МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Системы автоматизированного проектирования»

БАЗЫ ЗНАНИЙ И ПОДДЕРЖКА ПРИНЯТИЯ РЕШЕНИЙ В САПР

Лабораторный практикум
для студентов специальности 1-40 01 02-01 «Информационные системы и технологии в проектировании и производстве»

В 3 частях

Ч а с т ь 2

М и н с к
Б Н Т У
2 0 1 3

УДК 004.65+658.512.22-027.44(076.5)
ББК 32.81я7
Б17

Составители:
В. А. Кочуров, Ю. О. Герман

Рецензенты:
доцент кафедры ПОВТ БНТУ, канд. техн. наук *В. В. Бугай*;
доцент кафедры ИТАС БГУИР, канд. техн. наук *О. В. Герман*

Базы знаний и поддержка принятия решений в САПР : лабора-
Б17 торный практикум для студентов специальности 1-40 01 02-01 «Информационные системы и технологии в проектировании и производстве» : в 3 ч. / сост.: В. А. Кочуров, Ю. О. Герман. – Минск : БНТУ, 2012– . – Ч. 2. – 2013. – 39 с.
ISBN 978-985-550-226-6 (Ч. 2).

Материал предназначен для использования в качестве методических указаний при изучении дисциплины «Базы знаний и поддержка принятия решений в САПР». Издание содержит описание лабораторных работ, выполнение которых позволит реализовать процесс подключения и использования баз данных на платформе Microsoft Visual Studio 2008–2010 и поможет:

- создать и настроить набор данных в приложениях на основе объектов баз данных;
- создать элементы управления с привязкой к данным;
- создать и использовать хранимые процедуры;
- создать запросы LINQ и т.д.

Первая часть вышла в БНТУ в 2012 г.

УДК 004.65+658.512.22-027.44(076.5)
ББК 32.81я7

ISBN 978-985-550-226-6 (Ч. 2)
ISBN 978-985-525-908-5

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 5

СОХРАНЕНИЕ ДАННЫХ В БАЗЕ ДАННЫХ (НЕСКОЛЬКИХ ТАБЛИЦ)

В этой работе рассматриваются следующие задачи:

- создание нового проекта Приложение Windows;
- создание и настройка источника данных в приложении с помощью мастера настройки источника данных;
- настройка элементов управления из элементов окна Источники данных;
- создание элементов управления с привязкой к данным при помощи перетаскивания элементов из окна Источники данных на форму;
- изменение пары записей в каждой таблице в наборе данных;
- изменение кода отправки обновленных данных в наборе данных обратно в базу данных.

Для выполнения этого пошагового руководства потребуется доступ к образцу базы данных «Борей».

Создание приложения Windows см. Лабораторную работу №1 (часть 1).

Создание источника данных см. Лабораторную работу №4 (часть 1).

Установка создаваемых элементов управления

В данном пошаговом руководстве данные в таблице *Suppliers* будут в макете ***Сведения***, где данные отображаются в отдельных элементах управления. Данные из таблицы *Goods* будут в макете ***Сетка***, отображаемом в элементе управления DataGridView.

Для установки типа переноса для элементов в окне "Источники данных"

1. Разверните узел *Suppliers* в окне ***Источники данных***.

2. Замените элемент управления таблицы Suppliers на отдельные элементы управления, выбрав ***Сведения*** из списка элементов управления на узле Suppliers.

Задание элементов управления, которые должны быть созданы для таблиц или объектов данных

Прежде чем перетащить элементы, представляющие таблицы или объекты данных из окна **Источники данных**, можно выбрать, как следует отображать данные: в одном элементе управления или каждый столбец или свойство в отдельном элементе управления.

В этом контексте термин объект означает пользовательский бизнес-объект или сущность (в модели EDM) или объект, возвращенный службой.

Чтобы задать создание элементов управления для таблиц или объектов данных:

1. Убедитесь, что открыт конструктор WPF или конструктор Windows Forms.

2. В окне **Источники данных** выберите элемент, представляющий таблицу или объект данных, который следует задать.

3. Щелкните раскрывающееся меню для этого элемента, затем щелкните один из следующих элементов в меню:

- Чтобы отобразить каждое поле данных в отдельном элементе управления, щелкните **Таблица**. При перетаскивании элемента данных в конструктор для каждого столбца или свойства родительской таблицы или объекта данных будет создан отдельный элемент управления, привязанный к данным, в месте с метками для каждого элемента управления.
- Чтобы отобразить все данные в одном элементе управления, выберите другой элемент управления в списке, например, **Сетка данных** или **Список** в приложении WPF или DataGridView в приложении Windows Forms.

Список доступных элементов управления зависит от открытого конструктора, версии .NET Framework, используемой для проекта, и добавления пользовательских элементов управления, поддерживающих привязку к данным, на **панель инструментов**. Если создава-

емый элемент управления присутствует в списке доступных элементов управления, можно добавить элемент управления в список.

Сведения о создании пользовательского элемента управления Windows Forms, который может быть добавлен в список элементов управления таблицы или объекта данных, расположенный в окне **Источники данных**, см. в разделе Создание пользовательского элемента управления Windows Forms со сложной привязкой данных.

Задание элементов управления, которые должны быть созданы для столбцов или свойств данных

Прежде, чем перетащить элемент, представляющий столбец или свойство объекта, из окна **Источники данных** в конструктор, можно задать создание такого элемента управления.

Чтобы задать создание элементов управления для столбцов и свойств

1. Убедитесь, что открыт конструктор WPF или конструктор Windows Forms.
2. В окне **Источники данных** разверните нужную таблицу или объект для отображения его свойств или столбцов.
3. Выберите каждый столбец или свойство, для которых следует создать элементы управления.
4. Щелкните раскрывающееся меню для этого столбца или свойства, затем выберите элемент управления, который следует создать, во время перетаскивания элемента в конструктор.

Сведения о создании пользовательского элемента управления Windows Forms, который может быть добавлен в список элементов управления столбцов или свойств данных, расположенный в окне **Источники данных**, см. в разделе Пошаговое руководство. Создание пользовательского элемента управления Windows Forms с простой привязкой данных.

В раскрывающемся списке выберите пункт **Нет**, если не следует создавать элемент управления для столбца или свойства. Это полезно, если следует перетащить в конструктор родительскую таблицу

или объект, но нет необходимости добавлять определенный столбец или свойство.

