Лабораторная работа 8

Администрирование и динамическое соединение с базой данных

Цель работы: ознакомиться с функциональными возможностями администрирования СУБД MySQL; изучить возможность динамического определения ODBC драйвера и использования его для соединения с базой данных; разработать приложение с возможностью работы с базой данных используя динамически формируемую строку подключения.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Ознакомиться с теоретическими сведениями.
2. Ознакомится с аспектами управления пользователями на уровне СУБД **MySQL**.
3. Создать двух пользователей, одному из которых будет доступна возможность редактирования таблицы **storage_transfers**, а другому только просмотр.
4. Реализовать процедуру определения **ODBC** драйвера **MySQL**.
5. Реализовать механизм соединения с базой данных.
6. Реализовать форму, на которой будет происходить редактирование данных таблицы **storage_transfers** .
7. Ответить на контрольные вопросы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В данной лабораторной работе будут рассмотрены вопросы, связанные с администрированием базы данных, включая управление учетными записями на уровне СУБД, а также реализацию механизма автоматического определения ODBC драйвера и использования его для соединения с базой данных, а также редактированием таблицы с двойной связью.

8.1. Администрирование базы данных

Правильное администрирование учетных записей пользователей и их привилегий в значительной степени способствует повышению безопасности базы данных. Для обеспечения высокого уровня безопасности БД мы будем использовать менеджер безопасности **Security Manager**, интегрированный в программу **dbForge Studio**.

Менеджер безопасности позволяет создавать, редактировать или удалять учетные записи пользователей, предоставлять или отменять привилегии как на глобальном уровне, так и на уровне объектов с использова-

нием графического интерфейса (хотя данные операции могут быть проделаны и с помощью **SQL**-запросов в консоли **MySQL**).

Окно менеджера безопасности (рис. 8.1) состоит из двух частей: левая показывает список пользователей, правая позволяет управлять учетной записью выбранного пользователя. Выберем меню **База данных**, и щелкнем на подпункт **Менеджер безопасности**.

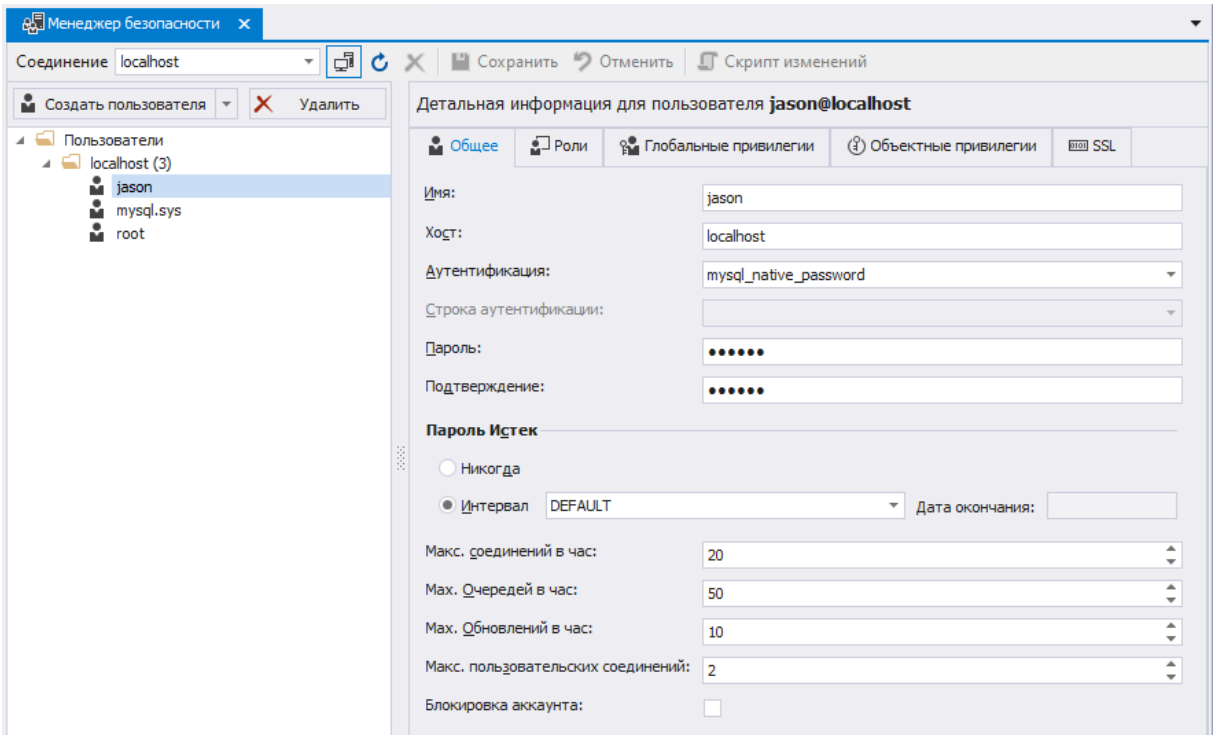

Рис. 8.1. Окно менеджера безопасности

Блок управления учетной записью включает в себя пять вкладок:

– **Общее** – содержит имя пользователя, хост, пароль, максимальное число соединений/запросов/обновлений;

– **Роли** – содержит роли, которые могут быть применены к учетной записи пользователя;

– **Глобальные привилегии** – позволяет задавать глобальные привилегии учетной записи;

– **Объектные привилегии** – позволяет задавать привилегии учетной записи для конкретного объекта БД;

– **SSL** (**Secure Sockets Layer** – уровень защищенных сокетов) – содержит опции соединений для учетной записи.

