Для доступа к Firebird нам необходимо выставить это свойство в значение FB. Для того чтобы подключение знало, с какой именно библиотекой доступа необходимо работать, разместим в главном дата модуле компонент TFBPhysFBDriverLink. Его свойство VendorLib позволяет указывать путь до клиентской библиотеки. Если оно не указано, то подключение к Firebird будет осуществляться через библиотеки, зарегистрированные в системе, например в system32, что в ряде случаев может быть нежелательно.
Параметры подключения

Компонент TFDConnection параметры подключения к базе данных содержатся в свойстве Params (имя пользователя, пароль, набор символов соединения и др.). Если воспользоваться редактором свойств TFDConnection (двойной клик на компоненте), то упомянутые свойства будут заполнены автоматически. Набор этих свойств зависит от типа базы данных.

Параметр	**Назначение**
Pooled	Используется ли пул соединений.
Database	Путь к базе данных или её псевдоним, определённый в файле конфигурации aliases.conf (или databases.conf) сервера Firebird.
User_Name	Имя пользователя.
Password	Пароль.
OSAuthent	Используется ли аутентификация средствами операционной системы.
Protocol	Протокол соединения. Допускаются следующие значения: • Local – локальный протокол;

	• NetBEUI – именованные каналы; • SPX – не поддерживается в современных версиях; • TCPIP – TCP/IP.
Server	Имя сервера или его IP адрес. Если сервер работает на нестандартном порту, то необходимо также указать порт через слэш, например localhost/3051.
SQLDialect	Диалект. Должен совпадать с диалектом базы данных.
RoleName	Имя роли.
CharacterSet	Имя набора символов соединения.

Дополнительные свойства:

Connected – управление подсоединением к БД, или проверка состояния соединения. Это свойство должно быть выставлено в True для работы мастеров других компонентов FireDac. Если ваше приложение должно запрашивать данные для авторизации, то важно не забыть сбросить это свойство в False перед компиляцией вашего приложения.

LoginPrompt – запрашивать ли имя пользователя и пароль при попытке соединения.

Transaction – компонент TFDTransaction, который будет использоваться в качестве умолчательного для выполнения различных операций TFDConnection. Если это свойство не назначено явно, TFDConnection создаст себе экземпляр TFDTransaction самостоятельно, его параметры можно указать в свойстве TxOptions.

UpdateTransaction – компонент TFDTransaction, который будет использоваться в качестве умолчательного для одноимённых свойств компонентов TFDQuery. Если это свойство не назначено явно, будет использовано значение из свойства Transaction.

Компонент TDataSource

Компонент DataSource действует как посредник между компонентами TDataSet (TTable, TQuery, TStoredProc) и компонентами Data Controls -

элементами управления, обеспечивающими представление данных на форме. Компоненты TDataSet управляют связями с библиотекой Borland Database Engine (BDE), а компонент DataSource управляет связями с данными в компонентах Data Controls.

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

- Свойство DataSet компонента DataSource идентифицирует имя компонента TDataSet. Можно присвоить значение свойству DataSet на этапе выполнения или с помощью инспектора объектов на этапе проектирования.
- Свойство Enabled компонента DataSource активизирует или останавливает взаимосвязь между компонентами TDataSource и Data Controls. Если значение свойства Enabled равно true, то компоненты Data Controls, связанные с TDataSource, воспринимают изменения набора данных. Использование свойства Enabled позволяет временно разъединять визуальные компоненты Data Controls и TDataSource, например, для того, чтобы в случае поиска в таблице с большим количеством записей не отображать на экране пролистывание всей таблицы.
- Свойство AutoEdit компонента DataSource контролирует, как инициируется редактирование в компонентах Data Controls. Если значение свойства AutoEdit равно true, то режим редактирования начинается непосредственно при получении фокуса компонентом Data Controls, связанным с данным компонентом TDataSet. В противном случае режим редактирования начинается, когда вызывается метод Edit компонента TDataSet, например, после нажатия пользователем кнопки Edit на компоненте DBNavigator. · Событие OnDataChange компонента DataSource наступает, когда происходит изменение значения поля, записи, таблицы, запроса.
- Событие OnUpdateData компонента DataSource наступает, когда пользователь пытается изменить текущую запись в TDataSet. Обработчик этого события следует создавать, когда требуется соблюсти условия ссылочной целостности или ограничения, накладываемые на значения полей изменяемой базы данных.

Компонент TDBGrid

Компонент TDBGrid обеспечивает табличный способ отображения на экране строк данных из компонентов TTable или TQuery. Приложение может использовать TDBGrid для отображения, вставки, уничтожения, редактирования данных БД. Обычно DBGrid используется в сочетании с DBNavigator, хотя можно использовать и другие интерфейсные элементы, включив в их обработчики событий методы First, Last, Next, Ptior, Insert, Delete, Edit, Append, Post, Cancel компонента TTable.

Внешний вид таблицы (например, надписи в заголовках столбцов) может быть изменен с помощью редактора свойств Columns Editor. Для вызова Columns Editor нужно либо выбрать соответствующую опцию в контекстном меню компонента DBGrid или щелкнуть мышью в колонке значений напротив свойства Columns в инспекторе объектов.

Вторым способом получения контроля над характеристиками DBGrid или другими компонентами является создание описанным выше способом статического набора компонентов TField. Имея компонент типа TField, созданный для каждого из полей в наборе данных, можно установить ширину, формат, маску, расположение, метку для отображения в DBGrid и другие характеристики.

Поля Float, Integer и Date обладают свойством DisplayMask. Это свойство можно использовать, чтобы форматировать данные в компоненте DBGrid или другом компоненте Data Controls. Например, экранный формат mm-dd-yy может использоваться для размещения полей типа дата.