Создание формы с привязкой к данным

Можно создавать элементы управления с привязкой к данным, перетаскивая элементы из окна Источники данных на форму.

Чтобы создать элементы управления с привязкой к данным на форме:

1. Перетащите главный узел Suppliers из окна Источники данных на Form1.

Элементы управления с привязкой к данным с подписями описания появятся на форме вместе с панелью инструментов (BindingNavigator) для управления записями.BoreiDataSet, SuppliersTableAdapter, BindingSource и BindingNavigator появляются в области компонентов.

2. Перетащите связанный узел Goods из окна Источники данных на форму Form1.

3. На форме появится элемент управления DataGridView и панель инструментов (BindingNavigator) для перемещения по записям.GoodsTableAdapter и BindingSource отображаются в области компонентов.

Добавление кода для обновления базы данных

Можно обновить базу данных, вызывая методы Update адаптеров таблиц TableAdapter Suppliers и Goods. По умолчанию обработчик событий для кнопки Сохранить панели BindingNavigator добавляется в код формы для отправки обновлений в базу данных. Эта процедура изменяет код для отправки обновлений в нужном порядке, чтобы исключить вероятность возникновения ошибок целостности данных. Кроме того, код реализует обработку ошибок путем заключения вызова обновления в блок try-catch. Можно изменить код в соответствии с требованиями приложения.

Чтобы добавить логику обновления в приложение:

1. Дважды щелкните кнопку Сохранить на BindingNavigator, чтобы открыть редактор кода для обработчика событий bindingNavigatorSaveItem_Click.

2. Замените код в обработчике событий на вызов методов Update связанных адаптеров таблиц TableAdapter. Следующий код сначала создает три временных таблицы данных для хранения обновленной информации для каждого DataRowState (Deleted, Added и Modified). Затем обновления выполняются в нужном порядке. Код должен выглядеть следующим образом:

```
this.Validate();
this.suppliersBindingSource.EndEdit();
BoreiDataSet.GoodsDataTable deletedGoods = (Borei-
DataSet.GoodsDataTable)
            BoreiDataSet.Goods.GetChanges(DataRowState.Deleted);

BoreiDataSet.GoodsDataTable newGoods = (Borei-
DataSet.GoodsDataTable)
BoreiDataSet.Goods.GetChanges(DataRowState.Added);

BoreiDataSet.GoodsDataTable modifiedGoods = (Borei-
DataSet.GoodsDataTable)
            BoreiDataSet.Goods.GetChanges(DataRowState.Modified);
try
 {
  if (deletedGoods != null)
   {
    goodsTableAdapter.Update(deletedGoods);
   }

  suppliersTableAdapter.Update(BoreiDataSet.Suppliers);
  if (newGoods != null)
   {
    goodsTableAdapter.Update(newGoods);
   }
```

```
        if (modifiedGoods != null)
        {
          goodsTableAdapter.Update(modifiedGoods);
        }
       BoreiDataSet.AcceptChanges();
        }

      catch (System.Exception ex)
        {
           MessageBox.Show("Update failed");
        }

      finally
        {
         if (deletedGoods != null)
      {
       deletedGoods.Dispose();
      }
     if (newGoods != null)
      {
       newGoods.Dispose();
      }
     if (modifiedGoods != null)
      {
       modifiedGoods.Dispose();
      }
        }
```

Тестирование приложения:

1. Нажмите клавишу F5.
2. Внесите изменения в данные одной или нескольких записей в каждой таблице.
3. Нажмите кнопку Сохранить.
4. Проверьте значения в базе данных, чтобы убедиться, что изменения были сохранены.

ЛАБОРАТОРНАЯ РАБОТА № 6

СОХРАНЕНИЕ ДАННЫХ С ПОМОЩЬЮ МЕТОДОВ DBDIRECT АДАПТЕРА ТАБЛИЦЫ

В этой работе рассматриваются следующие задачи:

- создание нового проекта Приложение Windows;
- создание и настройка источника данных в приложении;
- добавление элементов управления на форму для отображения данных;
- добавление кода для вставки, обновления и удаления записей;
- проверка работоспособности приложения.

Создание нового проекта приложения Windows

1. В Visual Studio из меню **Файл** выбрали новый **Проект**.
2. Назвали проект.
3. Выбрали **Приложение Windows** и нажали кнопку OK. Проект создается и добавляется в **Обозреватель решений**.

Создание источника данных из вашей базы данных

На этом шаге с помощью **Мастера настройки источника данных** создается источник данных на основе таблицы Suppliers базы данных "Борей".

1. В меню **Данные** выбрали команду **Показать источники данных**.
2. Чтобы запустить **Мастер настройки источника данных**, выбрали элемент **Добавить новый источник данных** в окне **Источники данных**.
3. На странице **Выбор типа источника данных** выбрали элемент **База данных** и нажали **Далее**.
4. Выбрали **Новое подключение** для открытия диалогового окна **Добавить/изменить подключение**.
5. Щелкнули **Далее** на странице **Сохранить строку подключения в файле конфигурации приложения**.

6. Развернули узел **Таблицы** на странице **Выбор объектов базы данных**.

7. Выбрали таблицу *Suppliers* и нажали кнопку **Готово**.

DataSet добавляется к проекту, и таблица Suppliers отображается в окне **Источники данных**.

Добавление элементов управления на форму для отображения данных

Создайте элементы управления с привязкой к данным путем перетаскивания элементов из окна **Источники данных** на форму. Для этого перетащите главный узел Suppliers из окна **Источники данных** на форму. На форме появился элемент управления DataGridView и панель инструментов (BindingNavigator) для перемещения по записям. В области компонента отобразились BoreiDataSet, SuppliersTableAdapter, BindingSource и BindingNavigator.

Добавление кнопок, которые будут вызывать отдельные методы DbDirect объекта TableAdapter

1. Перетащите три элемента управления Button из **Панели элементов** на Form1 (под RegionDataGridView).

2. Задайте следующие свойства **Имя** и **Текст** для каждой кнопки (таблица 1).

Таблица 1 – Задание свойств кнопкам

Имя	Text
InsertButton	Вставить
UpdateButton	Обновить
DeleteButton	Удалить

Добавление кода для вставки новых записей в базу данных

1. Дважды щелкните InsertButton, чтобы создать обработчик событий для события Click и открыть форму в редакторе кода.