С помощью менеджера безопасности создадим новую учетную запись и назначим для нее пароль. Если бы для этих целей использовалась консоль, то пришлось бы вводить следующие команды:

```
CREATE USER 'jason' @'%';
GRANT USAGE ON *.* TO 'jason' @'%' IDENTIFIED BY
```

```
     'testpassword' REQUIRE SSL WITH
MAX_QUERIES_PER_HOUR 20
MAX_UPDATES_PER_HOUR 50
MAX_CONNECTIONS_PER_HOUR 10
MAX_USER_CONNECTIONS 2;
SET PASSWORD FOR 'jason' @'localhost' =
     PASSWORD ('testpassword')
```

На вкладке **Роли** можно выбрать группу, к которой будет относиться учетная запись пользователя. Это позволяет назначить привилегии без их детального выбора (рис. 8.2).

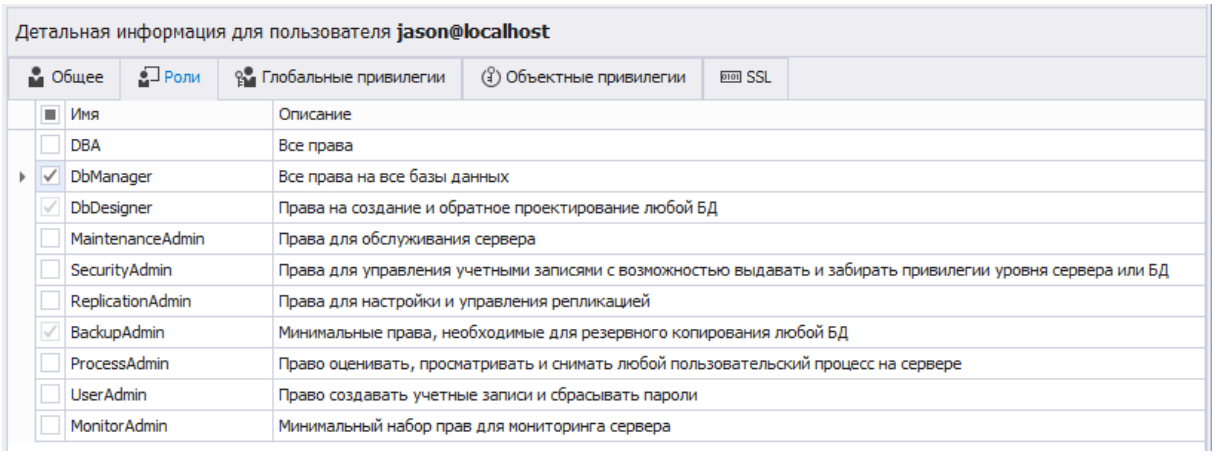

Рис. 8.2. Назначение роли учетной записи

Чтобы увидеть, какие привилегии имеются у каждой группы, перейдем на вкладку **Глобальные привилегии** (рис. 8.3). Установленные флажки показывают, какие привилегии определены для пользователя.

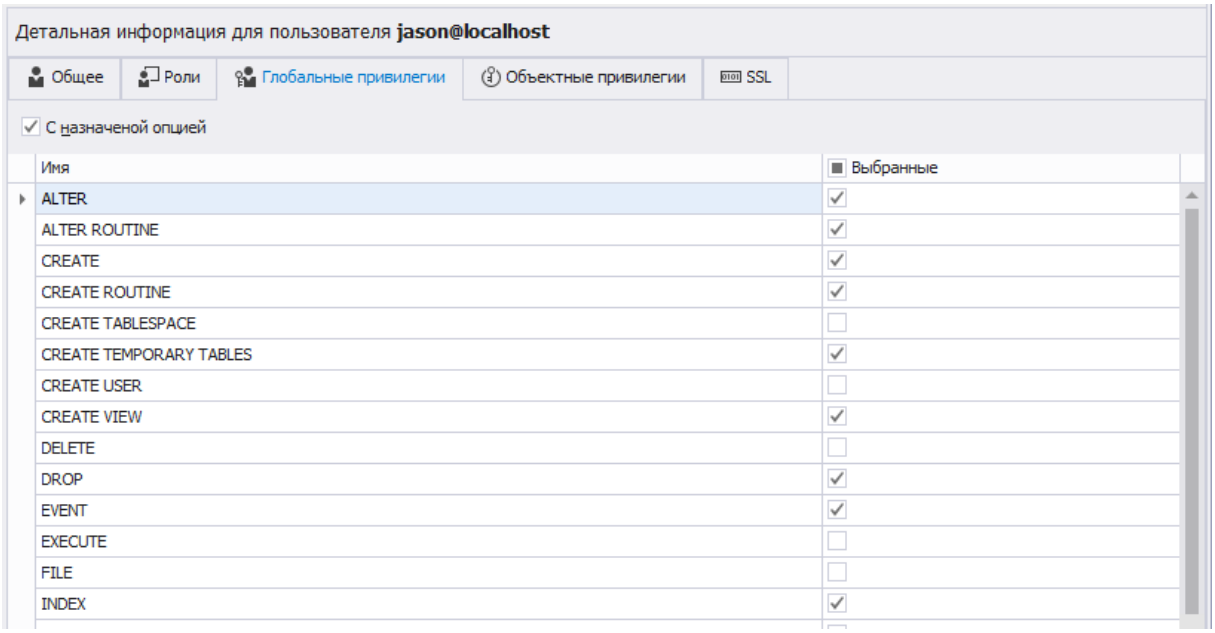

Рис. 8.3. Назначение глобальных привилегий пользователю

Для того чтобы назначить пользователю необходимые права на объект базы данных (таблицу, представление, поле и т. д.), нужно перейти на вкладку **Объектные привилегии** (рис. 8.4).