Некоторые компоненты TField (например, TStringField) обладают свойством EditMask, которое можно установить, вводя данные в DBGrid и другие компоненты Data Controls. Для установки свойства EditMask нужно установить компонент Field в Object Inspector и выбрать свойство EditMask, после чего появится диалоговая панель Input Mask Editor.ьЧтобы проверить маску редактирования, нужно ввести значение в поле Test Input.

Компонент TQuery

Компонент TQuery, как и компонент TTable, обладает всеми свойствами компонента TDataSet.

Как и в случае с компонентом TTable, компонент TDataSource управляет взаимодействием между компонентами Data Controls и компонентом TQuery. Обычно приложение имеет один компонент DataSource для каждого компонента TQuery.

Наиболее часто используются следующие свойства компонента TQuery:

- Active - указывает, открыт (true) или закрыт (false) данный запрос

- Eof, Bof - эти свойства принимают значение true, когда указатель текущей записи расположен на последней или соответственно первой строке набора данных, являющегося результатом выполнения запроса.
- DatabaseName - имя каталога либо псевдоним (alias) удаленной БД, к которой осуществляется запрос.
- DataSource - указывает источник данных для параметризованных запросов (т.е. запросов с параметрами, значение которых заранее неизвестно).
- Fields - это свойство доступно только во время выполнения (run-time only) и используется для чтения или модификации поля, определяемого по порядковому номеру.
- Params - содержит параметры для параметризованного запроса, как SomeNo в следующем примере:
  Select * from Orders where CustNo=:SomeNo
- SQL - строковый массив, содержащий текст оператора запроса SQL.

Отметим, что язык запросов SQL (Structured Query Language), традиционно применяемый при работе с серверными СУБД, может быть использован и при работе с таблицами формата dBase и Paradox. Не вдаваясь в подробное описание синтаксиса этого языка (с ним можно познакомиться в других источниках, например, в книге М.Грабера "Введение в SQL"), отметим одну его особенность. SQL – язык непроцедурный. На нем можно написать, что нужно получить в результате запроса, но нельзя написать, как это сделать, то есть нельзя описать саму процедуру выполнения запроса. Дело в том, что реализация выполнения тех или иных операторов SQL серверами баз данных может быть различна, и в большинстве случаев неинтересна клиентскому приложению, создаваемому с помощью C++ Builder. В случае таблиц dBase или Paradox реализацию SQL берет на себя библиотека Borland Database Engine.

Компонент TQuery позволяет использовать операторы SQL для того, чтобы определять или создавать наборы данных, которые можно отобразить на экране, вставлять, удалять и редактировать строки.

- RequestLive - если это свойство имеет значение true и синтаксис запроса таков, что его результат может быть модифицируем, пользователь может редактировать данные с сохранением их в базе данных. Если RequestLive имеет значение false, результат запроса возвращается в состоянии read-only.

  Наиболее часто используются следующие методы компонента TQuery:
- ExecSQL - выполняет SQL-запрос, содержащийся в свойстве SQL, если запрос не возвращает данные. Следует употреблять этот метод при

вставке, редактировании или удалении данных. При выполнении же оператора SELECT (выбор данных) следует использовать метод Open. Следующий пример показывает применение метода ExecSQL:

```
Query1->Close();
Query1->SQL->Clear();
Query1->SQL->Add("Delete emp where empno=1010");
Query1->ExecSQL();
```

- Open - открывает компонент TQuery. Он эквивалентен присвоению свойству Active значения true. Используется, если результатом запроса является набор данных (такие запросы обычно начинаются с оператора SELECT). Пример использования метода Open:

```
Query1->Open();
```

- Close - закрывает компонент TQuery. Вызов Close эквивалентен присвоению свойству Active значения false. Пример использования метода Close:

```
Query1->Close();
```

- Prepare - обеспечивает передачу серверу баз данных запроса, содержащегося в свойстве SQL, для оптимизации и компиляции. Полный запрос с параметрами не передается, пока не вызваны методы Open или ExecSQL. Даже если метод Prepare не вызывается явно, он будет вызван неявно, если используются методы Open или ExecSQL (в этом можно убедиться, запустив утилиту SQL Monitor). Пример использования метода Prepare:

```
Query1->Close();
Query1->SQL->Add("Delete emp where empno=:empno");
Query1->Prepare();
```

Компоненты TQuery обладают большим разнообразием методов, унаследованных от TDataSet. Наиболее часто используются следующие методы:

- First, Last, Next, Prior перемещают указатель текущей записи на первую, последнюю, следующую и предыдущую записи соответственно, например:
- MoveBy перемещает указатель на определенное количество строк.
- Insert, Edit, Delete, Append, Post, Cancel - позволяют модифицировать результат запроса. Метод Insert позволяет вносить в результат запроса строки, как в следующем примере:

```
Query2->Insert();
Query2->Fields[0]->AsInteger = 100;
```

```
Query2->Fields[1]->AsString =Edit1->Text;
Query2->Post();
```

Метод Post подтвержает операции Insert, Update или Delete, совершая реальное физическое изменение в базе данных. Метод Cancel отменяет незавершенные операции Insert, Delete, Edit или Append.

- FreeBookmark, GetBookmark, GotoBookmark- - позволяют создавать закладки (маркированные строки) в запросе и затем вернуться к такой строке позже.

Прежде чем перейти к непосредственному использованию запросов, следует остановиться на весьма полезном инструменте - генераторе запросов Visual Query Builder, с помощью которого можно определить свойство SQL компонента TQuery, если по каким-либо причинам это неудобно делать непосредственно в редакторе свойств.