2. Замените обработчик событий InsertButton_Click следующим кодом:

```
private void RefreshDataset()
      {
        this.suppliersTableAdapter.Fill(this.boreiDataSet.Suppliers);
      }
private void InsertButton_Click(object sender, EventArgs e)
      {
        Int32 newCodeSuppliers = 20;
        String newTitle = "Unicomp";
        String newAddressTo="Вероника Кудрявцева";
        String newPost = "Менеджер по закупкам";
        String newAddress= "ул. Большая Садовая, 12";
        String newCity = "Москва";
        String newIIndex = "123456";
        String newCountry = "Россия";
        String newTelephone = "(095)3252222";
        String newFax = "(095)3252222";
       try
        {
          suppliersTableAdapter.Insert(newCodeSuppliers, newTitle,
newAddressTo, newPost, newAddress, newCity,
            newIIndex, newCountry, newTelephone, newFax);
        }
        catch (Exception ex)
        {
          MessageBox.Show("Insert Failed");
        }
        RefreshDataset();
      }
```

Добавление кода для обновления записей в базе данных

1. Дважды щелкните UpdateButton, чтобы создать обработчик событий для события Click и открыть форму в редакторе кода.

2. Замените обработчик событий UpdateButton_Click следующим кодом:

```
private void UpdateButton_Click(object sender, EventArgs e)
{  try
      {
        suppliersTableAdapter.Update(20,          "Unicomputer",
"Василий Шпак", "Менеджер по продажам",
        "ул. Большая Садовая, 12", "Москва", "123456",
"Россия", "(095)3252222", "(095)3252222",
        20, "Unicomp", "Вероника Кудрявцева", "Менеджер
по закупкам",
        "ул. Большая Садовая, 12", "Москва", "123456",
"Россия", "(095)3252222", "(095)3252222");
      }
      catch (Exception ex)
      {
        MessageBox.Show("Update Failed");
      }
      RefreshDataset();
}
```

Добавление кода для удаления записей из базы данных

1. Дважды щелкните DeleteButton, чтобы создать обработчик событий для события Click и открыть форму в редакторе кода.

2. Замените обработчик событий DeleteButton_Click следующим кодом:

```
private void DeleteButton_Click(object sender, EventArgs e)
    {
      try
      {
        suppliersTableAdapter.Delete(20, "Unicomp", "Вероника
Кудрявцева", "Менеджер по закупкам",
        "ул. Большая Садовая, 12", "Москва", "123456",
"Россия", "(095)3252222", "(095)3252222");
```

```
            suppliersTableAdapter.Delete(20, "Unicomputer",
"Василий Шпак", "Менеджер по продажам",
                "ул. Большая Садовая, 12", "Москва", "123456",
"Россия", "(095)3252222", "(095)3252222");
            }
            catch (Exception ex)
            {
               MessageBox.Show("Delete Failed");
            }
            RefreshDataset();
        }
```

Запуск приложения

- Нажмите клавишу F5 для запуска приложения.
- Нажмите кнопку **Вставить** и убедитесь, что новая запись отображается в сетке.
- Нажмите кнопку **Обновить** и убедитесь, что запись обновляется в сетке.
- Нажмите кнопку **Удалить** и убедитесь, что запись удаляется из сетки.

ЛАБОРАТОРНАЯ РАБОТА № 7

СОЗДАНИЕ И ВЫПОЛНЕНИЕ ИНСТРУКЦИЙ SQL, ВОЗВРАЩАЮЩИХ СТРОКИ

В данной работе показано создание запроса адаптера таблицы с помощью Мастера настройки запроса адаптера таблицы, а также приводятся сведения об объявлении экземпляра адаптера таблицы и выполнении запроса.

Чтобы создать возвращающую строки инструкцию SQL с помощью адаптера таблицы

1. Откройте набор данных в ***Конструкторе наборов данных***. Дополнительные сведения см. Лабораторная работа № 2 (часть1).

2. Создайте адаптер таблиц, если не сделали этого ранее.

3. Если запрос на адаптер таблицы, использующий возвращающую строки инструкцию SQL, уже существует, перейдите к следующей процедуре: "Чтобы объявить экземпляр Адаптера таблицы и выполнить запрос". В противном случае продолжите с шага 4, чтобы создать новый запрос, возвращающий строки.

4. Щелкните требуемый адаптер таблицы правой кнопкой мыши и используйте контекстное меню для добавления запроса.

Откроется Мастер настройки запроса адаптера таблиц.

5. Оставьте значение по умолчанию для элемента ***Использовать SQL инструкции*** и нажмите кнопку ***Далее***.

6. Оставьте значение по умолчанию для элемента ***SELECT, возвращающий строки*** и нажмите кнопку ***Далее***.

7. Введите инструкцию SQL или воспользуйтесь ***Построителем запросов*** для упрощения создания, затем нажмите кнопку ***Далее***.

8. Введите имя для запроса.

9. Завершите работу мастера; запрос добавляется к адаптеру таблицы.

10. Скомпилируйте проект.

Чтобы объявить экземпляр адаптера таблиц и выполнить запрос:

1. Объявите экземпляр ***адаптера таблиц***, содержащий запрос, который требуется выполнить.

- Для создания экземпляра с помощью *средств времени проектирования* перетащите нужный адаптер таблиц из ***Панели элементов***. (Компонент в проекте теперь отображается в Панели элементов под заголовком, совпадающим с именем проекта). Если адаптер таблиц не отображается в Панели элементов, может потребоваться скомпилировать проект.

либо

o Чтобы создать экземпляр в коде, подставьте в следующий код имена *DataSet* и адаптера таблиц.

2. Вызовите запрос, как вызвали бы любой другой метод в коде. Запрос представляет собой метод адаптера таблиц.

3. Полный код объявления экземпляра адаптера таблицы и выполнения запроса должен выглядеть примерно следующим образом:

Язык C#

```
try
        {
          boreiDataSetTableAdapters.SuppliersTableAdapter
tableAdapter =
              new
boreiDataSetTableAdapters.SuppliersTableAdapter();
          tableAdapter.FillBy(boreiDataSet.Suppliers);
        }
        catch (System.Exception ex)
        {
          System.Windows.Forms.MessageBox.Show(ex.Message);}
```

Выполнение возвращающих строки инструкций SQL с помощью командного объекта

В следующем примере демонстрируется способ создания команды и выполнения инструкции SQL, возвращающей строки. Сведения об установке и получении значений параметров для команды см. в разделе Установка и получение параметров командных объектов.

