

Создание приложения управления базой данных *Sqlite* в *Lazarus*

Для управления базами данных подготовлено множество систем управления базами данных СУБД. Например, одной из наиболее распространенных СУБД является *MicroSoft Office Access*. Этот продукт, как и многие другие, является универсальным средством работы с базами данных, требующим значительных усилий для адаптации его под выполнение индивидуальных задач.

Поэтому одним из решений в этом случае является разработка специализированных приложений, предназначенных для создания, редактирования и использования определенной базы данных. Интегрированная среда Lazarus IDE может помочь в проектировании нужного приложения.

3.3.1 Подготовка проекта приложения

В начале разработки приложения нужно создать отдельную папку для проекта с помощью проводника Windows. Имя папки и имя проекта следует согласовать с преподавателем, оно должно содержать только латинские буквы, например *ABC_Lazarus*.

Запустить интегрированную среду разработки приложений Lazarus и создать новый проект как приложение, сохранить проект в созданной специально для него папке.

Проверить успешность компиляции и запуска нового пустого проекта. Закрывать запущенный пустой проект.

Продолжить сборку проекта, т.е. продолжить добавлять визуальные и не визуальные компоненты, обеспечивая их функциональность необходимыми методами – процедурами.

Проверять работу добавленных элементов и периодически сохранять проект, используя пункты меню системы *Lazarus*.

При сохранении проекта создаются два файла **.pas* и **.lpr* (рисунок 3.1). Файл проекта (*lpr*) и файл модуля (*pas*) должны иметь разные имена.

..	
LAZA_OBORUD	exe
LAZA_OBORUD	lpi
LAZA_OBORUD	lps
LAZA_OBORUD	res
unit1	lfm
unit1	pas
LAZA_OBORUD	lpr
LAZA_OBORUD	ico
sqlite3	dll

Рисунок 3.1 – Файлы проекта Lazarus

Проект также включает:

файл *.exe – основной исполняемый файл программы;

файл *.lpi – основной файл проекта *Lazarus*;

файл *.lpr – исходный код основной программы;

файлы *.pas – модули, содержащие коды форм;

файлы *.lfm – файлы, в которых хранятся описания форм модулей;

файл *.lrs – автоматически генерируемый файл ресурсов.

Далее следует подготовить место для размещения компонентов, обеспечивающих выполнение функций, планируемых разработчиком приложения.

Для простого приложения в проекте достаточно иметь один модуль с формой (*Form1*). На этой форме будут располагаться визуальные компоненты, которыми можно быстро заполнить площадь формы.

Поэтому целесообразно использовать компонент *PageControl*, позволяющий в ходе проектирования объединять на одной форме несколько вкладок, содержащих разные элементы управления. Доступ к каждой из вкладок осуществляется при помощи корешков с названиями. В момент установки компонент *PageControl* не содержит в себе ни одной страницы. При помощи команды *New Page* из контекстного меню создается новая страница, при этом в Инспекторе объектов создается новый объект *TabSheet1*. Изменить его название (название вкладки) можно в свойстве *Caption* в Инспекторе объектов.

В предлагаемом примере размещения компонентов на форме (рисунок 3.3) сверху размещается строка с пунктами главного

меню. Ниже следует расположить компонент *Panel* со свойством *Align=alTop*, на котором будут сконцентрированы кнопки управления и навигаторы. Под панелью *Panel* вставляют компонент *PageControll* со свойством *Align=alTop*. И внизу формы размещают компонент *DBGrid1* со свойством *Align=alClient*. Свойство компонента *Align* упрощает процесс его точного размещения.

3.3.2 Доступ к базе данных

Одной из самых простых и достаточно мощных баз данных является *SQLite* ([Приложение 2](#)). Что бы приложение управления базой данных могло функционировать необходимо в папку с проектом поместить динамическую библиотеку *sqlite3.dll* (рисунок 3.1).

Для использования этой библиотеки в *LAZARUS* имеется стандартный набор компонентов на вкладке *SQLdb*. В состав этой вкладки входят компоненты приведенные на рисунке 3.2.



	TSQLQuery	Компонент для получения и изменения данных
	TSQLTransaction	Компонент для работы с транзакциями
	TSQLScript	Компонент для работы с большим количеством Sql кода
	TSQLConnector	Компонент для установления подключения к различным базам данных
	TMSSQLConnection	Компонент для подключения к СУБД MSSql (Microsoft SQL Server)
	TSvbaseConnection	Компонент для подключения к СУБД Sybase (Svbase SQL Server)
	TPQConnection	Компонент для подключения к СУБД PostgreSQL
	TPQTEventMonitor	Компонент для взаимодействия с событиями в PostgreSQL
	TOracleConnection	Компонент для работы с СУБД Oracle
	TODBCConnection	Компонент для доступа к данным посредством драйверов ODBC
	TMySQL40Connection	Компонент для подключения к СУБД MySQL версии 4.0
	TMySQL41Connection	Компонент для подключения к СУБД MySQL версии 4.1
	TMySQL50Connection	Компонент для подключения к СУБД MySQL версии 5.0
	TMySQL51Connection	Компонент для подключения к СУБД MySQL версии 5.1
	TMySQL55Connection	Компонент для подключения к СУБД MySQL версии 5.5
	TMySQL56Connection	Компонент для подключения к СУБД MySQL версии 5.6
	TSQLite3Connection	Компонент для подключения к СУБД SQLite
	PIBConnection	Компонент для подключения к СУБД Firebird. Interbase
	TFBAdmin	Компонент для управления сервером Firebird
	TFBEventMonitor	Компонент для работы с событиями Firebird. Interbase
	TSQLDBLibraryLoader	Компонент для загрузки библиотек доступа к данным

Рисунок 3.2 – Состав вкладки компонентов SQLdb

Для подключения приложения к динамической библиотеке *SQLite* на форме (*Form1*) (рисунок 3.3) следует разместить компонент соединения с базой данных *SQLite TSQLite3Connection*. Он позволит настроить подключение к имеющейся в распоряжении пользователя базе данных *SQLite*.

В качестве примера базы данных следует принять существующую базу «*OBORUDOVANIE*», которая будет выдана преподавателем. Ее нужно разместить в папке *OBORUD* проекта.

Так как *SQLite* является системой управления локальной базой данных, то достаточно указать только путь к файлу базы с помощью компонента *OpenDialog1* и кодировку текста (*UTF8*) для работы, а также компонент для транзакции *TSQLTransaction*,

обеспечивающий целостность работы с данными, т. е. сохраняющий изменения (рисунок 3.3).

Дополнительно потребуется компонент *DataSource* из вкладки *Data Access*, который является посредником между компонентами для доступа к данным (*SQLQuery*) и компонентами для отображения данных (*DBGrid*).