Таким образом можно задавать разным пользователям соответствующие привилегии в зависимости от типа пользователя программы.

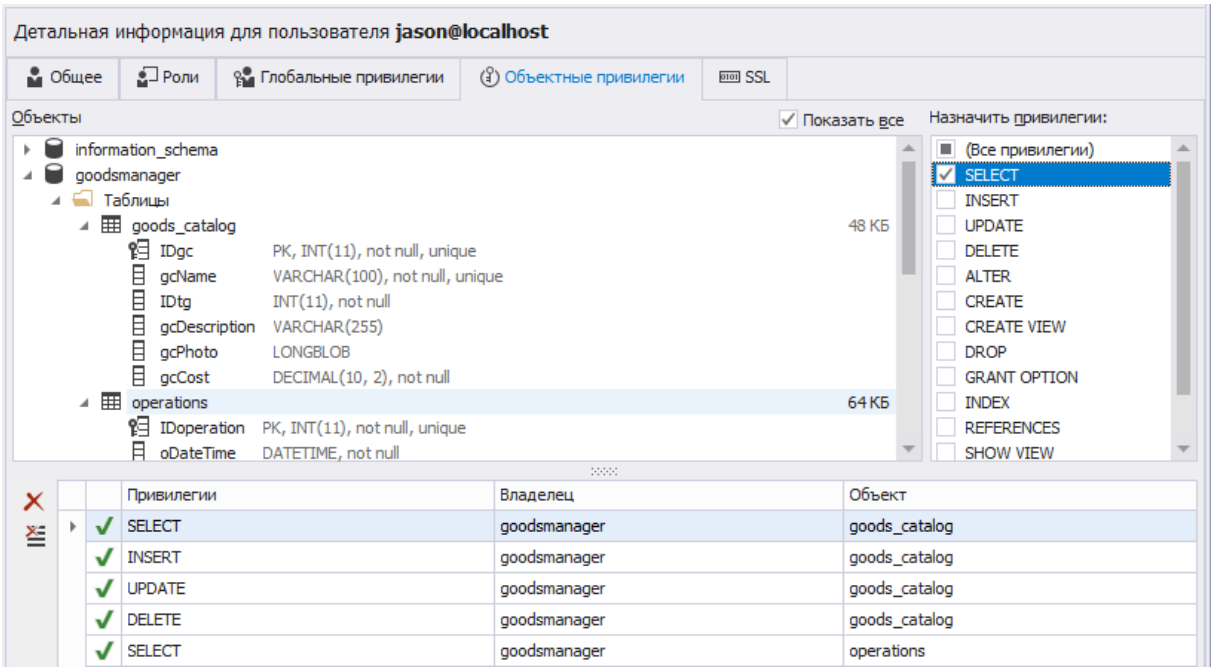

Рис. 8.4. Назначение прав на отдельные объекты БД

Создадим двух новых пользователей, которые могут просматривать все данные. При этом один из них сможет редактировать перемещение товара (изменять данные в таблицах **storage_transfers** и **transfer_items**).

8.2. Определение ODBC драйвера для подключения к MySQL

В большинстве случаев разработчик приложения определяет строку подключения в момент создания приложения, что накладывает ограничения на область применения разработанного продукта. В таком случае приходится иметь необходимую версию ODBC драйвера для функционирования.

С целью обеспечения большей универсальности и упрощения дальнейшей поддержки, включая учет обновления ODBC драйверов, целесообразно при запуске приложения находить имеющийся драйвер и использовать его.

Рассмотрим простой пример по автоматическому определению **ODBC** драйвера **MySQL** с поддержкой **Unicode** имеющегося в операционной системе. Для этого создадим в Delphi новый проект, на форме разместим компоненты **TMemo** и **TButton**, которому зададим имя **B_ListDrivers**

и в свойстве **caption** установим значение **Найти Unicode ODBC драйвера для MySQL** (Рис. 8.5).