1.2 База данных SQLite

Основные команды Sqlite

CREATE TABLE

Новая таблица создаётся с помощью команды **CREATE TABLE** после которой указывается имя таблицы, а затем в круглых скобках указываются имена столбцов с параметрами.

Создать новую таблицу **tbl_info** с четырьмя столбцами:

```
CREATE TABLE tbl_info (
   _id integer PRIMARY KEY AUTOINCREMENT,
   name text NOT NULL,
   age integer NOT NULL,
   city text NOT NULL
);
```

Ключевое слово **NOT NULL** требует обязательно присваивать значение столбцу при команде **INSERT INTO**.

Существует также ключевое слово **DEFAULT**, позволяющее вставить значение по умолчанию, например, **city text NOT NULL DEFAULT "Москва"**.

DROP TABLE

Удаление таблицы происходит с помощью команды **DROP TABLE**, затем указывается имя таблицы. При желании можно поставить проверочное условие **IF EXISTS**.

Удалить таблицу **tbl_info**:

```
DROP TABLE IF EXISTS tbl_info;
```

INSERT INTO VALUES

Для вставки новой записи в таблицу используется команда **INSERT INTO**, затем указывается имя таблицы, а в скобках имена столбцов. После них идёт ключевое слово **VALUES**, после которого в скобках идут вставляемые значения. Важно соблюдать количество столбцов с вставляемыми данными и их очерёдность при перечислении.
Вставить запись в таблицу **tbl_info**:

```
INSERT INTO tbl_info(name, age, city) VALUES ("Кот Васька", 29, "Москва, Россия");
```

Существует укороченная запись без перечисления столбцов. В этом случае нужно указывать все значения и в том порядке, в котором создавались соответствующие столбцы.

```
INSERT INTO tbl_info VALUES (3, "Кот Мурзик", 10, "Рязань, Россия");
```

Также можно указывать только нужные столбцы для вставки данных. В первом примере мы использовали данный способ, указав только три столбца, хотя таблица содержит четыре столбца.

SELECT * FROM

Для чтения данных используется команда **SELECT** с условием, затем ключевое слово **FROM** с указанием имени таблицы. Чтобы просмотреть все записи, для условия **SELECT** используется звёздочка (*).

```
SELECT * FROM tbl_info;
```

UPDATE

Обновить запись в таблице **tbl_info**:

```
UPDATE tbl_info SET age=12 WHERE _id=1;
```

После **UPDATE** указываете таблицу, после **SET** - в каком столбце нужно внести изменения и указывается новое значение, а затем указывается условие.
Можно обновлять группу столбцов, указывая их через запятую.

```
UPDATE ваша_таблица SET первый_столбец="новое значение", второй_столбец="новое значение";
```

Команда **UPDATE** заменяет собой пару команд **INSERT/DELETE**.
Обновить данные в нужном столбце:

```
UPDATE table_name SET имя_столбца = новое_значение WHERE имя_столбца = старое_значение
```

Также можно производить математические действия: прибавлять, отнимать, умножать, делить. Увеличим возраст кота на день рождения.

```
UPDATE tbl_info SET age=age+1 WHERE name="Мурзик";
```

DELETE

Удалить запись из таблицы **tbl_info**:

```
DELETE FROM tbl_info WHERE _id=1;
```

Вам не надо перечислять все столбцы, достаточно указать в условии нужный столбец. Условие **WHERE** работает аналогично как в команде **SELECT** и позволяет использовать ключевые слова **LIKE**, **BETWEEN** и т.д.

Удалить все записи:

```
DELETE * FROM tbl_info;
```

SELECT

Рассмотрим подробнее команду **SELECT**.

Чтобы не искать все записи, можно ограничить поиск условием **WHERE**, после которого идёт имя столбца и условие равенства. Показать всех котов, чей возраст меньше 15.

```
SELECT * FROM tbl_info WHERE age < 15;
```

Вместо звёздочки можно указать столбцы, которые вам нужны. Например, нам нужны только имена котов с этим же условием.

```
SELECT name FROM tbl_info WHERE age < 15;
```

Столбцы указываются через запятую. Нам нужны имена и адреса котов с этим же условием.

```
SELECT name, city FROM tbl_info WHERE age < 15;
```

Условие **WHERE** можно объединять с помощью ключевого слова **AND**. Список котов младше 15 лет и проживающих в Москве.

```
SELECT name, city FROM tbl_info WHERE age < 15 AND city="Москва";
```

Также доступны слова **OR** (ИЛИ):

```
SELECT name, city FROM tbl_info WHERE age < 15 OR city="Москва";
```

Слов **AND** и **OR** может быть несколько в одном запросе.

Проверку на NULL можно сделать с помощью ключевого слова **IS NULL** (если столбец таблицы создавался без **NOT NULL**)

```
SELECT * FROM tbl_info WHERE age IS NULL;
```