Этот пример использует объект SqlCommand и требует:

- Ссылки на пространства имен System, System.Data, System.Data.SqlClient и System.Xml.
- Подключение к данным с именем sqlConnection1.
- Таблицу с именем Suppliers в источнике данных, к которому подключается sqlConnection1. (В противном случае необходима допустимая инструкция SQL для источника данных).

Выполнение возвращающей строки инструкции SQL программными средствами с помощью командного объекта

- Добавьте следующий код к методу, из которого он должен выполняться. Возвращайте строки, вызвав метод команды ExecuteReader (например, ExecuteReader). Данные возвращаются в SqlDataReader.

Язык C#

```
SqlConnection sqlConnection1 = new SqlConnection("Data Source=.\\SQLEXPRESS;AttachDbFilename=G:\\базы данных\\Borei\\borei.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True");
            SqlCommand cmd = new SqlCommand();
            SqlDataReader reader;
            cmd.CommandText = "SELECT * FROM Suppliers";
            cmd.CommandType = CommandType.Text;
            cmd.Connection = sqlConnection1;

            sqlConnection1.Open();
            reader = cmd.ExecuteReader();
            int i = 0;
            while (reader.Read())
            {
              dataGridView1.Rows.Add();
              dataGridView1[0, i].Value =
        reader["CodeSuppliers"].ToString();
              dataGridView1[1, i].Value = reader["Title"].ToString();
              dataGridView1[2,i].Value= reader["AddressTo"].ToString();
                dataGridView1[3, i].Value = reader["Post"].ToString();
                dataGridView1[4, i].Value = reader["City"].ToString();
                dataGridView1[5, i].Value = reader["Country"].ToString();
                i++;
            }
            reader.Close();
            sqlConnection1.Close();
```

ЛАБОРАТОРНАЯ РАБОТА № 8

ВЫПОЛНЕНИЕ ХРАНИМОЙ ПРОЦЕДУРЫ, ВОЗВРАЩАЮЩЕЙ ОДИНОЧНОЕ ЗНАЧЕНИЕ

Синтаксис хранимых процедур

Создать хранимые процедуры можно при помощи выражения CREATE PROCEDURE языка Transact-SQL. Перед созданием хранимой процедуры необходимо убедиться, что:

- выражения CREATE PROCEDURE не могут быть совмещены с другими выражениями языка SQL в одной партии [batch] (партия ограничена операторами GO);
- нужно иметь соответствующие права в базе данных для создания хранимых процедур (CREATE PROCEDURE и ALTER);
- имена хранимых процедур должны соответствовать правилам именования идентификаторов;
- хранимую процедуру можно создать только в текущей базе данных.

При создании хранимой процедуры вы должны определить:

- входные параметры и возвращаемые значения;
- выражения языка SQL, которые выполняют действия над данными в базе, включая вызовы других процедур;
- статусное значение, которое будет возвращено вызывающей процедуре или партии [batch], чтобы обозначить успех или неудачу выполнения (и причину неудачи);
- различные обработчики ошибок, возникающие в процессе выполнения процедуры.

Синтаксис:

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
       [ VARYING ] [ = default ] [ [ OUT [ PUT ]
    ] [ ,...n ]
```

```
[ WITH <procedure_option> [ ,...n ]
[ FOR REPLICATION ]
AS { <sql_statement> [;][ ...n ] | <method_specifier> }
[;]
<procedure_option> ::=
   [ ENCRYPTION ]
   [ RECOMPILE ]
   [ EXECUTE_AS_Clause ]

<sql_statement> ::=
{ [ BEGIN ] statements [ END ] }

<method_specifier> ::=
EXTERNAL NAME assembly_name.class_name.method_name
```

Просмотр имеющихся в базе данных хранимых процедур

Чтобы просмотреть имеющиеся в базе данных хранимые процедуры (пользовательские или системные), нужно в *Management Studio* в окне *Object Explorer* отыскать и развернуть ветку дерева BOREI/Programmability/Stored Procedures (рис. 9).

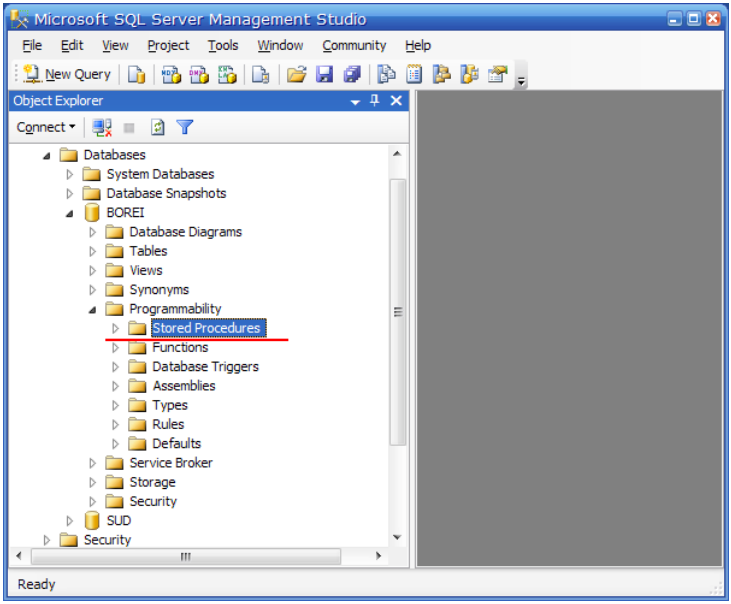

Рис. 9. Просмотр имеющихся в БД хранимых процедур

Создание простой SELECT-процедуры

1. Выполним следующий код, чтобы создать хранимую процедуру с именем get_client_orders. Существует полезное соглашение начинать имена переменных префиксом “v_” для улучшения читабельности кода, однако это соглашение не является обязательным. Следующий код представляет собой единичный SQL оператор, введём и выполним его как единое целое:

