ПРОГРАММИРОВАНИЕ АДАПТЕРОВ ДАННЫХ

Объект *DataAdapter* конкретного поставщика данных автоматически обслуживает подключение к БД. Для повышения масштабируемости адаптеры данных держат подключение открытым минимально возможное время. Как только вызывающий процесс получит объект *DataSet*, вызывающий слой полностью отключается от БД и остается с локальной копией удаленных данных. Теперь в нем можно вставлять, удалять или изменять строки различных объектов *DataTable*, но физическая БД не обновляется, пока вызывающий процесс явно не передаст объект *DataSet* адаптеру данных для обновления. По сути, объекты *DataSet* имитируют постоянное подключение клиентов, хотя на самом деле они работают с находящейся в памяти БД. Объект *DataAdapter* отвечает за передачу любого обновления, вставки или удаления в физическую БД.

Обновление связанных таблиц и адаптеры данных

Данные из связанных таблиц базы данных при передаче их в приложение считываются отдельно в набор данных различными адаптерами.

Адаптер данных имеет четыре свойства: ***SelectCommand, InsertCommand, UpdateCommand и DeleteCommand***, но объект ***DataAdapter*** не создает автоматически команды INSERT, UPDATE и DELETE для обновления источника данных в соответствии с изменениями данных в объекте ***DataSet***. Эти команды можно указать различными способами, например, следующими:

- Использовать объект ***CommandBuilder*** для автоматической генерации команд во время выполнения приложения.
- Явно запрограммировать эти команды.
- Использовать хранимые процедуры.

Рассмотрим технологию программирования адаптера данных для выполнения вставки, удаления или изменения строк таблицы.

Для таблиц Emp и Dep базы данных «Отдел кадров» построим интерфейс, в котором пользователь может заполнять таблицу Dep новыми записями, удалять или изменять имеющиеся. Сведения о руководителе отдела, хранящиеся в таблице Emp, можно выбирать из списка, но не вводить новые или удалять имеющиеся записи.

На рисунке 1 приведен фрагмент модели базы данных с выделенным стрелкой отношением, которое формирует значения поля DepartmentHeadId для таблицы Dep.

На рисунке 2 показан пример интерфейса формы в режиме конструирования. Для представления связанных данных о руководителе отдела (поле DepartmentHeadId таблицы Dep) можно использовать элемент управления comboBox с составной привязкой к столбцам EmpId и FIO таблицы Emp.