Ключевое слово **LIKE** позволяет сократить множество операторов **OR**. Например, мы хотим узнать имена котов, которые заканчиваются на "ик":

```
SELECT name FROM tbl_info WHERE name LIKE '%ик'
```
Символ **%** в строке указывает на любое слово с нужным окончанием (представляет любое количество неизвестных символов).
Также можно использовать спецсимвол **_** для одного символа. Ищем Ваську.

```
select name from tbl_info where name like '_аська'
```
С помощью ключевого слова **BETWEEN** можно быстро и удобно задать диапазон.

```
SELECT name FROM tbl_info WHERE age BETWEEN 10 and 20;
```
Сравните с более длинной записью

```
SELECT name FROM tbl_info WHERE age >= 10 and age <=20;
```
С помощью условия **IN** за которыми в скобках идут нужные значения, можно задать нужные параметры.

```
SELECT name FROM tbl_info WHERE age IN (10, 29);
```
Это короче, чем

```
SELECT name FROM tbl_info WHERE age = 10 OR age = 29;
```
Ключевое слово **NOT IN** выполняет обратную задачу и позволяет получить записи, которые не входят в данное условие.

```
SELECT name FROM tbl_info WHERE age NOT IN (10);
```
Ключевое слово **NOT** можно использовать не только с **IN**, но и с **BETWEEN**, **LIKE**.
Узнать число записей можно через функцию **COUNT**. Если запись содержит **NULL**, то она не учитывается.

```
SELECT COUNT(name) FROM table_info;
```
Для показа минимального или максимального значения используются функции **MIN** или **MAX**:

```
SELECT name, MAX(age) FROM table_info;
```
Если нам нужно вывести только определённое количество записей, то используйте ключевое слово **LIMIT** с указанием значения.

```
SELECT * FROM table_info LIMIT 3;
```

Существует расширенная версия, когда можно указать два значения через запятую. В первой указывается номер записи (отсчёт от 0), а вторая - число записей. Например, показать вторую запись из таблицы.

```
SELECT * FROM table_info LIMIT 1,1;
```

Первичный ключ

Первичный ключ - столбец таблицы, имеющий уникальное значение для каждой записи. Назначается при создании таблицы. Ключ не может содержать NULL, потому что теряется уникальность, ведь в других записях тоже может оказаться NULL. Значения первичного ключа должны оставаться неизменными.
Во многих случаях для этой цели создают новый столбец, который будет содержать уникальный номер. В Android столбец называют **_id**.
Для указания первичного ключа используется ключевое слово **PRIMARY KEY**.
Чтобы база данных сама заботилась об уникальности первичного ключа, можно добавить к нему ключевое слово**AUTOINCREMENT**, которое будет автоматически увеличивать значение на единицу при вставке новой записи.

ALTER

Добавить новый столбец в таблицу можно с помощью необязательного ключевого слова **ALTER**, за которым идёт название столбца в таблице.

```
ALTER TABLE tbl_info ADD COLUMN weight INTEGER;
```

Чтобы указать, после какого столбца нужно добавить новый столбец, используйте ключевое слово **AFTER**.
Другие ключевые слова: **FIRST**, **BEFORE**, **LAST**, **SECOND**, **THIRD**.
Не все эти команды поддерживаются в SQLite, хотя часто используются Кроме **ADD**, также можно изменить имя и тип данных столбцов (**CHANGE**), изменить тип данных или позиции столбцов (**MODIFY**), удалить столбец из таблицы (**DROP**). в обычных SQL.
Переименовать саму таблицу (**RENAME TO**).

```
ALTER TABLE tbl_info RENAME TO table_info;
```

UPPER

Преобразовать текст из указанного столбца в верхний регистр.

```
UPDATE table_info SET name = UPPER(name);
```

На следующих примерах команд SQL вы быстро научитесь ориентироваться в базе данных SQLite:

```
// установить заголовки столбцов, которые будут отображаться в программе
sqlite> .headers on

// выделить все строки таблицы
select * from table1;

// вычислить количество строк в таблице
select  count(*)  from  table1;

// выбрать определенный набор столбцов
select  coll,  col2  from  table1;

// выделить  различные значения в столбце
select  distinct col1 from table1;

// вычислить различные  значения
select count(col1) from (select distinct col1 from table1);

// сгруппировать  по
select count(*), col1 from table1 group by col1;

// регулярное внутреннее объединение
select * from table1 tl, table2 t2
where  t1.col1 = t2.col1;

// оставшееся  внешнее соединение
// дать всю  информацию  в t1,  даже  если  в t2 нет строк
select * from  table t1 left outer join table t2
on  t1.col1 = t2.col1
where  ....
```

II. Основная часть

2.1 Создание базы данных на SQLite

Для создание базы данных существуют много сред разработки. Я выбрал для создания базы данных среду разработки SQLite Manager. Она лёгкая и понятная. Создаём базу данных ***days.sqlite***, в нём создаём несколько таблиц. Так как база данных который я создаю связан с расписанием я создаю таблицу с псевдонимом Days. Таблица состоит из шести полей как: DayNum с типом,

Serial с типом INT, Subject с типом TEXT, Teacher с типом TEXT, Time с типом TIME и Room с типом INT.

DayNum – для хранения номера дня;

Serial – для очерёдности уроков;

Subject – для хранение имени предмета;

Teacher – для хранения имени преподавателя;

Time – для хранение времени начало предмета;