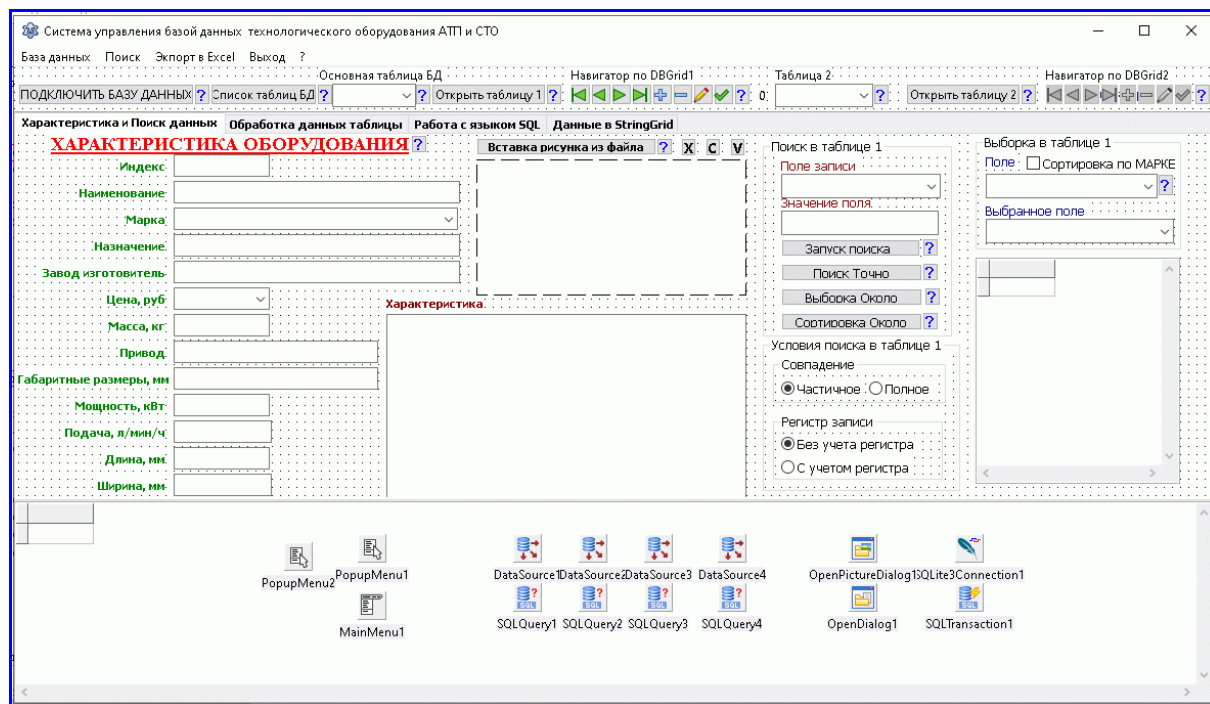


Рисунок 3.3 – Форма (*Form1*) и компоненты программы ABC_LAZARUS

Вставленные компоненты для работы с базой данных должны быть настроены, т. е. некоторые их свойства должны быть определены заранее с помощью Инспектора объектов или прописать в кодах процедур приложения (таблица 3.1).

Таблица 3.1 – Настройка компонентов приложения ABC_LAZARUS

Компонент	Настраиваемые свойства*
SQLite3Connection1	DatabaseName = OpenDialog1.FileName;
(настраивается в процедуре)	Transaction = SQLite3Transaction1;
SQLiteTransaction1	Charset := 'UTF8'
DataSource1	Database = SQLite3Connection1
	DataSet = SQLQuery1

<i>DataSource2</i>	<i>DataSet = SQLQuery2</i>
<i>DataSource4</i>	<i>DataSet = SQLQuery4</i>
<i>SQLQuery1</i>	<i>Database = SQLite3Connection1;</i> <i>Transaction = SQLTransaction1</i>
<i>SQLQuery2</i>	<i>Database = SQLite3Connection1;</i> <i>Transaction = SQLTransaction1</i>
<i>SQLQuery4</i>	<i>Database = SQLite3Connection1;</i> <i>Transaction = SQLTransaction1</i>
<i>DBGrid1</i>	<i>DataSource = DataSource1</i>
<i>DBGrid2</i>	<i>DataSource = DataSource2</i>
<i>DBGrid4</i>	<i>DataSource = DataSource2</i>
<i>DBGrid5</i>	<i>DataSource = DataSource4</i>
<i>OpenDialog1</i>	<i>DefaultExt=.db3</i>
<i>DBNavigator1</i>	<i>DataSource = SQLQuery1</i>
<i>DBNavigator2</i>	<i>DataSource = SQLQuery2</i>
<i>DBEdit</i>	<i>DataSource = DataSource1;</i> <i>DataField = «Наименование</i> <i>поля»</i>
<i>DBMemo</i>	<i>DataSource = DataSource1</i> <i>DataField = «Наименование</i> <i>поля»</i>
<i>DBImage</i>	<i>DataSource = DataSource1</i> <i>DataField = «Наименование</i> <i>поля»</i>

* остальные свойства компонентов можно оставить без изменения.

3.3.3 Подготовка механизма управления приложением

Управлять приложением можно с помощью меню, располагаемого сверху формы (*Form1*).

Следует создать главное меню программы с пунктами и подпунктами (рисунок 3.4), используя заранее вставленный компонент *MainMenu1*.

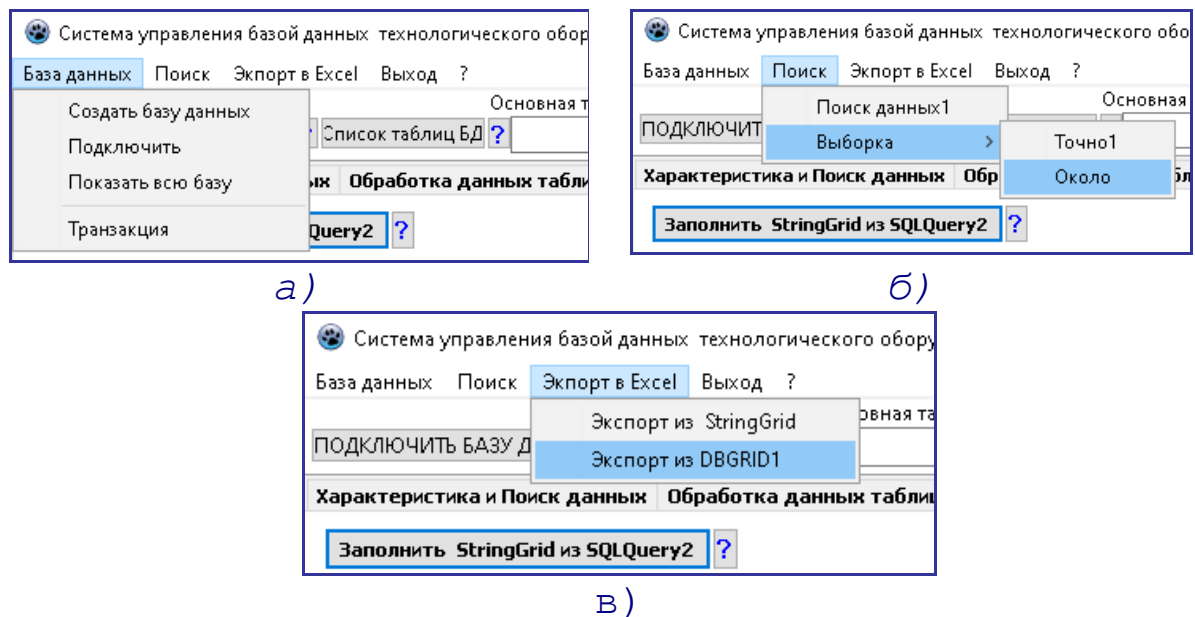


Рисунок 3.4 – Пункты главного меню приложения

Кроме меню для управления используют кнопки, упрощающие доступ к выполнению нужной процедуры, например, кнопка «Подключить базу данных» (см. рисунок 3.3).

При нажатии на пункт меню или кнопку выполняется пустая процедура (метод *Click*), поэтому нужно вставить код с соответствующими действиями в заготовку для метода *Click* соответствующего компонента.

В метод пункта меню «База данных – Подключить» (рисунок 3.4 а) вставляем код первой процедуры разрабатываемого приложения.

Для облегчения понимания кода программы разработчики часто вводят поясняющие надписи, которые выделяют либо фигурными скобками, например {это текст пояснения на нескольких строках} или двумя наклонными линиями – //это текст пояснения на одной строке.

// Код процедуры «Подключение к базе данных».


```

Begin
//Определение пути к проекту.
GetDir(0,MyDir);
OpenDialog1.InitialDir:= MyDir+'\oborud';
if OpenDialog1.Execute then
    // Указываем путь к базе данных.
    SQLite3Connection1.DatabaseName :=
    OpenDialog1.FileName;
    // Указываем рабочую кодировку.
    SQLite3Connection1.CharSet := 'UTF8';
    // Указываем менеджер транзакций.
    SQLite3Connection1.Transaction :=
    SQLiteTransaction1;
    // Пробуем подключиться к базе.
    try
        SQLite3Connection1.Open;
        SQLiteTransaction1.Active := True;
        Form1.Caption:= 'Система управления базой
данных технологического оборудования АТП и СТО
-> ' + OpenDialog1.FileName;
        // Выводим сообщение о ошибке.
    Except
        ShowMessage('Ошибка подключения к базе!');
    end;
end.