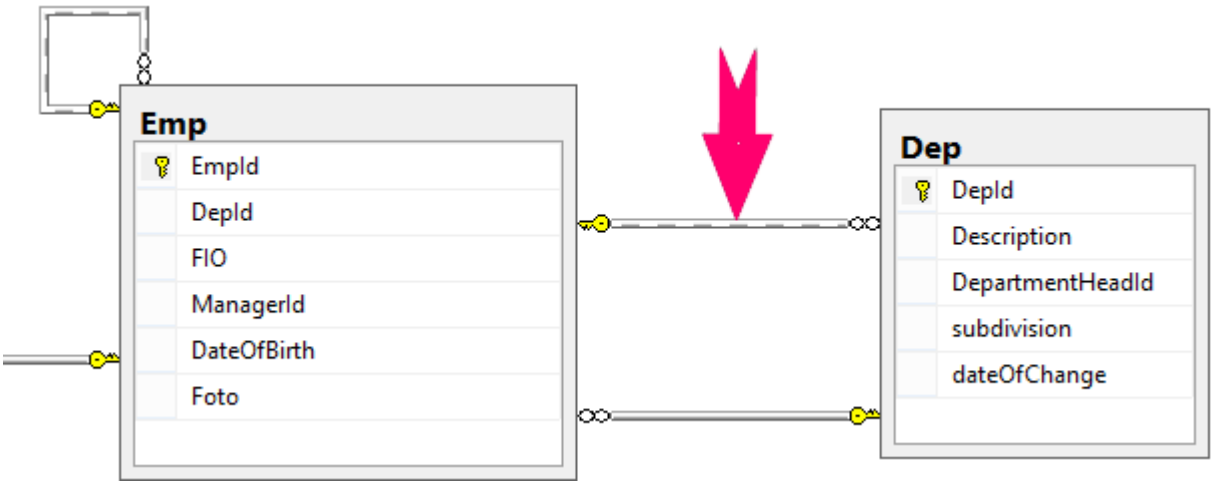

Рисунок 1 - Фрагмент модели

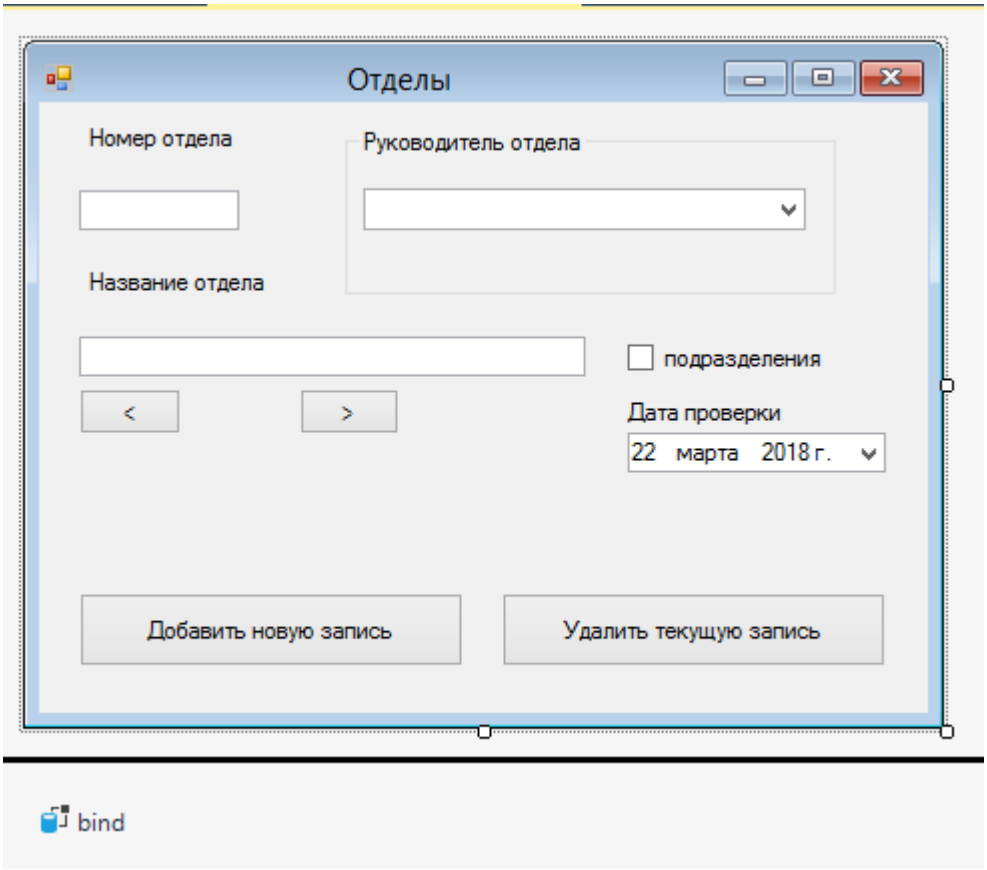

Рисунок 2 - Интерфейс формы в режиме конструктора

Описание свойств адаптеров для выборки данных из таблиц Dep и Emp:

```
private void Form14_Load(object sender, EventArgs e)
{
    //Программируем адаптеры на выборку данных из таблиц базы данных
    daDep.SelectCommand=new SqlCommand("select * from Dep",cnn);
    daEmp.SelectCommand = new SqlCommand("select EmpId,FIO from Emp", cnn);
```

Описание свойств адаптера таблицы Dep для команды вставки новых строк:

```
//Программируем адаптер на вставку новых данных в таблицу Departments
daDep.InsertCommand = new SqlCommand("insert into Dep values (@DepId, @Description,
  @DepartmentHeadId, @sub, @datech)", cnn);
daDep.InsertCommand.Parameters.Add("@DepId",SqlDbType.Int,4,"DepId");
daDep.InsertCommand.Parameters.Add("@Description", SqlDbType.VarChar,50,"Description");
daDep.InsertCommand.Parameters.Add("@DepartmentHeadId", SqlDbType.Int,4, "DepartmentHeadId");
daDep.InsertCommand.Parameters.Add("@sub", SqlDbType.Bit, 1, "subdivision");
daDep.InsertCommand.Parameters.Add("@datech", SqlDbType.DateTime, 8, "dateOfChange");
```