```
CREATE PROCEDURE get_client_orders
      @v_codeclient SMALLINT
AS
SELECT Count(CodeClient)
FROM orders
WHERE CodeClient = @v_codeclient
GO
```

Эта процедура принимает в качестве входного параметра код клиента (v_codeclient) и возвращает количество заказов (Codeclient) этого клиента.

2. Чтобы увидеть, как работает процедура, введём следующий запрос:

```
DECLARE @v_codeclient smallint
EXEC dbo.get_client_orders 5
```

Для выполнения сохраненной процедуры, возвращающей одиночное значение, с помощью адаптера таблиц:

1. Откройте набор данных в ***Конструкторе наборов данных***.
2. Создайте адаптер таблиц, если не сделали этого ранее.
3. Если для адаптера таблиц уже имеется запрос, использующий сохраненную процедуру, которая возвращает одиночное значение, переходите к процедуре "Объявление экземпляра адаптера таблицы и выполнение запроса". В противном случае необходимо продолжить выполнение шага 4, чтобы создать новый запрос, возвращающий одиночное значение.
4. Щелкните требуемый адаптер таблиц правой кнопкой мыши и используйте контекстное меню для добавления запроса.

Откроется Мастер настройки запроса адаптера таблиц.

5. Выберите ***Использовать существующую сохраненную процедуру*** и нажмите кнопку ***Далее***.
6. Выберите сохраненную процедуру из раскрывающегося списка, и затем нажмите ***Далее***.
7. Выберите параметр ***Одиночное значение*** и нажмите кнопку ***Далее***.
8. Введите имя для запроса.
9. Нажмите кнопку ***Далее*** или ***Готово***, чтобы завершить работу мастера. Запрос добавляется в адаптер таблиц.
10. Скомпилируйте проект.

Выполнение хранимой процедуры, возвращающей одиночное значение, с помощью DataCommand

Добавьте следующий код к методу, из которого он должен выполняться. Одиночное значение возвращается путем вызова метода *ExecuteScalar* команды. Данные возвращаются в object. Для этого на форму ставим кнопку, куда и записываем нижеприведенный код, а также textbox и label:

```
if (textBox1.Text == "")
  textBox1.Text = "0";
        SqlConnection SqlConnect = new SqlConnection("Data
Source=.\\SQLEXPRESS;AttachDbFilename=E:\\дисциплины\\borei.m
df;Integrated Security=True;Connect Timeout=30;User Instance=True");
        SqlCommand cmd = new SqlCommand();
        Object ReturnValue;
        cmd.CommandText = "EXEC dbo.get_client_orders " + text-
Box1.Text;
        cmd.CommandType = CommandType.Text;
        cmd.Connection = SqlConnect;
        SqlConnect.Open();
        ReturnValue = cmd.ExecuteScalar();
        SqlConnect.Close();
        Label1.Text = ReturnValue.ToString();
```

ЛАБОРАТОРНАЯ РАБОТА № 9

СОЗДАНИЕ ЗАПРОСОВ В C# (LINQ)

В этой работе показано использование возможностей C#, которые служат для написания выражений запросов LINQ. После выполнения данной работы можно переходить к примерам и документации по конкретным поставщикам LINQ, которые вас заинтересовали, например LINQ to SQL, LINQ в набор данных или LINQ to XML.

Для выполнения работы требуется Visual Studio 2008.

Создание проекта C#

1. Запустите Visual Studio.
2. В меню ***Файл*** последовательно выберите пункты ***Создать*** и ***Проект***.
3. В правом верхнем углу диалогового окна ***Создание проекта*** находится три значка. Щелкните левый значок и убедитесь, что установлен флажок ***.NET Framework версии 3.5.***
4. Щелкните значок ***Консольное приложение*** в области ***Установленные шаблоны Visual Studio***.
5. Присвойте приложению новое имя или примите имя по умолчанию и нажмите кнопку ОК.
6. Обратите внимание, что проект содержит ссылку на *System.Core.dll* и директиву *using* для пространства имен System.Linq.

Создание расположенного в памяти источника данных

Источником данных для запросов является простой список объектов *Student*. Каждая запись *Student* имеет имя, фамилию и массив целых чисел, представляющий результаты тестирования в классе. Скопируйте этот код в проект. Обратите внимание на следующие характеристики.

- Класс *Student* состоит из автоматически реализованных свойств.
- Каждый учащийся в списке инициализируется с помощью инициализатора объектов.
- Сам список инициализируется с помощью инициализатора коллекций.

Вся эта структура данных будет инициализирована и создана без явных вызовов конструкторов или явного доступа к членам.

Добавление источника данных

Добавьте класс *Student* и инициализированный список учащихся к классу *Program* в проекте.

```
public class Student
{
    public string First { get; set; }
    public string Last { get; set; }
    public int ID { get; set; }
    public List<int> Scores;
}

// Create a data source by using a collection initializer.
static List<Student> students = new List<Student>
{
    new Student {First="Svetlana", Last="Omelchenko", ID=111,
Scores= new List<int> {97, 92, 81, 60}},
    new Student {First="Claire", Last="O'Donnell", ID=112, Scores=
new List<int> {75, 84, 91, 39}},
    new Student {First="Sven", Last="Mortensen", ID=113, Scores=
new List<int> {88, 94, 65, 91}},
    new Student {First="Cesar", Last="Garcia", ID=114, Scores= new
List<int> {97, 89, 85, 82}},
    new Student {First="Debra", Last="Garcia", ID=115, Scores= new
List<int> {35, 72, 91, 70}},
    new Student {First="Fadi", Last="Fakhouri", ID=116, Scores= new
List<int> {99, 86, 90, 94}},
    new Student {First="Hanying", Last="Feng", ID=117, Scores= new
List<int> {93, 92, 80, 87}},
    new Student {First="Hugo", Last="Garcia", ID=118, Scores= new
List<int> {92, 90, 83, 78}},
    new Student {First="Lance", Last="Tucker", ID=119, Scores= new
List<int> {68, 79, 88, 92}},
    new Student {First="Terry", Last="Adams", ID=120, Scores= new
List<int> {99, 82, 81, 79}},
    new Student {First="Eugene", Last="Zabokritski", ID=121,
Scores= new List<int> {96, 85, 91, 60}},
    new Student {First="Michael", Last="Tucker", ID=122, Scores=
new List<int> {94, 92, 91, 91} }, List<int> {99, 82, 81, 79}},
```