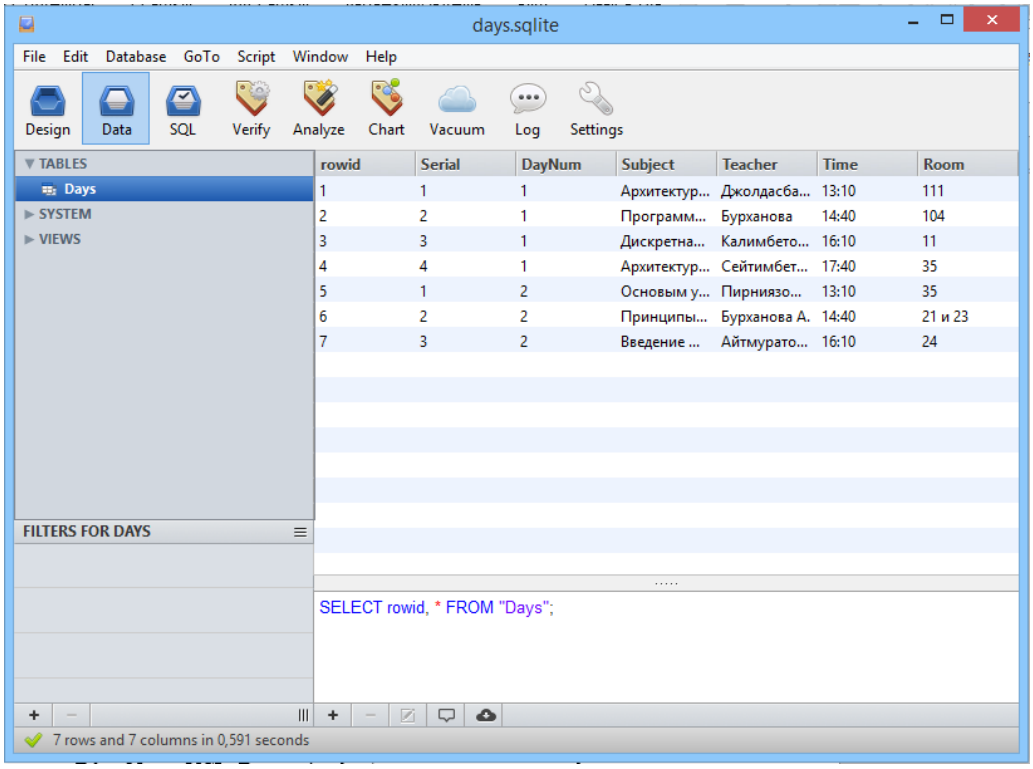
Room – для хранения номера аудитории.

Код базы данных выглядит таким образом:

```
CREATE TABLE "Days" (
"Serial" INTEGER, "DayNum" INTEGER, "Subject" VARCHAR(30),
"Teacher" VARCHAR(30), "Time" TIME, "Room" VARCHAR(9))
```

Эти таблицы можно заполнит непосредственно в среде разработки SqliteManager или программном коде. Я заполнил таблицы в среде разработки.

Пример заполнение таблицы:



2.2 Создание основной формы программы

Для создание программы открываем C++ Builder и создаём новый проект File->New->VCL Forms Application создаётся новая форма под названием Form1.

Теперь настраиваю форму с помощью свойств в Object inspector. В свойстве Caption напишем текст заголовка программы “**Расписание занятии групп**”, в свойствах ClienHeight и ClientWidth пишем 322 и 790. Свойству

BorderStyle заменяем на **bsSingle** чтобы невозможно было менять высоту и ширину формы.

После того как я настроил форму, я поместил нужные компоненты в форму. Сначала я поместил в форму видимые компоненты как Panel, DBGrid, 2 ComboBox, 2 Label и одну Button;

Panel – это компонента для упорядочивания элементов как ComboBox, Label и Button. В свойствах меняем только размеры под свои настройки, а свойство Align меняем в **alRight** для выравнивание в правый край формы.В Panel помещаем наши ComboBoxы, Buttonы и Label;

В свойствах ComboBox1 меняем свойство Style в **csDropDown**. В свойстве Items напишем "**Понедельник, Вторник, Среда, Четверг, Пятница, Суббота**" пишем их в каждую строку.

В свойствах ComboBox2 меняем свойство Style в **csDropDown**. В свойстве Items напишем "**2-в КИ, 2-в ПИ, 2-в ТТ, 2-в ПО**" пишем их в каждую строку.

В Label1 меняем свойство Caption меняем на "**Выберите день**"

В Label2 меняем свойство Caption меняем на "**Выберите группу**"

В Button1 меняем свойство Caption на "**Выход**"

Потом добавляем в форму компоненту DBGrid для отображения полей базы данных. Далее меняем свойства:

Align меняем на **alClient**;

Color меняем на **clInactiveCaption**;

DrawSingle меняем на **gdsGradient**;

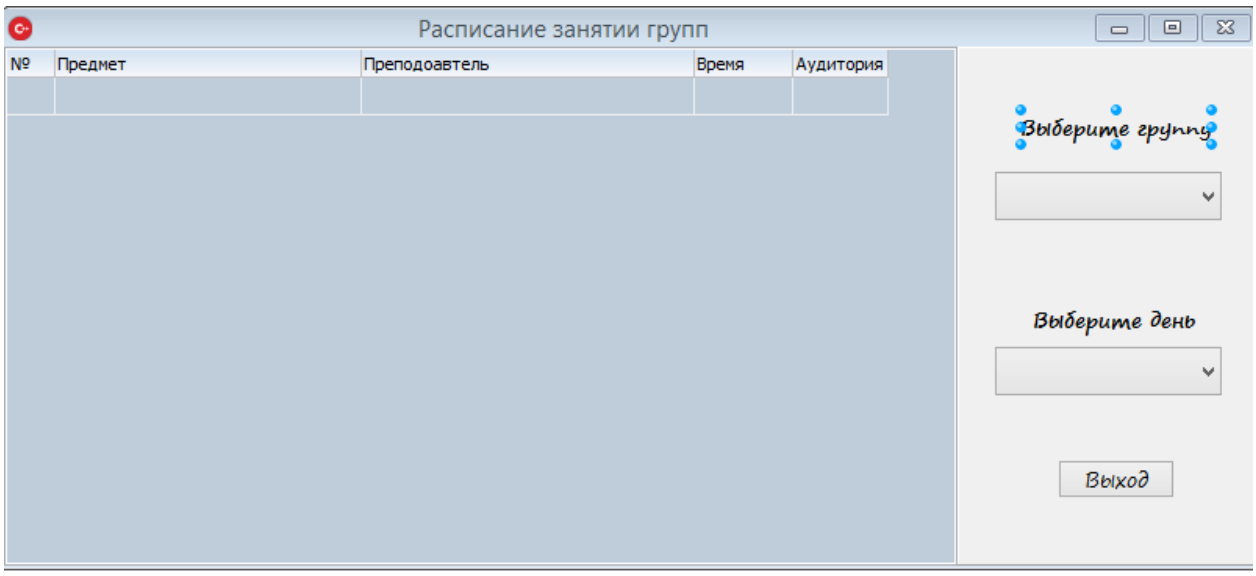
GradientEndColor меняем на **clInactiveBorder**;

GradientStartColor меняес на **clGradientInactiveCaption**;

На Options в goEditing и goIndocator меняем в **false**, а в остальное не трогаем.