```

В вышеприведенном коде процедура **OpenDialog1.Execute** позволяет отрыть существующую базу данных, которая должна находиться в папке Oborud (рисунок 3.5) и присвоить свойству **DatabaseName** полное имя выбранной базы данных.

После вставки вышеприведенной процедуры нужно проверить работоспособность приложения и при необходимости устранить возникшие ошибки. *Проверять правильность сборки приложения следует после каждого изменения кода или добавления операторов.*

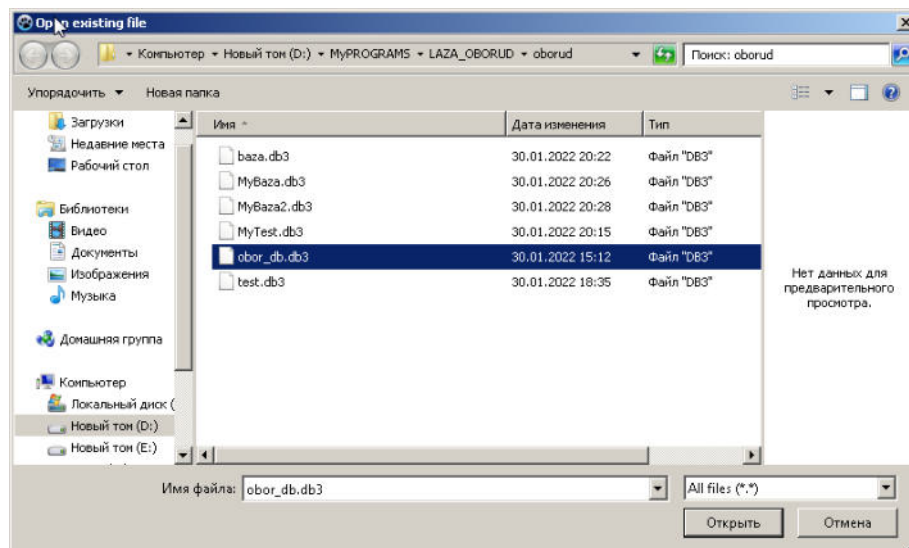


Рисунок 3.5 – Окно для открытия существующей базы данных

Каждая база данных может содержать несколько таблиц с записями. Для того, чтобы эти таблицы увидеть используют кнопку «Список таблиц» (см. рисунок 3.3). В метод *Click* указанной кнопки вставляют код следующей процедуры.

{ Процедура «Определения имен таблиц в базе данных и вывод их в списки ComboBox1 и ComboBox2».

var i:integer;

begin

If SQLite3Connection1.DatabaseName <> '' then
begin

SQLQuery3.Close; // Закрываем компонент.

SQLQuery3.Clear; // Очищаем компонент.

ComboBox1.Clear; // Очищаем компонент.

ComboBox2.Clear; // Очищаем компонент.

// Пробуем подключиться к базе.

try

SQLQuery3.SQL.Text:='SELECT * FROM
sqlite_master WHERE type= "table";

// Выполняем запрос.

SQLQuery3.ExecSQL;

except

// Выводим сообщение о ошибке.

ShowMessage ('Неправильный запрос');

Exit;

end;

//Подтверждаем изменения.

```

SQLTransaction1.Commit;
SQLQuery3.Open;
i:=0;
SQLQuery3.First;
// Создаем цикл вставки имен таблиц в списки ComboBox.
While not SQLQuery3.EOF do
begin
if
SQLQuery3.FieldName('tbl_name').AsString<>
'sqlite_sequence' then
begin
ComboBox1.Items[i]:=
SQLQuery3.FieldName('tbl_name').AsString;
ComboBox2.Items[i]:=
SQLQuery3.FieldName('tbl_name').AsString;
Inc(i);
end;
// Переходим на следующую запись.
SQLQuery3.Next;
end;
end else ShowMessage('Выберите БАЗУ
ДАННЫХ!!!');
end.

```

После создания списка таблиц следует открыть свернутый список и выбрать имя основной таблицы (рисунок 3.3).

Просмотр данных этой таблицы базы данных осуществляется с помощью настроенного компонента *SQLQuery1*, у него в качестве свойства *Database* должен быть указан компонент *SQLite3Connection1*, а в качестве параметра *Transaction* компонент *SQLTransaction1* (таблица 3.1).

Так как в проектируемом приложении предполагается использовать систему *SQLite*, далее будет рассматриваться применение запросов на языке *SQL* для выполнения необходимых действий с базой данных.

Используя синтаксис языка *SQL* нужно составить запрос для выборки данных.

```

// Процедура «Показ данных основной таблицы «oborud_db»».
begin
SQLQuery1.Close;
SQLQuery1.SQL.Text:='SELECT * FROM «имя таблицы»';

```

SQLQuery1.Open;
end.

После этого будет открыт для обозрения и анализа набор данных выбранной таблицы. Для работы с таблицей компонент *SQLQuery1* содержит базовый набор функций и свойств.

Передвижение по строкам (записям) данных позволяют выполнить следующие методы: *First* (перейти на самую первую запись), *Last* (перейти на самую последнюю запись), *Prior* (перейти на предыдущую запись), *Next* (перейти на следующую запись).

Для реализации этих методов перемещения по таблице используют визуальный компонент *TDBNavigator*, который имеет соответствующие кнопки управления. Обеспечение полной функциональности некоторых кнопок требует создания дополнительных кодов.

Отображение данных таблицы возможно компонентами из вкладки *Data Controls*. Они созданы на базе обычных визуальных компонентов и предназначены для отображения полей различных типов таблицы базы данных.

Что бы отобразить данные на форме (*Form1*) размещают компонент *DataSource1* и визуальный компонент *DBGrid1*. У компонента *DataSource1* свойству *DataSet* присваивают имя компонента *SQLQuery1*, а у компонента *DBGrid1* вставляют свойство *DataSource* равное *DataSource1*. Если открыть источник данных, т. е. выполнить процедуру *SQLQuery1.Open*, то в таблице отобразятся записи базы данных (рисунок 3.6).

Иногда требуется более детальное представление характеристики оборудования, тогда применяют компоненты *TDBEdit* (для показателей), *TDBMemo* (для описания), *TDBImage* (для рисунков), настройка которых приведена в таблице 3.1.