```
    new Student {First="Natalia", Last="Lesnikovich", ID=123,
Scores= new List<int> {92, 90, 96, 77}},
    new Student {First="Olga", Last="Krivetz", ID=124, Scores= new
List<int> {99, 98, 98, 95}},
    new Student {First="Artem", Last="Borodich", ID=125, Scores=
new List<int> {91, 90, 90, 90}}};
```

Добавление нового учащегося в список учащихся

- Добавьте нового *Student* в список *Students* и используйте любое имя и результаты тестирования. Попробуйте набрать всю информацию о новом учащемся, чтобы лучше понять синтаксис инициализатора объектов.

Создание простого запроса

В методе Main создайте простой запрос, который при выполнении вернет список всех учащихся, результат тестирования которых превысил 90 баллов. Обратите внимание, что поскольку объект Student выбирается целиком, типом запроса будет IEnumerable<Student>. Хотя код мог также использовать неявную типизацию при помощи ключевого слова var, используется явная типизация, чтобы ясно показать результаты.

Обратите внимание, что переменная диапазона запроса student служит ссылкой на каждый объект Student в источнике, предоставляя доступ к членам для каждого объекта.

```
// Create the query.
// The first line could also be written as "var studentQuery ="
IEnumerable<Student> studentQuery =
  from student in students
  where student.Scores[0] > 90
  select student;
```

Выполнение запроса

1. Напишите цикл foreach, который приведет к выполнению запроса. Обратите внимание на следующие моменты.

- Доступ к каждому элементу в возвращаемой последовательности осуществляется с помощью переменной итерации в цикле foreach.
- Типом этой переменной является Student, а типом переменной запроса является совместимый IEnumerable<Student>.

2. После добавления этого кода выполните построение и запустите приложение, нажав сочетание клавиш Ctrl + F5 для просмотра результатов в окне Консоль.

```
// Execute the query.
// var could be used here also.
foreach (Student student in studentQuery)
{
   Console.WriteLine("{0}, {1}", student.Last, student.First);
}
```

Добавление дополнительного условия фильтра

Чтобы уточнить запрос, можно объединить несколько логических условий в предложении *where*. Следующий код добавляет условие таким образом, чтобы запрос возвращал тех учащихся, первый результат которых был более 90 баллов, а последний результат был меньше 80. Предложение where должно выглядеть следующим образом.

```
where student.Scores[0] > 90 && student.Scores[3] < 80
```

Изменение запроса

Упорядочение результатов

1. Просматривать результаты легче, если они как-то упорядочены. Возвращаемую последовательность можно упорядочить по любому доступному полю в исходных элементах. Например, следующее предложение *orderby* сортирует результаты в алфавитном порядке от А до Я по фамилии каждого учащегося. Добавьте следующее предложение orderby к запросу, указав его после инструкции where и перед оператором select.

```
orderby student.Last ascending
```

2. Теперь измените предложение orderby таким образом, чтобы оно сортировало результаты в обратном порядке по результату первого тестирования, от высшего к низшему показателю.

```
orderby student.Scores[0] descending
```

3. Измените строку форматирования WriteLine, чтобы видеть результаты тестирования.

```
Console.WriteLine("{0}, {1} {2}", student.Last, student.First, student.Scores[0]);
```

Группировка результатов

1. Группировка является мощной возможностью выражений запроса. Запрос с предложением *group* создает последовательность групп, где каждая группа содержит *Key* и последовательность, состоящую из всех членов этой группы. Следующий новый запрос группирует учащихся по первой букве их фамилии в качестве ключа.

```
// studentQuery2 is an IEnumerable<IGrouping<char, Student>>
var studentQuery2 =
   from student in students
   group student by student.Last[0];
```

2. Обратите внимание, что тип запроса изменился. Теперь он создает последовательность из групп, имеющих тип char в качестве ключа, и последовательность объектов Student. Поскольку тип запроса изменился, следующий код изменяет также и цикл foreach.

```
// studentGroup is a IGrouping<char, Student>
foreach (var studentGroup in studentQuery2)
{
   Console.WriteLine(studentGroup.Key);
   foreach (Student student in studentGroup)
   {
      Console.WriteLine("{0}, {1}", student.Last, student.First);
   }
}
```

3. Нажмите сочетание клавиш Ctrl + F5 для выполнения приложения и просмотра результатов в окне Консоль.

Присвоение переменной неявного типа

Явное кодирование IEnumerables из IGroupings может быстро стать трудоемким. С помощью var можно более просто написать тот же запрос и цикл foreach. Ключевое слово var не приводит к изменению типов объектов, оно просто сообщает компилятору о необходимости определения типов. Измените тип studentQuery и переменную итерации group на var и перезапустите запрос. Обратите внимание, что во внутреннем цикле foreach переменная итерации по-прежнему типизирована как Student и запрос работает так же, как и раньше. Измените переменную итерации s на var и повторно выполните запрос. Результаты остались неизменными.

```
var studentQuery3 =
   from student in students
   group student by student.Last[0];
foreach (var groupOfStudents in studentQuery3)
{
   Console.WriteLine(groupOfStudents.Key);
   foreach (var student in groupOfStudents)
   {
      Console.WriteLine("   {0}, {1}",
         student.Last, student.First);
   }
}
```

Упорядочение групп по значению их ключа

При выполнении предыдущего запроса группы были расположены не в алфавитном порядке. Для упорядочения необходимо указать предложение *orderby* после предложения *group*. Но, чтобы использовать предложение *orderby*, нужен идентификатор, служащий в качестве ссылки на группы, создаваемые предложением *group*. Предоставить идентификатор можно с помощью ключевого слова *into* следующим образом.

```
var studentQuery4 =
   from student in students
   group student by student.Last[0] into studentGroup
   orderby studentGroup.Key
   select studentGroup;

foreach (var groupOfStudents in studentQuery4)
{
   Console.WriteLine(groupOfStudents.Key);
   foreach (var student in groupOfStudents)
   {
      Console.WriteLine("   {0}, {1}",
         student.Last, student.First);
   }
}
```

При выполнении этого запроса группы будут отсортированы в алфавитном порядке.

Введение идентификаторов с помощью let

Ключевое слово let можно использовать для представления идентификатора для любого результата выражения в выражении запроса. Этот идентификатор может применяться для удобства, как в следующем примере, или он может повысить производительность, сохраняя результаты выражения так, чтобы оно не вычислялось повторно.