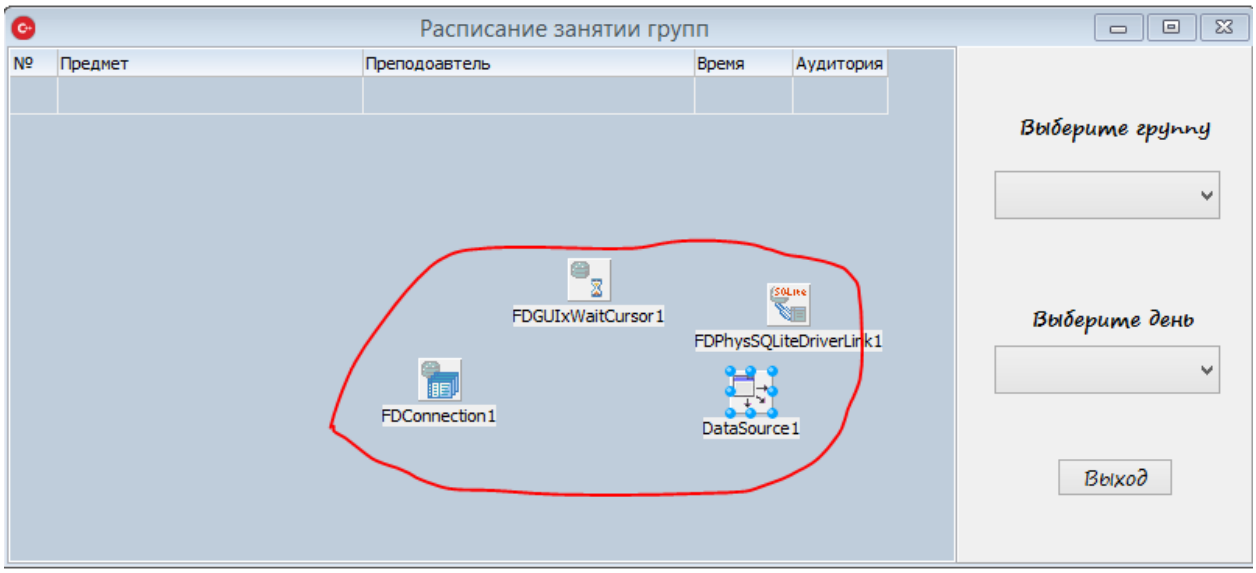
В DBGrid добавляем 5 столбцов для отображение нужных полей нашей базы данных. В свойствах FieldName укажем имена полей.

В итоге главная форма будет выглядит следующим образом.