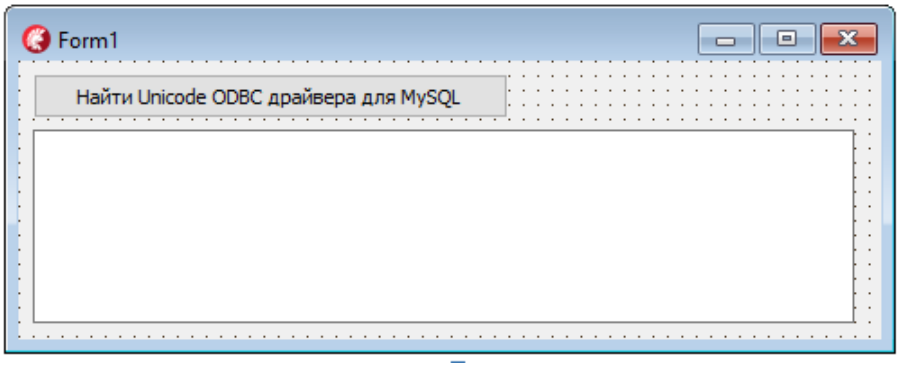

Рис. 8.5. Пример размещения компонентов

Теперь в режиме редактирования кода после блока подключения формы **{$R *.dfm}** напишем процедуру **ListMySQLDriverUnicode**:

```
procedure ListMySQLDriverUnicode(ListMySQLName: TStrings);
var
  reg: TRegistry; // Переменная для доступа к реестру
  ListDriverName: TStringList; // Список ODBC драйверов
  i: integer; // Переменная для цикла
begin
  // Выполняем очистку списка полученного как параметр
  ListMySQLName.Clear;
  // Создаем новый объект списка
  ListDriverName := TStringList.Create;
  // Создаем объект для доступа к реестру
  reg := TRegistry.Create;
  try
    // пытаемся определить корневой параметр локальной машины для
    // поиска установленных ODBC драйверов
    reg.RootKey := HKEY_LOCAL_MACHINE;
    // В зависимости от версии компилируемой программы x64 или x86
    // (64-бит или 32-бита) открывается соответствующая ветка реестра
    if
{$IFDEF CPUX64} // для случая 64-битной программы
      reg.OpenKeyReadOnly('\Software\ODBC\ODBCINST.INI')
{$ELSE} // для случая 32-битной программы
      reg.OpenKeyReadOnly('SOFTWARE\WOW6432Node\ODBC\ODBCINST.INI')
{$ENDIF}
    then
    begin // при успешном открытии ветки реестра с ODBC драйверами
      // заносим их в список ListDriverName
      reg.GetKeyNames(ListDriverName);
      // перебираем в списке драйверов все элементы
      for i := 0 to ListDriverName.Count - 1 do
        // если найдутся совпадения MySQL и Unicode
```

```
        if pos('MySQL', ListDriverName[i]) <> 0 then
          if pos('Unicode', ListDriverName[i]) <> 0 then
            // то записывае информацию в возращаемый список
            ListMySQLName.Add(ListDriverName[i])
    end;
  finally // в завершении закрываем ветку реестра и очищаем объект
    reg.CloseKey;
    FreeAndNil(reg);
  end;
end;
```

Данная процедура позволит получить список драйверов имеющихся в операционной системе. Для её работы необходимо чтобы в блоке **uses** был прописан модуль **registry**.

Выполним проверку работоспособности, для этого на обработчик события OnClick кнопки пропишем код:

```
procedure TForm1.B_ListDriversClick(Sender: TObject);
begin
  ListMySQLDriverUnicode(memo1.Lines);
end;
```

Теперь можно запустить приложение и посмотреть какие драйвера для **MySQL** с поддержкой **Unicode** будут найдены (Рис 8.6).

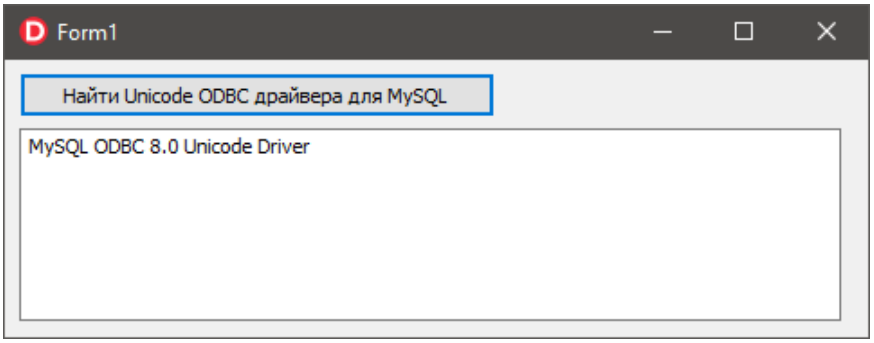

Рис. 8.6. Пример обнаруженных ODBC драйверов для MySQL

Научившись обнаруживать **ODBC** драйвера можно приступить к более сложной задаче – динамическому формированию строки параметров подключения к базе данных.

8.3. Динамическая настройка соединения с MySQL

Строка подключения для MySQL учитывает такие основные параметры как используемый драйвер (**Driver**), адрес сервера **MySQL** (**SERVER**) и порт по которому будет выполняться соединение с сервером (**PORT**) и имя базы данных (**DATABASE**). Эти параметры задают базовую часть. Помимо базовой части для сервера указываются пользовательские

данные логин или имя пользователя (**UID**) и пароль (**PWD**), который указывается только при наличии. В качестве необязательных параметров указывается кодировка для работы с базой данных (**CHARSET**) и параметры колонок (**COLUMN_SIZE_S32**).

Для демонстрации механизма динамической настройки добавим в проект **DataModule** (имя будет **DataModule2**) на который разместим компонент **TADOConnection** (имя **ADOConnection1**, свойству **LoginPrompt** установим значение **false**). Теперь свяжем основную форму (**Form1**) с **DataModule2**.

На форме **Form1** разместим два компонента **TGroupBox,** первый будет содержать компоненты для определения данных сервера **MySQL** и базы данных, а второй для параметров авторизации пользователя. На **GroupBox1** разместим три компонента **TEdit** (имена будут **E_ServerIP**, **E_ServerPort** и **E_Database**), а также компоненты **TLabel** для пояснения. В **GroupBox2** поставим компоненты **TLabel** и два компонента и **TEdit** (Имена **E_Login** и **E_Password**). Также поместим кнопку **Соединение с СУБД MySQL** (имя **B_Connect**). **Form1** примет следующий вид (Рис. 8.7)