```
// studentQuery5 is an IEnumerable<string>
// This query returns those students whose first test score was higher
//than their average score.
var studentQuery5 =
   from student in students
   let totalScore = student.Scores[0] + student.Scores[1] +
      student.Scores[2] + student.Scores[3]
      where totalScore / 4 < student.Scores[0]
      select student.Last + " " + student.First;
```

```
foreach (string s in studentQuery5)
{
   Console.WriteLine(s);
}
```

Использование синтаксиса метода в выражении запроса

Некоторые операции запроса могут быть выражены только с помощью синтаксиса методов. В следующем коде вычисляется общий результат для каждого Student в исходной последовательности, а затем вызывается метод Average(), использующий результаты запроса для вычисления среднего балла класса. Обратите внимание на круглые скобки вокруг выражения запроса.

```
var studentQuery6 =
   from student in students
   let totalScore = student.Scores[0] + student.Scores[1] +
      student.Scores[2] + student.Scores[3]
      select totalScore;

double averageScore = studentQuery6.Average();
Console.WriteLine("Class average score = {0}", averageScore);
```

Преобразование или проецирование в предложении select

Очень часто в запросах создаются последовательности, элементы которых отличаются от элементов в исходных последовательностях. Удалите или закомментируйте предыдущий запрос и цикл и замените его следующим кодом. Обратите внимание, что запрос возвращает последовательность строк (не Students), и этот факт отражается в цикле foreach.

```
IEnumerable<string> studentQuery7 =
   from student in students
   where student.Last == "Garcia"
   select student.First;

Console.WriteLine("The Garcias in the class are:");
```

```
foreach (string s in studentQuery7)
{
   Console.WriteLine(s);
}
```

Приведенный ранее в этом руководстве код показывает, что средний результат для всего класса составляет примерно 334. Для создания последовательности Students, суммарный результат баллов которых больше среднего вместе с их Student ID, можно использовать анонимный тип в инструкции select.

```
var studentQuery8 =
   from student in students
   let x = student.Scores[0] + student.Scores[1] +
      student.Scores[2] + student.Scores[3]
   where x > averageScore
   select new { id = student.ID, score = x };

foreach (var item in studentQuery8)
{
   Console.WriteLine("Student ID: {0}, Score: {1}", item.id,
item.score);
}
```

ЛАБОРАТОРНАЯ РАБОТА № 10

ПРОСТАЯ МОДЕЛЬ ОБЪЕКТОВ И ПРОСТОЙ ЗАПРОС (C#) (LINQ TO SQL)

Данная работа включает следующие задачи:

- создание решения LINQ to SQL в среде Visual Studio;
- сопоставление класса с таблицей базы данных;
- назначение свойств классу для представления столбцов базы данных;
- указание подключения к базе данных Borei;
- создание простого запроса к базе данных;

- выполнение запроса и просмотр результатов.

Введение

Ниже представлены общие сведения, которые касаются всех последующих лабораторных работ, в т.ч. и части 3 данного пособия.

- Среда. В каждой работе по тематике «LINQ to SQL» в качестве интегрированной среды разработки используется Visual Studio.
- Ядро SQL. В данной и последующих лабораторных работах для реализации используют SQL Server Express.

Примечание. *В качестве строки подключения в данных лабораторных работах «LINQ to SQL» используется имя файла. Простое указания имени файла является одной из удобных возможностей, которые технология LINQ to SQL предоставляет пользователям SQL Server Express.*

В этом и последующих руководствах «LINQ to SQL» используется база данных "Borei". Для лабораторных работ, в которых рассматриваются многоуровневые сценарии, сервер должен быть установлен на компьютере, отличном от компьютера разработки, и у пользователя должны быть соответствующие права доступа к этому серверу.

- Именем класса, который обычно представляет таблицу "Goods" в базе данных "Borei", является [Good].

Устранение неполадок

Если у пользователя нет достаточных прав для доступа к базам данных, то во время выполнения работы могут возникать ошибки. Ниже приведены действия, которые могут помочь в решении наиболее распространенных проблем.

Проблемы входа

Чтобы проверить или изменить учетную запись для входа в базу данных, выполните следующие действия.

1. В меню ***Пуск*** операционной системы Windows последовательно выберите ***Все программы***, Microsoft SQL Server 2008, ***Сред-***

ства настройки, а затем щелкните пункт ***Диспетчер конфигурации SQL Server***.

2. В левой области Диспетчера конфигурации SQL Server выберите ***Службы SQL Server 2008***.

3. В правой области щелкните правой кнопкой мыши пункт SQL Server (SQLEXPRESS) и затем выберите ***Свойства***.

4. Перейдите на вкладку ***Вход*** и проверьте учетную запись, используемую для входа на сервер.

В большинстве случаев разрешено использовать учетную запись ***Локальная система.***

Если производятся изменения, щелкните **Перезапустить**, чтобы перезапустить службу.

Протоколы

В некоторых случаях, чтобы приложение могло получить доступ к базе данных, необходимо правильно настроить протоколы. Например, протокол ***Именованные каналы***, который требуется для выполнения лабораторных работ «LINQ to SQL», не включен по умолчанию.

Включение протокола именованных каналов

1. В левой области ***Диспетчера конфигурации SQL Server*** разверните узел ***Сетевая конфигурация SQL Server 2008*** и щелкните **Протоколы** для SQLEXPRESS.

2. В правой области проверьте, что протокол ***Именованные каналы*** включен. Если протокол не включен, щелкните правой кнопкой мыши пункт ***Именованные каналы*** и выберите команду ***Включить***. После этого необходимо остановить и снова запустить службу.

Выполните действия, описанные в следующем разделе.

Остановка и повторный запуск службы

1. В левой области ***Диспетчера конфигурации SQL Server*** выберите ***Службы SQL Server 2008***.

2. В правой области щелкните правой кнопкой мыши пункт SQL Server (SQLEXPRESS) и затем выберите команду ***Остановить***.

3. Щелкните правой кнопкой мыши пункт SQL Server (SQLEXPRESS) и выберите команду ***Перезапустить***.

В данной работе представлен базовый комплексный сценарий LINQ to SQL с подробным объяснением выполняемых действий. В нем создается класс сущностей, который моделирует таблицу *Suppliers* в образце базы данных *Borei*. После этого создается простой запрос на получение списка поставщиков, находящихся в Москве.

Данное руководство ориентировано на создание кода, чтобы продемонстрировать основные понятия технологии LINQ to SQL. Обычно с помощью ***Реляционного конструктора объектов*** создается собственная объектная модель.

Создание решения LINQ to SQL

1. В меню Файл среды Visual Studio укажите пункт ***Создать*** и выберите команду ***Проект***.
2. В диалоговом окне Создание проекта в области Тип проекта выберите Visual C#.
3. В области ***Шаблоны*** щелкните ***Консольное приложение***.
4. В поле ***Имя*** введите LinqConsoleApp.
5. В поле ***Расположение*** выберите папку для сохранения файлов проекта.
6. Нажмите кнопку ОК.

Добавление ссылок и директив LINQ

В лабораторной работе используются сборки, которые могут быть не установлены по умолчанию в проект. Если *System.Data.Linq* не входит в список ссылок проекта (разверните узел ***Ссылки*** в обозревателе решений), добавьте ее, как описано в следующих действиях.

Добавление сборки System.Data.Linq

1. В Обозревателе решений щелкните правой кнопкой мыши узел ***Ссылки*** и выберите команду ***Добавить ссылку***.

2. В диалоговом окне Добавление ссылки щелкните .NET, выберите сборку *System.Data.Linq*, а затем нажмите кнопку ОК.

Сборка будет добавлена в проект.

3. Добавьте следующие директивы в начало Program.cs.