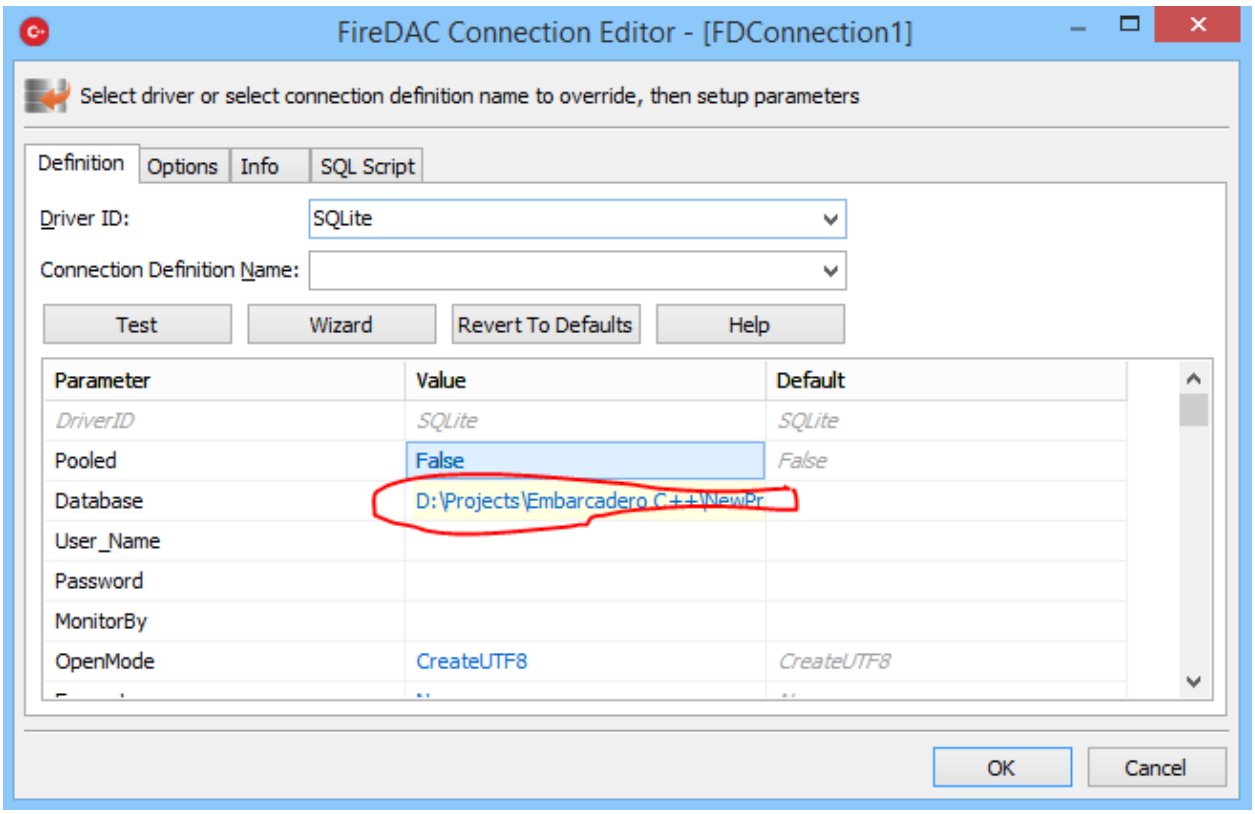


2.3 Использование FireDAC для соединение базы данных

Теперь, когда мы закончили с дизайном формы, соединяем базу данных с программой. Для этого в форму вставляем компоненты как FDConnection, FDGUIxWaitCursor, FDPhysSQLiteDriverLink и DataSource.



В компоненте FDConnection щёлкаем два раза и в открывшимся окне указываем тип базы данных и укажем путь к этой базе данных.



В инспекторе объектов меняем свойство LoginPrompt на **false**, свойство Connection на **true**. В коде программы добавляем библиотеку <FireDAC.DApt.hpp>

В методе FormCreate напишем следующий код:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
	FDConnection1->DriverName = "SQLite";
	FDConnection1->Params->Values["Datbase"] =
ExtractFilePath(ParamStr(0)) + "Days.sqlite";
	try{
		FDConnection1->Open();
	}
	catch(Exception &e){
		ShowMessage("Ошыбка базы данных");
	}
}
```

В ComboBox1 в инспекторе объектов выбираем событие onClick и напишем в нём следующий код.

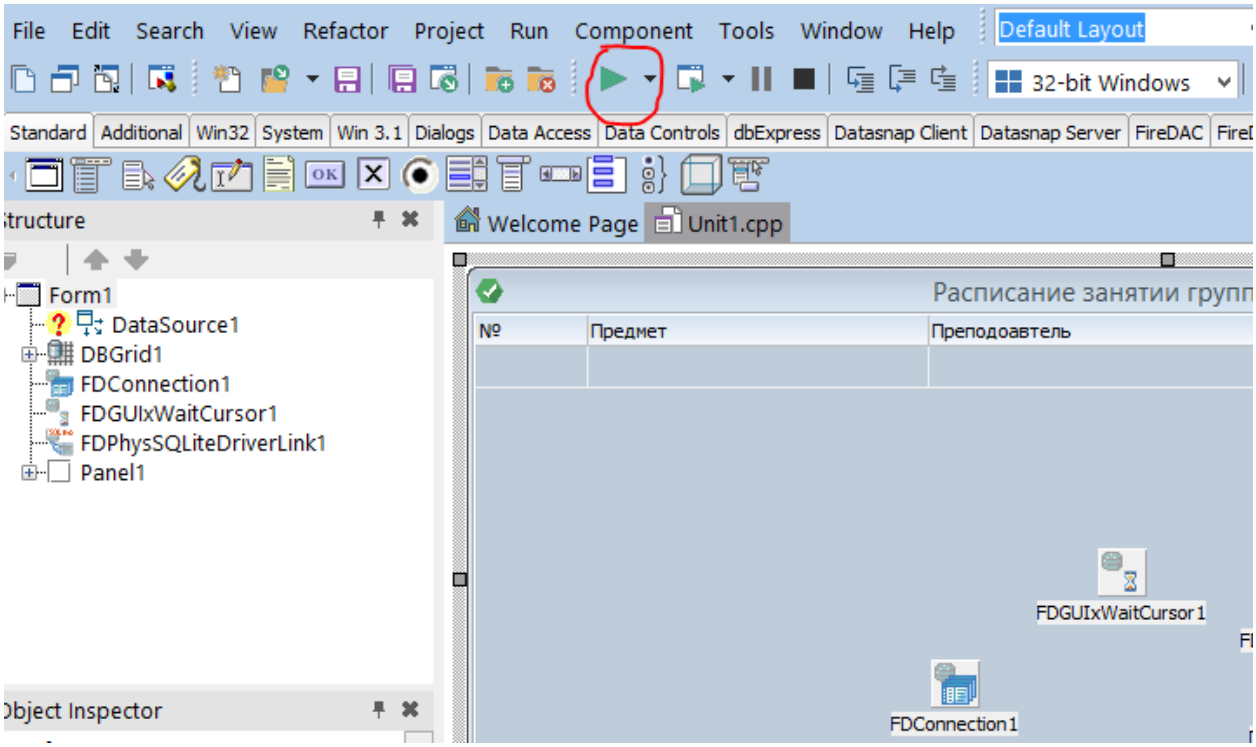
```
void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
	int  nst;
	nst = ComboBox1->ItemIndex;
```