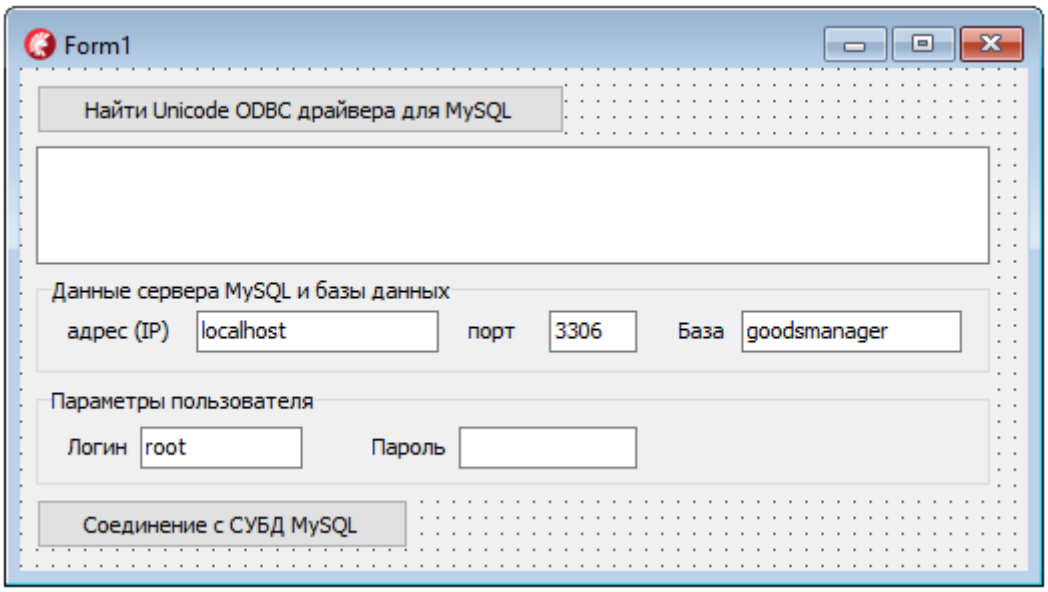

Рис. 8.7. Пример обнаруженных ODBC драйверов для MySQL

Теперь зададим код обработчика события **OnClick** для кнопки **B_Connect:**

```
procedure TForm1.B_ConnectClick(Sender: TObject);
var
  // переменная для хранения собираемой строки подключения
  ConnectionStr: string;
  // список доступных ODBC драйверов
  ListSQLDriverName: TStringList;
  // Имя используемого драйвера
  MySQLDriverUnicode: string;
```

```
begin
  // закрываем соединение если оно открыто
  DataModule2.ADOConnection1.Connected := false;
  // создаем список для определения драйверов
  ListSQLDriverName := TStringList.Create;
  // Заолняем список доступными драйверами
  ListMySQLDriverUnicode(ListSQLDriverName);
  // Если найден драйвер
  if ListSQLDriverName.Count > 0 then
    // то назначаем его переменной MySQLDriverUnicode
    MySQLDriverUnicode := ListSQLDriverName.Strings[0]
  else
  Begin
    // в противном случае выводим сообщение
    ShowMessage('ODBC Connector для MySQL не обнаружен');
    // и прерываем работу
    exit;
  End;
  // При успешном обнаружении драйвера  формируем строке подключения
  // указываем драйвер
  ConnectionStr := 'Driver=' + MySQLDriverUnicode + ';' +
    'SERVER=' + E_ServerIP.Text + ';' + // задаем адрес сервера
    'PORT=' + E_ServerPort.Text + ';' + // указываем порт соединения
  // указываем имя базы данных, с которой соединяться
    'DATABASE=' + E_Database.Text + ';' +
  // задаем параметры пользователя
    'UID=' + e_login.Text + ';'; // указываем имя пользователя
  // проверяем если длина пароля хотя бы один знак
  if Length(e_password.Text) > 0 then
    // добавляем параметр использования пароля в строке подключения
    ConnectionStr := ConnectionStr + 'PWD=' + e_password.Text + ';';
  // устанавливаем дополнительные параметры в виде кодировки
  ConnectionStr := ConnectionStr + 'CHARSET=utf8mb4;' +
      'COLUMN_SIZE_S32=1'; // и настройки колонок
  // Назначаем ADOConnection1 сформированную строку соединения
  DataModule2.ADOConnection1.ConnectionString := ConnectionStr;
  Try // Пытаемся
    // выполнить попытку соединения
    DataModule2.ADOConnection1.Connected := true;
  except
    // в случае ошибки выводим сообщение с номером и описанием ошибки
    with DataModule2.ADOConnection1 do
      ShowMessage('Не удалось подключиться к базе данных. ошибка:' +
        inttostr(Errors.Item[Errors.Count - 1].NativeError) +
        ' ' + Errors.Item[Errors.Count - 1].Description);
    // прерываем выполнение кода
    exit;
  end;
```

```
  // Если соединение было успешным
  if DataModule2.ADOConnection1.Connected then
    // выводим сообщение об этом
    ShowMessage('Соединение с базой ' + E_Database.Text + ' через ' +
      MySQLDriverUnicode + ' выполнено успешно.');
end;
```

Теперь можно проверить как осуществляется соединение с базой данных в СУБД MySQL. При успешном соединении можно увидеть сообщение следующего характера (Рис. 8.8)