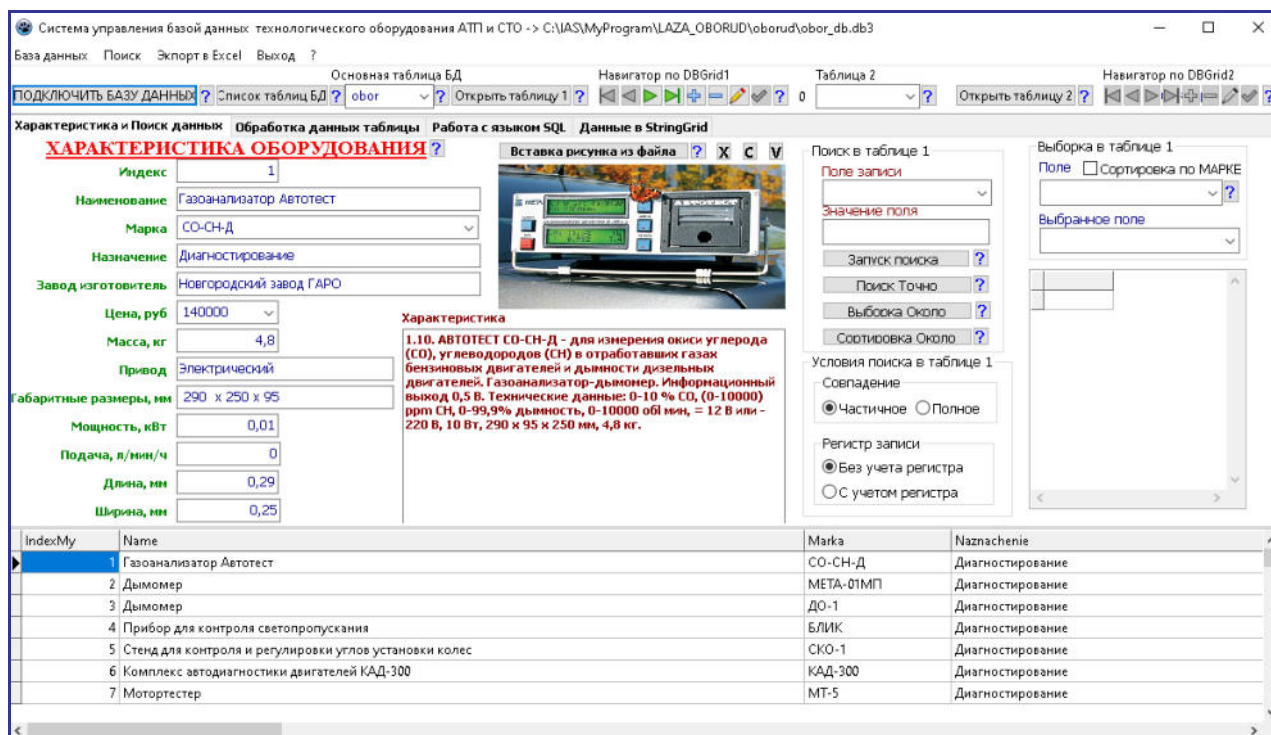


Рисунок 3.6 – Главное окно программы (вкладка «Характеристика и поиск данных»)

3.3.4 Добавление, удаление и редактирование записей таблицы базы данных

Самые простейшие действия, совершаемое с таблицей базы данных это добавление информации, удаление и редактирование полей базы данных. При этом совершаются действия со всеми элементами записи, представляющей собой строку таблицы, имеющей множество полей – переменных, в которые вставляются, удаляются, редактируются определенные значения, тексты или рисунки (рисунок 3.7).

Наиболее простой путь выполнения этих действий это использование предварительно настроенного навигатора *TDBNavigator*. Настройка заключается в подготовке процедуры *Click* этого компонента.

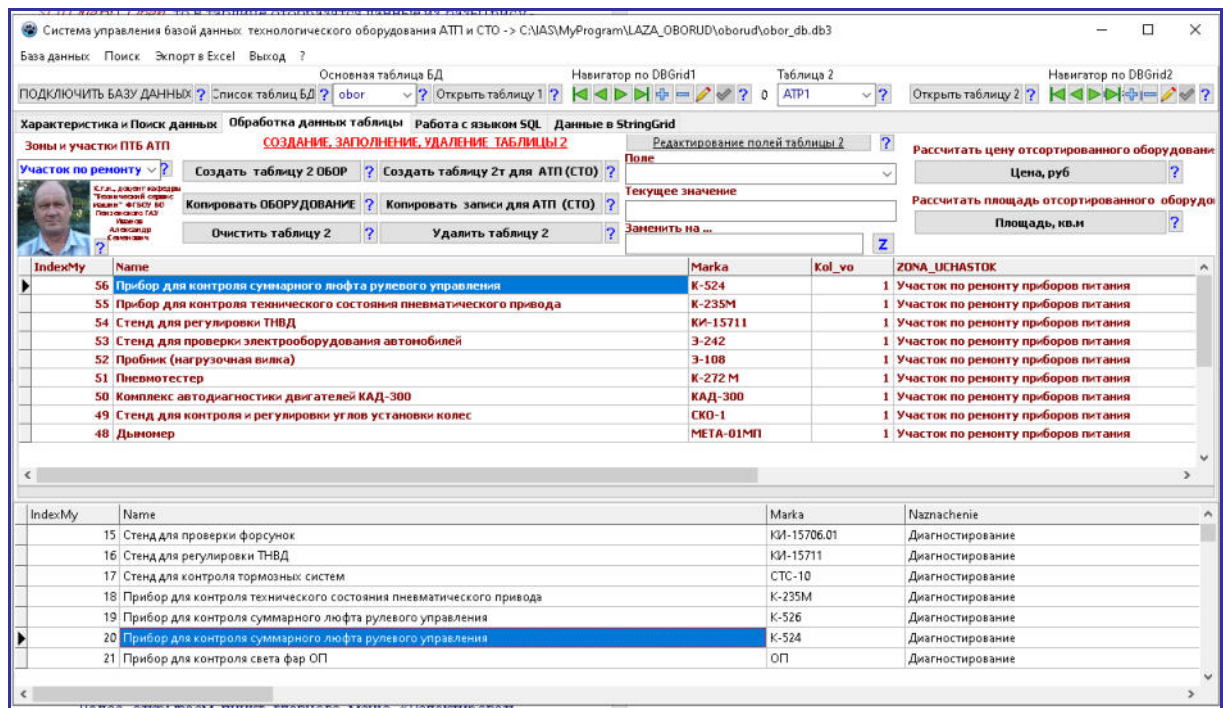


Рисунок 3.7 – Пункт главного меню «Редактировать»

```
// Настройка процедуры Click компонента DBNavigator1.
begin
    case button of
        nbInsert: // Кнопка «Вставка».
        begin
            ShowMessage('Нажата кнопка навигатора INSERT');
            // Открываем процедуру добавления данных.
            SQLQuery1.Append;
            // Присваиваем записи текстовое значение.
            SQLQuery1.Fields.FieldName('Name').AsString := '
Новое оборудование';
            // Сохраняем данные.
            SQLQuery1.Post;
            // Отправляем изменения в базу данных.
            SQLQuery1.ApplyUpdates;
        end;
        nbDelete: // Кнопка «Удаление».
        begin
            // Удаляем запись.
            SQLQuery1.Delete;
            // Отправляем изменения в базу данных.
            SQLQuery1.ApplyUpdates;
        end;
        nbPost: // Кнопка «Сохранение».
        begin
```



```

ShowMessage('Нажата кнопка навигатора POST');
SQLQuery1.ApplyUpdates;
end;
end;
end.

```

В данной процедуре для кнопок *Insert*, *Delete* и *Post* заданы вышеопределенные действия, которые обеспечивают выполнение результатов изменения в базе данных.

Применение языка управления базами данных *SQL* является наиболее универсальным средством, так как *SQL* используют практически все языки программирования, в том числе и *Pascal*, при обращении к самым различным базам данных.

Достаточно изучить основные команды *SQL*, методики их формирования и пользователь сможет получить всю необходимую информацию из базы данных, а также производить действия добавления, изменения, удаления, поиска, сортировки, группировки нужных данных.

Поэтому изменение значений полей записей таблицы, также производится с помощью инструкций языка *SQL*. Для этого следует определиться с полями, которые будут изменяться. Затем составить запрос на изменение с помощью инструкции **«UPDATE ... SET ... WHERE»**. Запрос выполняется с помощью команды *ExecSQL* и подтверждается изменение командой *Commit*.

На вкладке «Обработка данных таблицы», вставляем компоненты *Edit1*, *Edit2*, компонент *Pole1* (*ComboBox*) и кнопку «Z». В методе *Click* этой кнопки размещаем процедуру для замены значения компонента *Edit2* на значение компонента *Edit1* для поля указанного с помощью компонента *Pole1*.

```

// Процедура редактирования записей таблицы с помощью UPDATE.
Begin
    SQLQuery2.Close;
    SQLQuery2.SQL.Text:='UPDATE ' + ComboBox2.Text+
    ' SET ' + Pole1.Text+ ' = '' + Edit1.Text + ''
    where ' + Pole1.Text+ ' = '' + Edit2.Text+'''';
    SQLQuery2.ExecSQL; // Выполняем запрос.
end;
end.

```

Изменение поля «Рисунок», имеющего тип *BLOB* производится в два этапа: вначале графический файл с рисунком (*.jpg) вставляется в поле компонента *DBImage* (вкладка «Характеристика и поиск данных»), затем заканчивается редактирование и происходит отправка изменений в базу данных.