```
        TFDQuery *query;
        query = new TFDQuery(NULL);

        query->Connection = FDConnection1;
        DataSource1->DataSet = query;
        switch (nst) {
                case 0:
                query->SQL->Text = "SELECT * FROM Days WHERE
DayNum=1";
                break;
                case 1:
                query->SQL->Text = "SELECT * FROM Days WHERE
DayNum=2";
                break;
                case 2:
                query->SQL->Text = "SELECT * FROM Days WHERE
DayNum=3";
                break;
                case 3:
                query->SQL->Text = "SELECT * FROM Days WHERE
DayNum=4";
                break;
                case 4:
                query->SQL->Text = "SELECT * FROM Days WHERE
DayNum=5";
                break;
                case 5:
                query->SQL->Text = "SELECT * FROM Days WHERE
DayNum=6";
                break;
   default:
                break;
        }
        query->Open();

}
```

И этим все подключения к базу данных выполнена. Теперь только осталось отладит программу.

2.4 Запуск и отладка приложения.

Запускаем наш проект через команду Run как показано на рисунке ниже:

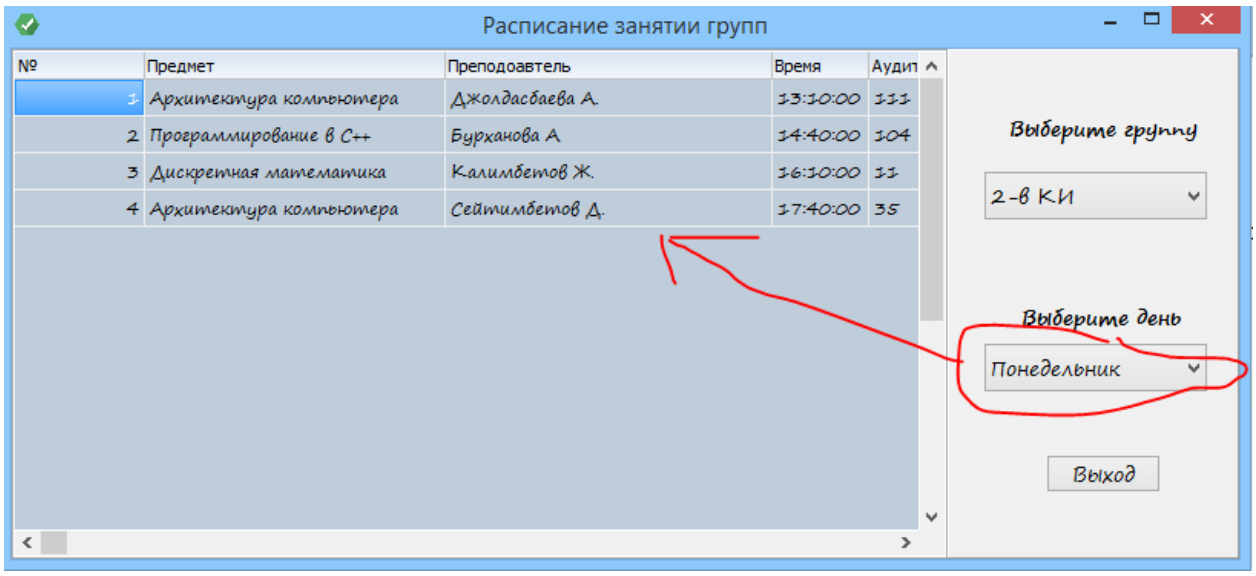


После того как запустилась наша программа тестируем его.

Нажимаем на выпадающие списки.

В первом выпадает список групп, но, когда кликаем по элементам ничего не происходит, потому что для него не написано обработчик событии. Я сделаю это в будущем.

Во втором выпадает список дней недели, когда кликаем по элементам этого списка в DBGrid выводится наша база данных, как показано на рисунке ниже.



Заключение

В заключении можно сказать что с использованием базы данных разработка программы становится намного легче, размер программы уменьшается. Кроме того, чтобы внести изменения в программу не нужно будет изменить исходный код, достаточно изменить базу данных.

На сегодняшний день баз данных есть почти у каждого везде. В компьютерах, в мобильных устройствах, в сайтах, которые мы посещаем и т.д. Поэтому чтобы создать что-то новое требуется знание базы данных. Потому что все популярные приложения работают с базами данных.

Работа с базами данных в последней версии C++ Builder упрощена. А технология FireDAC имеет возможность создать одну базу данных для несколько платформ, и ещё в новом C++ Builder есть возможность создание программ для разных ОС как и мобильных и для настольных платформ. Оно работает почти всеми языками Баз Данных.

В следующим я хочу сделать программу клиент-серверным. Чтобы можно было обновлять базу данных программы с воздуха (с интернета). В последнее время клиент серверные приложения бурно развиваются и это становится некой нормой этого поколения программ. Я хочу сделать эту программу совместимым с популярными мобильными платформами как Android, IOS, WindowsPhone т.к. сейчас мобильные приложения очень популярны и почти у каждого есть смартфон. С этой программой студенты могут узнать о изменения расписании через смартфон. Кроме студентов это расписание могут использовать и преподаватели. Для них будет отдельная программа использующий тот же сервер для обновления.

Эта программа будет отличным дополнение к электронному деканату.

Использованная литература

1. Культин Н. Б. “C++Builder в задачах и примерах” СП6: БХВ-Петерберг 2005-г. 336 с.

2. Архангельский А. Я “Программирование в C++ Builder” «Бином-Пресс», 2010-г. 896с.

3. Habrahabr.ru

4. ru.wikipedia.org

5. Культин Н. Б. “C++Builder ” 2-е изд. перер. И доп. СП6: БХВ-Петерберг 2008-г. 464 с.

6. Ржеуцкая С. Ю “Язык SQL” учебное пособие/С.Ю. Ржеуцкая – Вологда: ВоГТУ, 2010-г. 159 стр.

7. Форта, Бен. “SQL за 10 минут” 4-е издание 2014-г. 288 стр