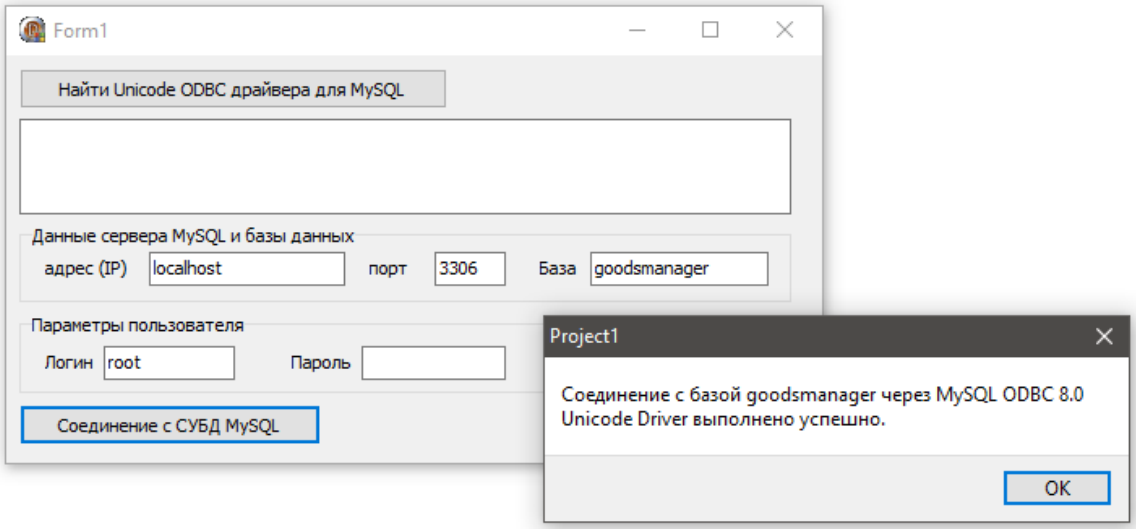

Рис. 8.7. Пример сообщения об успешном соединении с базой данных

Научившись использовать динамическое подключение к базе данных, необходимо сделать настройки для автоматической активации компонентов **TADOTable** и **TADOQuery**.

8.4. Автоматическая активация НД компонентов ADO

Для демонстрации динамической активации **ADO** компонентов на **DataModule2** разместим три компонента **TADOTable** и один компонент **TADOQuery** и четыре компонента **TDataSource**, которые свяжем с компонентами наборов данных. Для всех компонентов наборов данных (**TADOTable** и **TADOQuery**) свойству **Connection** установим значение **ADOConnection1**.

Компонент **TADOQuery** назовем **Q_v_storage_transfers** и в его свойстве SQL пропишем следующий код:

```
SELECT * FROM view_storage_transfers
```

Компоненты **TADOTable** со стандартными именами **ADOTable1, ADOTable2** и **ADOTably3** настроим согласно параметрам, представлен-

ным в таблице 8.1. Настройку необходимо выполнять с активным соединением с базой данных.

Таблица 2.1

Свойства компонентов для вывода описания товаров

Свойство	Значение
ADOTable1	
Name	T_storage_transfers
TableName	storage_transfers
ADOTable2	
Name	T_storage_description_from
TableName	storage_description
ADOTable3	
Name	T_storage_description_to
TableName	storage_description

Как можно заметить созданы две таблицы обращающиеся к одной и той же таблице из базы данных – **storage_description**. Это сделано для того чтобы можно было организовать выпадающие списки для задания склада отправления и назначения у перемещения через **TDBLookupComboBox**. В случае если использовался бы один набор данных, то не получилось бы одновременно выбрать две разные записи.

После настройки **DataModule2** будет выглядеть, как показано на рисунке 8.8.

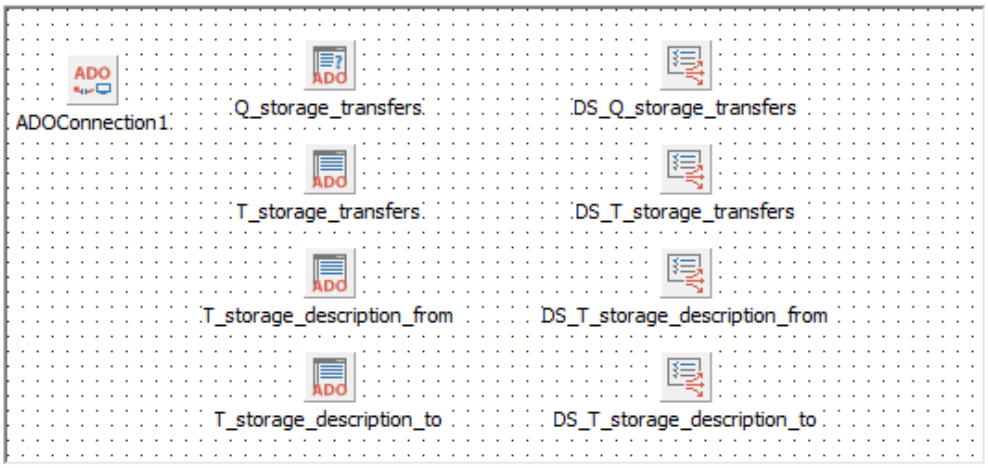

Рис. 8.8. DataModule2 с настроенными компонентами

Соединенные с **TADOConnection** компоненты набора данных фиксируются в свойстве **ADOConnection.DataSets** количество соединенных компонентов хранится в свойстве **ADOConnection.DataSetCount**.