Для выполнения указанных действий следует использовать компонент *OpenPictureDialog1* и кнопку «Вставка рисунка из файла». В методе *Click* кнопки размещают процедуру вставки рисунка.

```
// Процедура вставки рисунка.
    if OpenPictureDialog1.Execute then
begin
    SQLQuery1.Edit;    // Входим в режим редактирования.
    SQLQuery1.Post;    // Заканчиваем режим редактирования.
    SQLQuery1.ApplyUpdates; //Отправляем изменения в БД.
end;
end

    else ShowMessage('Откройте таблицу 1')
end.

begin
if SQLQuery1.Active=True then
begin
    OpenPictureDialog1.InitialDir:=MyDir+' \ris';
    if OpenPictureDialog1.Execute then
begin
        // Входим в режим редактирования.
        SQLQuery1.Edit;
        // Присваиваем полю Рисунок загруженный Filename.
        (SQLQuery1.fieldbyname('Рисунок') as
        TBlobField).LoadFromFile(OpenPictureDialog1.FileName);
        // Заканчиваем режим редактирования.
        SQLQuery1.Post;
        //Отправляем изменения в базу данных.
        SQLQuery1.ApplyUpdates;
    end;
end

    else ShowMessage('Откройте таблицу 1')
end.
```


Чтобы упорядочить процесс выбора готового рисунка, нужно заранее создать папку «*Ris*» в каталоге проекта, в которую поместить все рисунки необходимые для таблицы. *Внимание! Размер рисунков должен быть как можно меньше. Максимальный размер 50 Кб.*

Чтобы удалить рисунок из таблицы следует вставить кнопку «*X*» и вписать в ее метод *Click* процедуру удаления рисунка.

```
// Процедура удаления рисунка из таблицы.  
begin  
    if SQLQuery1.Active=True then  
begin  
    // Входим в режим редактирования.  
    SQLQuery1.Edit;  
    (SQLQuery1.fieldbyname('Рисунок') as  
    TBlobField).Clear;  
    // Заканчиваем режим редактирования.  
    SQLQuery1.Post;  
    //Отправляем изменения в базу.  
    SQLQuery1.ApplyUpdates;  
end else ShowMessage('Откройте таблицу 1')  
end.
```

Для переноса рисунка из одной записи в другую скопируйте в буфер нужный рисунок, нажав на кнопку «*C*», перейдите к соответствующей записи, используя навигатор и вставьте рисунок, нажав на кнопку «*V*».

Предварительно следует обеспечить функционал кнопок «*C*» и «*V*», используя ниже приведенные процедуры.

```
{ Процедура копирования рисунка в буфер обмена (для метода Click  
кнопки «C»).}  
Begin  
if SQLQuery1.Active=True then MyStr :=  
SQLQuery1.CreateBlobStream(SQLQuery1.FieldByName('Р  
исунок'), bmRead)  
else ShowMessage('Откройте таблицу 1')  
end.  
  
// Процедура вставки рисунка в таблицу из буфера обмена (для  
метода Click кнопки «V».)  
begin  
if SQLQuery1.Active=True then
```

```

begin
    // Входим в режим редактирования.
SQLQuery1.Edit;
    (SQLQuery1.fieldbyname('Рисунок') as
TBlobField).LoadFromStream(MyStr);
    // Заканчиваем режим редактирования.
SQLQuery1.Post;
    //отправляем изменения в базу.
SQLQuery1.ApplyUpdates;
end
    else ShowMessage('Откройте таблицу 1');
end.

```

3.3.5 Поиск данных

Поиск данные является одной из необходимых операций при работе с таблицами баз данных. Поиск осуществляется по конкретному полю или полям записи с указанием значения поля и условий поиска (полное совпадение, учет регистра и т. п.).

Один из способов поиска предполагает использование встроенной функции компонента *SQLQuery*, в качестве аргументов которой указывают поле и значение, например *SQLQuery1.Locate(Pole.Text, Znachenie.Text, [loCaseInsensitive, loPartialKey])*.

Для выполнения поиска по различным колонкам (полям) и определенным значениям целесообразно подготовить панель с условиями поиска, позволяющую в процессе работы с приложением оперативно менять условия (рисунок 3.8).

Рисунок 3.8 – Панель с условиями поиска

```
// Процедура поиска данных.
begin
    if (Pole_Poiska.Text<>'') and
    (Znachenie_Poiska.Text<>'') then
    begin
        If (RadioGroup1.ItemIndex=0) and
        (RadioGroup2.ItemIndex=0) then
        SQLQuery1.Locate(Pole_Poiska.Text,Znachenie_Pois
        ka.Text,[loCaseInsensitive, loPartialKey]);
        If (RadioGroup1.ItemIndex=1) and
        (RadioGroup2.ItemIndex=0) then
        SQLQuery1.Locate(Pole_Poiska.Text,Znachenie_Pois
        ka.Text,[ loPartialKey]);
        If (RadioGroup1.ItemIndex=1) and
        (RadioGroup2.ItemIndex=1) then
        SQLQuery1.Locate(Pole_Poiska.Text,Znachenie_Pois
        ka.Text,[loCaseInsensitive]);
        If (RadioGroup1.ItemIndex=0) and
        (RadioGroup2.ItemIndex=1) then
        SQLQuery1.Locate(Pole_Poiska.Text,Znachenie_Pois
        ka.Text,[]);
        //'Name' - название нужного столбца.
        //'Нужное значение' - то что следует найти.
        //'LoCaseInsensitive - игнорируем регистр записи.
```

```
//LoPartialKey - частичное совпадение.
end else ShowMessage('Заполните условия поиска!');
end.
```

При использовании данной процедуры маркер выбранной записи переместиться на найденную запись, если искомым данным не будет обнаружено, то маркер переместиться на последнюю запись.

3.3.6 Выборка и сортировка данных таблицы

Выборка данных выполняется по инструкции языка *SQL* «**SELECT ... FROM ... WHERE**» с указанием значения поля по которому следует сформировать выборку.

Процедуру, приведенную ниже следует поместить в метод *Click* пункта главного меню «Поиск – Выборка Точно» или кнопки «Выборка Точно» на панели (вкладка «Характеристика и поиск данных») (рисунок 3.8).

```
// Процедура выполнения выборки точных данных.
begin
if (Pole_Poiska.Text<>'') and
(Znachenie_Poiska.Text<>'') then
begin
// Закрываем компонент.
SQLQuery1.Close;
// Создаем запрос на поиск данных.
SQLQuery1.SQL.Text := 'select * from
'+ComboBox1.Text+' where '+ Pole_Poiska.Text+'
=: '+Pole_Poiska.Text;
//Правильное заполнение.
SQLQuery1.ParamByName(Pole_Poiska.Text).AsString :=
Znachenie_Poiska.Text;
// Выполняем запрос.
SQLQuery1.Open;
end else ShowMessage('Заполните условия
поиска!');
end.
```

Данный запрос применяется для поиска по точным данным. Между параметром поиска и значением установлен знак равенства *Pole_Poiska.Text+':='+Pole_Poiska.Text.*

Если требуется найти все похожие данные, то применяется функция *SQL LIKE*, позволяющая задать примерный поиск

значения в записях для конкретного поля, например инструкция **LIKE 'M%'** выведет все имена, начинающиеся на букву М.