Изменение строк таблицы Dep

```
//Программируем адаптер на обновление данных в таблице Departments
daDep.UpdateCommand = new SqlCommand("update Dep set
  Description=@Description,DepartmentHeadId=@DepartmentHeadId,subdivision=@sub,
  dateOfChange=@datech  where DepId=@DepId", cnn);
daDep.UpdateCommand.Parameters.Add("@DepId", SqlDbType.Int,4, "DepId");
daDep.UpdateCommand.Parameters.Add("@Description", SqlDbType.VarChar, 50, "Description");
daDep.UpdateCommand.Parameters.Add("@DepartmentHeadId", SqlDbType.Int,4, "DepartmentHeadId");
daDep.UpdateCommand.Parameters.Add("@sub", SqlDbType.Bit, 1, "subdivision");
daDep.UpdateCommand.Parameters.Add("@datech", SqlDbType.DateTime, 8, "dateOfChange");
```

Удаление строк из таблицы Dep:

```
//Программируем адаптер на удаление  данных из таблицы Departments
daDep.DeleteCommand = new SqlCommand("delete from Dep where DepId=@DepId", cnn);
daDep.DeleteCommand.Parameters.Add("@DepId", SqlDbType.Int,4, "DepId");
```

После получения выбранных данных из таблиц можно организовать интерфейс формы:

```
//Заполняем хранилище данных
daDep.Fill(ds, "Dep");
daEmp.Fill(ds, "Emp");
//Создаем объект для синхронизации связанных данных
bind.DataSource = ds.Tables["Dep"];
```

```
//Связываем поля формы с соответствующими данными
textBox1.DataBindings.Add("text", bind, "DepId");
textBox2.DataBindings.Add("text", bind, "Description");
comboBox1.DataSource = ds.Tables["Emp"];
comboBox1.ValueMember = "EmpId";
comboBox1.DisplayMember = "FIO";
comboBox1.DataBindings.Add("SelectedValue", bind, "DepartmentHeadID");
checkBox1.DataBindings.Add("Checked", bind, "subdivision");
dateTimePicker1.DataBindings.Add("Value", bind, "dateOfChange");
```

Просмотр и удаление текущей записи удобно выполнять свойствами и методами объекта BindingSource:

```
//Просмотр записей <
private void button1_Click(object sender, EventArgs e)
{
    bind.Position -= 1;
 }
//Просмотр записей >
private void button2_Click(object sender, EventArgs e)
{
    bind.Position += 1;
 }
```

```
//Удалить текущую запись
private void button4_Click(object sender, EventArgs e)
{
    bind.RemoveCurrent();
    if (ds.Tables["Dep"].GetChanges(DataRowState.Deleted) != null) daDep.Update(ds.Tables["Dep"]);

}
```

Добавление новой записи:

```
//Добавить новую запись
private void button3_Click(object sender, EventArgs e)
 {
     DataRow row = ds.Tables["Dep"].NewRow();
     row[0] = Convert.ToInt32(ds.Tables["Dep"].Rows.Count+1);
     row[1] = "Введите название отдела";
     row[2] = Convert.ToInt32(comboBox1.SelectedValue);
     row[3] = checkBox1.Checked;
     row[4] = dateTimePicker1.Value;
     ds.Tables["Dep"].Rows.Add(row);
     if (ds.Tables["Dep"].GetChanges(DataRowState.Added) != null) daDep.Update(ds.Tables["Dep"]);
     bind.MoveLast();
}
```

Сохранение всех изменений, добавлений, удалений можно выполнить при закрытии формы:

```
//При закрытии формы сохранить все изменения в БД
        private void Form14_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (ds.Tables["Dep"].GetChanges() != null) daDep.Update(ds.Tables["Dep"]);
        }
```

Протестируйте созданную форму: вносите новые записи, просматривайте, удаляйте и изменяйте имеющиеся.

Задание для самостоятельной работы

Разработайте форму для связанных данных из нескольких таблиц в вашем приложении. Обновление источника данных реализуйте программированием адаптеров данных.