Для активации всех НД компонентов в которых указан **ADOConnection1** воспользуемся событием **AfterConnect**, в обработчик которого запишем следующий код:

```
procedure TDataModule2.ADOConnection1AfterConnect(Sender: TObject);
var
  i: integer;  // переменная для цикла
begin
  // работа с элементами ADOConnection1
  with ADOConnection1 do
  // Перебираем все элементы
  for I := 0 to DataSetCount-1 do
    try // Пытаемся активировать связанны НД
      DataSets[i].Active:=true;
    except
      // В случае ошибки ничего не делаем
    end;
end;
```

Активация компонентов настроена, но для того чтобы проверить работоспособность необходимо прописать обработчики событий у компонентов **Q_v_storage_transfers и T_storage_transfers**.

В событии **AfterScroll** у компонента **Q_v_storage_transfers** будет осуществляться синхронизация отображаемых данных из результата запроса с табличными значениями в компоненте **T_storage_transfers** по полю **IDst.** У компонента **T_storage_transfers** необходимо прописать обработчики событий **AfterPost** и **AfterDelete**. Реализация данных событий будет осуществлена по аналогии с тем, как было сделано в предыдущих лабораторных работах для **Q_v_goods_operations** и **T_operations**.

8.4. Редактирование данных НД с двойной связью

Настроив обработчики событий можно приступить к реализации интерфейсной части по редактированию записей о перемещениях. Для этих целей создадим новую форму (**Form3**). На форме разместим **TDBGrid** и две панели **TPanel**, которым назначим им свойства согласно таблице 8.2.

Таблица 8.2

Свойства компонентов TPanel

Свойство	Значение
Panel1	
Align	alBottom
Name	InsertEditDeletePanel
AlignWithMargins	True
Panel2	
Align	alBottom
Name	PanelToInputValues
Visible	False
AlignWithMargins	True

В **DBGrid1** настроим отображение трех полей **stDateTime sdName_from** и **sdName_to** и свойству **Align** зададим значение **alClient**.

На панель (**InsertEditDeletePanel**) установим три кнопки (компонент **TButton**) которые будут отвечать за активацию режима добавления, изменения и удаления записей (имена **B_Append B_Edit** и **B_Delete** соответственно).

На вторую панель (**PanelToInputValues**) расположим два компонента **TDBLookupComboBox** для выбора склада отправления и склада назначения, два компонента **TDateTimePicker** (один будет отвечать за выбор даты, второй – выбор времени), компоненты **TLabel** для поясняющего текста, а также две кнопки для сохранения и отмены (**B_Post** и **B_Cancel**). Компоненту **DateTimePicker2** ,отвечающему за выбор времени свойству **Kind** установим значение **dtkTime**.

Пример настройки расположения приведен на рисунке 8.9. Компоненты **TDBLookupComboBox** настроим как показано в таблице 8.3.

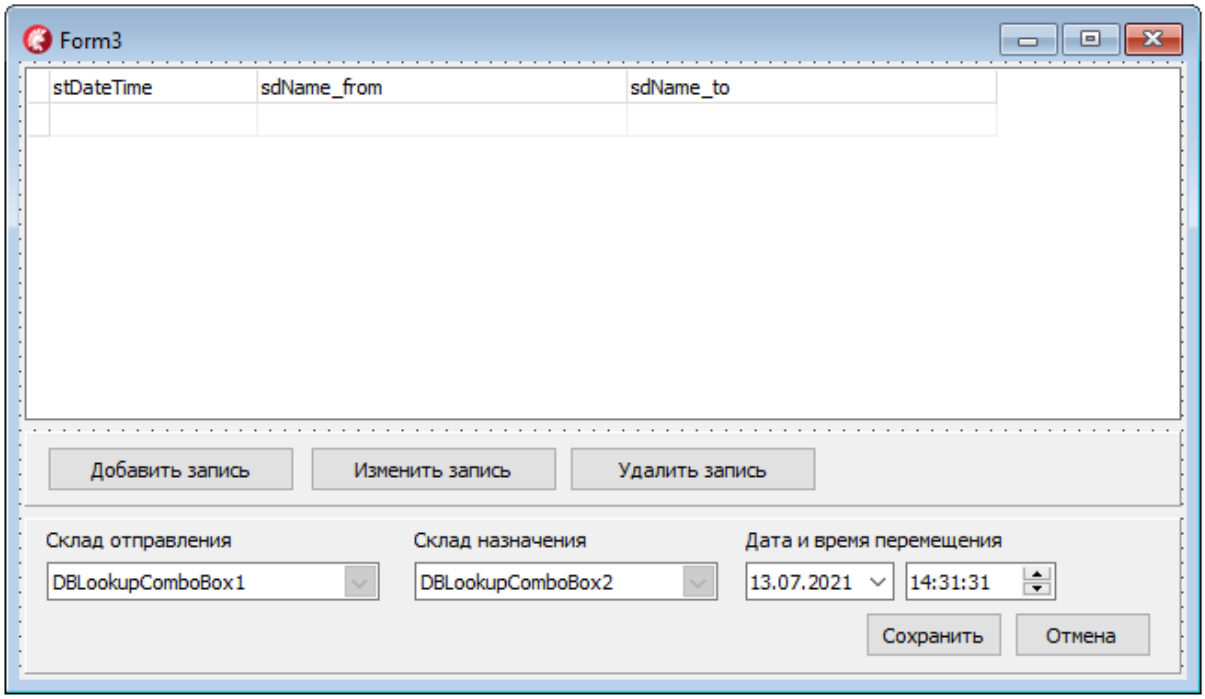

Рис. 8.8. Пример настройки компонентов

Таблица 8.3

Свойства компонентов TDBLookupComboBox

Свойство	Значение
DBLookupComboBox1 (выбор склада отправления)	
DataSource	DataModule2.DS_T_storage_transfers
DataField	IDsd_from
ListSource	DataModule2.DS_T_storage_description_from
ListField	sdName
KeyField	IDsd