В следующей процедуре запрос осдержит выражение «**Pole_poiska.Text + ' LIKE: '+Pole_poiska.Text**», а расширение зоны поиска осуществляется путем добавления знака процента «%» при присвоении требуемого параметра

```
SQLQuery1.ParamByName(Pole.Text).AsString:=  
'%'+Znachenie.Text+'%'.
```

```
//Процедура примерной выборки.
```

```
begin
```

```
if (Pole_Poiska.Text<>'') and  
(Znachenie_Poiska.Text<>'') then
```

```
begin
```

```
// закрываем компонент.
```

```
SQLQuery1.Close;
```

```
// Создаем запрос на поиск данных.
```

```
SQLQuery1.SQL.Text := 'select * from  
' + ComboBox1.Text + ' where ' + POLE_Poiska.Text + '  
like :'+Pole_Poiska.Text;
```

```
//Правильное заполнение.
```

```
SQLQuery1.ParamByName(Pole_Poiska.Text).AsString  
:= Znachenie_Poiska.Text+'%';
```

```
// Выполняем запрос.
```

```
SQLQuery1.Open;
```

```
end else ShowMessage('Заполните условия  
поиска!');
```

```
end.
```

В СУБД *SQLite* оператор **LIKE** поддерживаться полностью только для латиницы, для кириллицы он работает только с соблюдением регистра букв.

Сортировка данных осуществляется с помощью инструкции «**SELECT ... FROM ... WHERE ... ORDER BY ...**». В панель (рисунок 3.7) добавляють кнопку «Сортировка Около», а затем в метод *Click* этой кнопки вставляют процедуру осуществляющую сортировку.

```
// Процедура сортировки.
```

```
begin
```

```
if (Pole_Poiska.Text<>'') and  
(Znachenie_Poiska.Text<>'') then
```

```

begin
    // Закрываем компонент.
SQLQuery1.Close;
    // Составляем запрос на поиск данных.
SQLQuery1.SQL.Text := 'select * from
'+ComboBox1.Text+' where '+POLE_Poiska.Text+'
like :'+POLE_Poiska.Text+' Order By '+
POLE_Poiska.Text;
    //Правильное заполнение.
SQLQuery1.ParamByName(Pole_Poiska.Text).AsString :=
Znachenie_Poiska.Text+'%';
    // Выполняем запрос.
SQLQuery1.Open;
end else ShowMessage('Заполните условия
поиска!');
end.

```

3.3.7 Транзакции

Транзакция – это распространение одного или нескольких изменений в базе данных. Транзакции в базах данных обеспечивают целостность и защищенность данных. Они позволяют изменить данные в базе либо вернуть предыдущее состояние данных в пределах транзакции.

Транзакции имеют следующие четыре стандартных свойства, обычно называемые аббревиатурой *ACID*.

Atomicity – обеспечение успешного завершения всех операций в рабочем модуле; в противном случае транзакция прерывается в момент сбоя, а предыдущие операции возвращаются в прежнее состояние.

Consistency – обеспечение правильного изменения базы данных при успешной транзакции.

Isolation – обеспечение возможности транзакциям работать независимо друг от друга и прозрачно.

Durability – обеспечивает, чтобы результат или эффект совершенной транзакции сохранялся в случае сбоя системы.

Управление транзакциями осуществляется командами:

BEGIN TRANSACTION – начать транзакцию;

COMMIT – сохранить изменения, можно использовать команду **END TRANSACTION**;

ROLLBACK – откат изменений.

Команды управления транзакциями используются только с командами *INSERT*, *UPDATE* и *DELETE*. Они не могут использоваться при создании таблиц или их удалении, поскольку эти операции автоматически фиксируются в базе данных.

Транзакцию можно запустить с помощью *BEGIN TRANSACTION* или просто команды *BEGIN*. Такие транзакции обычно сохраняются до появления следующей команды *COMMIT* или *ROLLBACK*. Однако транзакция будет также *ROLLBACK*, если база данных будет закрыта или произойдет ошибка. Ниже приведен простой синтаксис для начала транзакции.

Команда *COMMIT* — это транзакционная команда, используемая для сохранения изменений, вызванных транзакцией, в базу данных. Команда *COMMIT* сохраняет все транзакции в базе данных с момента последней команды *COMMIT* или *ROLLBACK*.

Команда *ROLLBACK* — это транзакционная команда, используемая для отмены транзакций, которые еще не были сохранены в базе данных. Команда *ROLLBACK* может использоваться только для отмены транзакций с момента выдачи последней команды *COMMIT* или *ROLLBACK*.

Пример работы с транзакциями из кода.

```
// Процедура запуска транзакций.  
begin  
SQLTransaction1.StartTransaction;  
try  
// Обрабатываем данные.  
SQLTransaction1.Commit;  
// Если нет ошибки, то вносим изменения в базу.  
except  
// При ошибке обработки данные возвращаются в исходное состояние.  
SQLTransaction1.Rollback;  
end;  
end.
```

Транзакции удобно использовать, когда необходимо загрузить в базу большое количество данных, так как транзакции ускоряют процесс работы с записями в сотни раз. Но не правильное использование транзакции может привести к зависанию приложений, работающих с базой.

После окончания работы с приложением, необходимо принудительно внести изменения в базу данных. Для этого

целесообразно поместить в метод *Close* формы *Form1* следующую запись

```
if SQLTransaction1.Active=True then  
SQLTransaction1.Commit.
```

3.3.8 Создание и удаление новой таблицы базы данных

Результатом работы над приложением по управлению базой данных «*OBORUDOVANIE*» должно быть обеспечение возможности создания таблиц с оборудованием для ремонтных мастерских автотранспортных предприятий или станций технического обслуживания автомобилей.

База данных *SQL* может содержать до 64 таблиц с записями ([приложение 2](#)). Для создания таблицы в данном примере использована процедура *Sozdat_TableATP* с запросом *SQL* в виде инструкции «*CREATE TABLE*» с последующим перечислением имен таблицы и полей с их типами. Ниже приведенную процедуру следует разместить внизу кода модуля и зарегистрировать в классе *TForm1* (*TForm1* = *class(TForm)*...

```
Sozdat_TableATP(name_mytable:string);).
```

{Процедура для создания таблицы соответствующей основной таблице базы данных «*OBORUDOVANIE*» .}

```
var s:string;
```

```
begin
```

```
    begin
```

```
        s:='CREATE TABLE '+name_mytable+ ' ';
```

```
        s:=s+'(IndexMy INTEGER PRIMARY KEY  
AUTOINCREMENT, ';
```

```
        s:=s+'Name VARCHAR(70), ';
```

```
        s:=s+'Marka VARCHAR(15), ';
```

```
        s:=s+'Kol_vo INTEGER, ';
```

```
        s:=s+'ZONA_UCHASTOK VARCHAR(70), ';
```

```
        s:=s+'Цена FLOAT, ';
```

```
        s:=s+'Мощность FLOAT, ';
```

```
        s:=s+'Подача FLOAT, ';
```

```
        s:=s+'Длина FLOAT, ';
```

```
        s:=s+'Ширина FLOAT) ';
```

```
        SQLQuery2.SQL.Text:=s;
```

```
        SQLQuery2.ExecSQL;
```

end;

end.

Далее нужно вставить кнопку «Создать таблицу 2 для АТП (СТО)» и в метод *Click* вписать следующую конструкцию из операторов:

begin

if ComboBox2.Text<>' ' then

Sozdat_TableATP(ComboBox2.Text) else

ShowMessage('Укажите имя таблицы, которую хотите создать!!!');

end.

После выполнения этой процедуры в базе данных «*Oborudovanie*» появится новая таблица. Чтобы ее увидеть в списке таблиц базы данных используйте кнопку «Список таблиц БД» (раздел 3.3.3).

Кроме возможности создания новой таблицы обычно предусматривают функцию удаления ненужной таблицы из базы данных с помощью инструкции *Drop* языка *SQL*.

// Процедура удаления таблицы из базы данных.

begin

if ComboBox2.Text<>' ' then

begin

// Закрываем компонент.

SQLQuery2.Close;

// Составляем запрос на удаление данных.

**SQLQuery2.SQL.Text:='Drop TABLE ' +
ComboBox2.Text;**

// Выполняем запрос.

SQLQuery2.ExecSQL;

//Подтверждаем изменения.

SQLTransaction1.Commit;

ComboBox2.Text:=' ';

**end else ShowMessage('Выберите нужную таблицу
из списка!!!');**

end.

Как видно из кодов вышеприведенных процедур создание и удаление таблиц происходит при указании имени таблицы 2 в компоненте *ComboBox2*.