Окончание таблицы 8.3

Свойство	Значение
DBLookupComboBox1 (выбор склада назначения)	
DataSource	DataModule2.DS_T_storage_transfers
DataField	IDsd_to
ListSource	DataModule2.DS_T_storage_description_to
ListField	sdName
KeyField	IDsd

Особенность компонентов **TDateTimePicker** заключается в том, сто данные компоненты хранят в себе дату и время и могут выводить их по отдельности и как тип **TDateTime**. Используя эту особенность будем хранить полный формат в компоненте **DateTimePicker1** для этого в событии **OnChange** у **DateTimePicker2** пропишем копирование значения времени из компонента **DateTimePicker2** в **DateTimePicker1:**

```
DateTimePicker1.Time:=DateTimePicker2.Time;
```

Теперь необходимо прописать код на обработчики события **OnClick** у кнопок расположенных на панели **InsertEditDeletePanel**:

```
procedure TForm3.B_AppendClick(Sender: TObject);
begin
  if DataModule2.T_storage_transfers.State = dsBrowse then
  begin
    // Отключаемм возможность изменения курсора НД в DBGrid1
    DBGrid1.Enabled := False;
    // Сделаем невидимой панель с кнопками Вставить, Изменить,
    // Удалить
    InsertEditDeletePanel.Visible := False;
    // Сделаем видимой панель для ввода значения в запись
    PanelToInputValues.Visible := True;
    // Заносим текущее значение даты и времени в DateTimePicker1
    DateTimePicker1.DateTime := Now();
    // Копируем значение времени в DateTimePicker2
    DateTimePicker2.Time:=DateTimePicker1.time;
    // Переведем НД в режим добавления записи
    DataModule2.T_storage_transfers.Append;
  end;
end;

procedure TForm3.B_EditClick(Sender: TObject);
begin
  if DataModule2.T_storage_transfers.State = dsBrowse then
  begin
    DBGrid1.Enabled := False;
    InsertEditDeletePanel.Visible := False;
    PanelToInputValues.Visible := True;
    // Заносим значение даты и времени из поля stDateTime таблицы
```

```
    // storage_transfers в DateTimePicker1
    DateTimePicker1.DateTime :=
      DataModule2.T_storage_transfersstDateTime.Value;
    // Копируем значение времени в DateTimePicker2
    DateTimePicker2.Time:=DateTimePicker1.time;
    // Переводим НД в режим редактирования записи
    DataModule2.T_storage_transfers.Edit;
  end;
end;

procedure TForm3.B_DeleteClick(Sender: TObject);
begin
  if DataModule2.T_storage_transfers.State = dsBrowse then
    if MessageDlg('Подтвердите удаление записи', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes then
      DataModule2.T_storage_transfers.Delete;
end;
```

Код обработчика события **OnClick** у кнопок сохранения и отмены будет следующим:

```
procedure TForm3.P_PostClick(Sender: TObject);
begin
  if DataModule2.T_storage_transfers.State in [dsInsert, dsEdit] then
  begin
    DataModule2.T_storage_transfersstDateTime.Value :=
      DateTimePicker1.DateTime;
    DataModule2.T_storage_transfers.Post;
    // Сделаем видимой панель с кнопками модификации НД
    InsertEditDeletePanel.Visible := True;
    // Сделаем невидимой панель для ввода значения в запись
    PanelToInputValues.Visible := False;
    // Включим возможность изменения курсора НД в компоненте DBGrid1
    DBGrid1.Enabled := True;
  end;
end;

procedure TForm3.B_CancelClick(Sender: TObject);
begin
  if DataModule2.T_storage_transfers.State in [dsInsert, dsEdit] then
  begin
    DataModule2.T_storage_transfers.Cancel;
    // Сделать видимой панель InsertEditDeletePanel
    InsertEditDeletePanel.Visible := True;
    // Сделать невидимой панель для ввода значения в запись
    PanelToInputValues.Visible := False;
    // Включаем возможность изменения курсора НД в компоненте DBGrid1
    DBGrid1.Enabled := True;
  end;
end;
```

Теперь остается настроить вызов формы **Form3**. Для этих целей на главной форме (**Form1**) можно создать элемент меню или же разместить кнопку, а также в обработчике **OnClick** прописать вызов **Form3**:

```
Form3.ShowModal;
```

Демонстрационное приложение готово. Можно выполнить компиляцию (F9) и посмотреть, как будет функционировать приложение, в случае использования различных пользователей.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Из каких разделов состоит блок управления учетной записью в программе **dbForge Studio**?
2. Дайте характеристику глобальных привилегий пользователей.
3. Охарактеризуйте объектные привилегии пользователей.
4. Каким образом задаются ограничения на сессию пользователя?
5. Назовите предустановленные роли для пользователя.
6. Определите назначение глобальных привилегий.
7. Каким образом устанавливаются объектные привилегии?
8. Как возможно определить имеющиеся в системе ODBC драйвера?
9. Какая типовая структура строки подключения к СУБД MySQL в TADOConnection?
10. Нужно ли в строке подключения писать пароль если он пустой?
11. Как можно активировать все наборы данных настроенные через использование TADOConnection?
12. Зачем необходимо использовать дублирующие компоненты НД?
13. Определите назначение компонента **TDateTimePicker**.
14. Какая особенность у компонента **TDateTimePicker** по работе с датой и временем?