3.3.9 Добавление данных в таблицу с оборудованием мастерских АТП или СТО

Таблица с оборудованием мастерских (таблица 2) может создана путем копирования записей из компонента *DBGrid1*, т. е. путем выборки из основной таблицы базы данных (таблица 1).

Следует вставить кнопку «Копировать записи для АТП (СТО)», а затем в метод *Click* добавить процедуру копирования данных из компонента *DBGrid1* в компонент *DBGrid2*, т.е. в таблицу 2.

```
// Процедура копирования массива данных.  
var i, j: Integer;  
begin  
  If ComboBox2.Text <> '' then  
    Begin  
      ProgressBar1.Position:=0;  
      SQLQuery1.First;  
      SQLQuery2.First;  
      ProgressBar1.Max:=DBGrid1.DataSource.DataSet.  
RecordCount;  
      for j := 0 to  
DBGrid1.DataSource.DataSet.RecordCount-1 do  
        begin  
          SQLQuery2.Insert;  
          SQLQuery2.Fields.FieldName('Name').AsString:=SQLQuery  
1.Fields.FieldName('Name').AsString;  
          SQLQuery2.Fields.FieldName('Марка').AsString:=SQLQuer  
y1.Fields.FieldName('Марка').AsString;  
          SQLQuery2.Fields.FieldName('Kol_vo').AsString:='1';  
          SQLQuery2.Fields.FieldName('ZONA_UCHASTOK').AsString:  
=ComboBox3.Text;  
          SQLQuery2.Fields.FieldName('Цена').AsString:=SQLQuery  
1.Fields.FieldName('Цена').AsString;  
          SQLQuery2.Fields.FieldName('Мощность').AsString:=SQLQ  
uery1.Fields.FieldName('Мощность').AsString;  
          SQLQuery2.Fields.FieldName('Подача').AsString:=SQLQue  
ry1.Fields.FieldName('Подача').AsString;  
          SQLQuery2.Fields.FieldName('Длина').AsString:=SQLQuer  
y1.Fields.FieldName('Длина').AsString;  
          SQLQuery2.Fields.FieldName('Ширина').AsString:=SQLQue  
ry1.Fields.FieldName('Ширина').AsString;
```

```

        SQLQuery2.Next;
        SQLQuery1.Next;
        SQLQuery2.ApplyUpdates;
        ProgressBar1.Position:=ProgressBar1.Position+ 1;
    end;
    //Подтверждаем изменение.
    SQLTransaction1.Commit;
    ProgressBar1.Position:=0;
end else ShowMessage('Выберите нужную таблицу
из списка!!!');
end.

```

Записи можно добавлять в таблицу выборочно. Для этого следует соединить с компонентом *DBGrid1* компонент *PopUpMenu1* (всплывающее меню) и вставить в него пункт «Добавить запись к ведомости оборудования ПТБ АТП». В метод *Click* этого пункта поместить процедуру вставки выбранной записи.

```

// Процедура вставки единичной записи.
begin
    SQLQuery2.Insert;
    SQLQuery2.Fields.FieldName('Name').AsString:=S
    QLQuery1.Fields.FieldName('Name').AsString;
    SQLQuery2.Fields.FieldName('Marka').AsString:=
    SQLQuery1.Fields.FieldName('Marka').AsString;
    SQLQuery2.Fields.FieldName('Kol_vo').AsString:
    ='1';
    SQLQuery2.Fields.FieldName('ZONA_UCHASTOK').As
    String:=ComboBox3.Text;
    SQLQuery2.Fields.FieldName('Цена').AsString:=S
    QLQuery1.Fields.FieldName('Цена').AsString;
    SQLQuery2.Fields.FieldName('Мощность').AsStrin
    g:=SQLQuery1.Fields.FieldName('Мощность').AsSt
    ring;
    SQLQuery2.Fields.FieldName('Подача').AsString:
    =SQLQuery1.Fields.FieldName('Подача').AsString
    ;
    SQLQuery2.Fields.FieldName('Длина').AsString:=
    SQLQuery1.Fields.FieldName('Длина').AsString;

```

```

SQLQuery2.Fields.FieldName('Ширина').AsString:
=SQLQuery1.Fields.FieldName('Ширина').AsString
;
    SQLQuery2.Post;
    sqlquery2.ApplyUpdates;
end;

```

3.3.10 Обработка полей таблицы базы данных

Отсортированные записи таблиц можно обработать с целью получения нужной информации, например для определения стоимости оборудования или площади занимаемой оборудованием.

Так как в таблице есть поля «Длина» и «Ширина» оборудования, то можно, создав цикл прохода по всем записям таблицы **«while not SQLQuery2.EOF do»** и рассчитав площадь каждого оборудования, определить суммарную площадь всего отсортированного оборудования (см. следующую процедуру).

```

// Процедуры обработки данных таблицы.
begin
    Ploshad:=0; i:=0;
    SQLQuery2.First;
    ProgressBar1.Position:=0;
    while not SQLQuery2.EOF do
// Цикл работает до конца набора данных.
begin
    Ploshad:= Ploshad +
SQLQuery2.Fields.FieldName('Длина').AsInteger*
SQLQuery2.Fields.FieldName('Ширина').AsInteger;
    SQLQuery2.Next;
// Переходим на следующую запись.
    Inc(i);
    ProgressBar1.Position:=
Round(100*i/SQLQuery2.RecordCount);
end;
    Button23.Caption:=' Общая площадь оборудования
'+IntToStr(Round(Ploshad/1000000)) + ' кв.м';
end.

```

Используя данный прием можно произвести практически все необходимые расчеты с использованием данных таблицы.

3.3.11 Экспорт данных таблицы в приложение *Microsoft Office Excel*

Для соединения проектируемого приложения с программой *Microsoft Office Excel* следует вставить в список модулей приложения модуль *ComObj*.

Экспорт данных производим через компонент *StringGrid1*. Расположив его во вкладке «Данные в *StringGrid*» и разместив рядом кнопку «Заполнить *StringGrid*» (рисунок 3.9), вносим в метод *Click* кнопки следующую процедуру.

```
{ Процедура заполнения компонента StringGrid выборкой данных
таблицы 2 (SQLQuery2).}
var i, j:integer;
begin
    if SQLQuery2.Active=True then
        begin
            // Задаемся количеством строк и колонок StringGrid1.
            StringGrid1.RowCount:=DBGrid2.DataSource.DataSet.
RecordCount;
            StringGrid1.ColCount:=DBGrid2.Columns.Count;
            // Переходим к первой записи таблицы.
            DBGrid2.DataSource.DataSet.First;
            // Запускаем цикл вывода данных таблицы в StringGrid.
            for j:=0 to StringGrid1.RowCount-1 do
                begin
                    // Выводим значения полей j-ой записи.
                    for i:=0 to StringGrid1.ColCount-1 do
                        StringGrid1.Cells[i,j]:=DBGrid2.Columns[i].
Field.AsString;
                    // Переходим к следующей записи таблицы.
                    DBGrid2.DataSource.DataSet.Next;
                end;
            // Выводим названия столбцов в StringGrid.
            for i:=0 to StringGrid1.ColCount-1 do
                StringGrid1.Cells[i,0]:=DBGrid2.Columns[i].
FieldName;
            end else ShowMessage('Откройте таблицу 2. ');
        end.
```

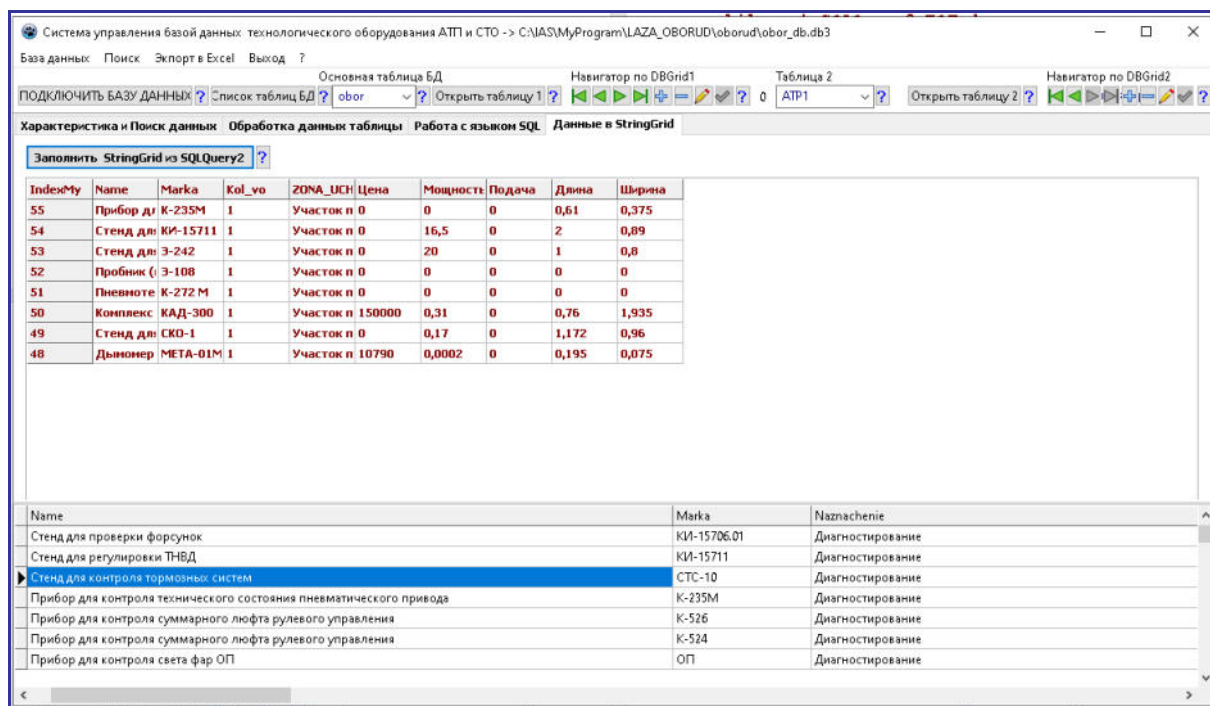


Рисунок 3.9 – Главное окно программы (вкладка «Данные в StringGrid»)

Далее нужно соединиться с приложением *Excel* и сделать его невидимым, что несколько повысит производительность работы с приложением.

Затем можно вывести наименование листа *Excel* и задать некоторые его параметры (высота строки, ширина столбца, параметры шрифта).

Чтобы избежать знаков вопроса вместо букв кириллицы при выводе данных из приложения в *Excel*, следует использовать функцию перекодировки *utf8Decode*.

В заключение нужно включить видимость *Excel*.

В результате вся информация из компонента *StringGrid1* будет выведена на лист *Microsoft Office Excel* (рисунок 3.10) помощью нижеприведенной процедуры, занесенной в метод *Click* пункта главного меню «Экспорт в *Excel* - Экспорт из *StringGrid*».

```
//Процедура экспорта данных в Microsoft Office Excel из StringGrid1.
```

```
//Нужен модуль ComObj.
```

```
Var
```

```
ExcelApp, ExcelSheet, ExcelCol, ExcelRow: Variant;
```

```
Size: Byte;
```

```
i, j, N: Word;
```

```
begin
```

```

// Соединяемся с Excel.
ExcelApp:=CreateOleObject('Excel.Application');
// Задаем невидимость Excel.
ExcelApp.Visible:=False;
// Добавляем строки.
ExcelApp.Workbooks.Add(-4167);
// Вводим имя листа Excel.
ExcelApp.Workbooks[1].Worksheets[1].Name:=
'Report';
ExcelCol:=ExcelApp.Workbooks[1].Worksheets
['Report'];
// Задаем высоту строки.
Size:=StringGrid1.DefaultRowHeight;
N:=StringGrid1.ColCount-1;
// Вводим ширину столбца.
For j:=0 To N Do
ExcelCol.Columns[j+1].ColumnWidth:=Size;
ExcelRow:=ExcelApp.Workbooks[1].Worksheets
['Report'].Rows;
// Устанавливаем параметры шрифта.
ExcelRow.Rows[1].Font.Bold:=True;
ExcelSheet:=ExcelApp.Workbooks[1].Worksheets
['Report'];
// Выводим StringGrid в Excel.
For i:=0 To StringGrid1.RowCount-1 Do
    For j:=0 To StringGrid1.ColCount-1 Do
ExcelSheet.Cells[i+1, j+1]:=
WideString(utf8Decode(StringGrid1.Cells[j, i]));
// Пример вывода строки в указанное место листа Excel.
ExcelApp.ActiveSheet.Cells.Item[1, 3].Value
:=WideString(utf8Decode('Извлечение из базы
данных')));
{ ExcelApp.ActiveSheet.Range['d2', 'd2'].Value
:=WideString(utf8Decode('Извлечение из базы данных'));}
// Возвращаем видимость Excel.
ExcelApp.Visible := true;
end.

```

Microsoft Excel - Лист1

Файл Правка Вид Вставка Формат Сервис Данные Справка

Введите вопрос

D5 K-235M

	D	E	F	G	H	I	J	K
	Измещение из базы данных			Цена	Масса	Привод	Габариты	Мощность
1	Name	Marka	Naznachenie	Zavod				
2	Прибор для контроля светопропускания	БЛИК	Диагностирование	Новгородский завод ГАРО	0	2	Электрический	220x75x155
3	Прибор для проверки карбюраторов	ППК	Ремонт и ТО системы питания	Новгородский завод ГАРО	0	24	Гидравлический	450x345x640
4	Прибор для контроля технического состояния	K-235M	Диагностирование	Новгородский завод ГАРО	0	19	Пневматический	610 x 375 x 115
5	Прибор для контроля суммарного люфта руля	K-526	Диагностирование	Новгородский завод ГАРО	0	3	Электрический	415x145x127
6	Прибор для контроля суммарного люфта руля	K-524	Диагностирование	Новгородский завод ГАРО	0	0,7	Электрический	350x135x160
7	Прибор для контроля света фар ОП	ОП	Диагностирование	Новгородский завод ГАРО	0	35	Электрический	660 x 590 x 1770
8	Прибор для испытания форсунок дизельных С-50 "Моторная"		Диагностирование		0	0	Электрический	
9	Прибор для прерывателей распределителя 3-213		Диагностирование		0	0	Электрический	
10	Прибор для проверки гидроусилителя руля K-405		Диагностирование		0	0	Электрический	
11	Прибор для проверки переднего моста авто T-1		Диагностирование		0	0	Электрический	
12	Прибор для проверки рулевого управления K-187		Диагностирование		0	0	Электрический	
13	Прибор для удаления воздуха из тормозной системы O-6		Разборочно-сборочные работы		0	0	Электрический	
14	Прибор для определения высыхания лакокрасочных работ ШГ-1		Малые работы		0	0	Электрический	
15	Прибор для определения технического состояния K-436		Ремонт и ТО системы питания		0	0	Электрический	570 x 500 x 465
16	Прибор для шлифовки клапанных гнезд 2447		Агрегатные работы		0	0	Электрический	
17	Прибор для проверки топливных насосов и 577Б		Ремонт и ТО системы питания		0	0	Электрический	365 X 320 X 500
18	Прибор для проверки упругости пружин 357		Ремонт и ТО системы питания		0	0	Электрический	160 X 350 X 16
19	Прибор для проверки и регулировки установки K-303		Диагностирование		0	0	Электрический	
20	Прибор для проверки якорей генераторов и 3-236		Электротехнические работы		0	0	Электрический	
21					0	0	Электрический	

Report /

Готово

Рисунок 3.10 – Результат вывода данных в приложение Microsoft Office Excel

Применяя выше перечисленные рекомендации и процедуры можно подготовить приложение для выполнения основных действий по управлению базой данных. Получив определенный опыт по сборке приложения целесообразно совершенствовать его для расширения функциональных возможностей и повышения производительности