```
using System.Data.Linq;
using System.Data.Linq.Mapping;
```

Сопоставление класса с таблицей базы данных

На этом этапе создается класс, который сопоставляется с таблицей базы данных. Подобный класс называется классом *сущностей*. Обратите внимание, что сопоставление осуществляется простым добавлением атрибута *TableAttribute*. Свойство *Name* задает имя таблицы в базе данных.

Для создания класса сущностей и его сопоставления с таблицей базы данных введите или вставьте следующий код в Program.cs непосредственно перед объявлением класса Program.

```
[Table(Name = "Suppliers")]
public class Supplier
{ }
```

Назначение свойств классу для представления столбцов базы данных

На этом этапе выполняется несколько задач.

- Используется атрибут *ColumnAttribute* для назначения классу сущностей свойств *CodeSuppliers* и *City*, представляющих столбцы в таблице базы данных.
- Назначается свойство *CodeSuppliers*, представляющее столбец первичного ключа в базе данных.
- Назначаются поля *_CodeSuppliers* и *_City* для закрытого хранения. После этого LINQ to SQL сможет извлекать и сохранять значения напрямую вместо использования открытых методов доступа, которые могут включать бизнес-логику.

Представление характеристик двух столбцов базы данных

- Для класса *Suppliers* введите или вставьте следующий код в Program.cs в фигурных скобках.

```
private int _CodeSuppliers;
[Column(IsPrimaryKey=true, Storage="_CodeSuppliers")]
public int CodeSuppliers
{
   get
   {
      return this._CodeSuppliers;
   }
   set
   {
      this._CodeSuppliers = value;
   }
}
private string _City;
[Column(Storage="_City")]
public string City
{
   get
   {
      return this._City;
   }
   set
   {
      this._City=value;
   }
}
```

Указание подключения к базе данных

На этом этапе для установки подключения между основанными на коде структурами данных и самой базой данных используется объект *DataContext*, который является основным каналом, через

который извлекаются объекты из базы данных и отправляются изменения источнику данных.

Также объявляется объект *Table<Supplier>*, который действует как логическая типизированная таблица для запросов к таблице "Suppliers" в базе данных. Эти запросы создаются и выполняются в последующих действиях.

- Введите или вставьте следующий код в метод Main.

```
// Use a connection string.
DataContext db = new DataContext
   (@"E:\дисциплины\borei.mdf");
// Get a typed table to run queries.
Table<Supplier> Suppliers = db.GetTable<Supplier>();
```

Создание простого запроса

На этом этапе создается запрос для поиска клиентов из таблицы *Suppliers* базы данных, находящихся в Москве. Код запроса, создаваемый на этом шаге, только описывает запрос, но не выполняет его. Подобный метод называется отложенным выполнением.

Можно также записать выходные данные в журнал, чтобы продемонстрировать команды SQL, создаваемые технологией LINQ to SQL. Функция ведения журнала (которая использует метод Log) очень полезна при отладке, а также при проверке того, что команды, отправляемые в базу данных, точно соответствуют запросу.

Введите или вставьте следующий код в метод Main после объявления Table<Supplier>.

```
// Attach the log to show generated SQL.
db.Log = Console.Out;
// Query for customers in London.
IQueryable<Supplier> suppQuery =
   from supp in Suppliers
   where supp.City == "Москва"
   select supp;
```

Выполнение запроса

На этом этапе осуществляется фактическое выполнение запроса. Выражения запроса, созданные на предыдущем этапе, не оцениваются до тех пор, пока не понадобятся результаты. После начала итерации foreach выполняется команда SQL для базы данных и объекты материализуются.

Введите или вставьте следующий код в конце метода Main (после описания запроса).

```
foreach (Supplier supp in suppQuery)
{
   Console.WriteLine("ID={0}, City={1}", supp.CodeSuppliers,
      supp.City);
}
// Prevent console window from closing.
Console.ReadLine();
```

Нажмите клавишу F5, чтобы начать отладку приложения.

1. В окне консоли отображаются следующие результаты запроса.
2. ID=1, City=Москва
3. ID=10, City=Москва
4. Чтобы закрыть приложение, в окне консоли нажмите клавишу ВВОД.

ЛИТЕРАТУРА

1. Solid Quality Learning Microsoft SQL Server 2005. Реализация и обслуживание. Учебный курс Microsoft / Пер. с англ. – М.: «Русская Редакция», СПб.: «Питер», 2007. – 768 с.: ил.
2. Электронная документация по SQL Server 2008 [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/ru-ru/library/ms365325.aspx, свободный. – Загл. с экрана.
3. Библиотека MSDN [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/ru-ru/library/default.aspx, свободный.- Загл. с экрана.

Учебное издание

БАЗЫ ЗНАНИЙ И ПОДДЕРЖКА ПРИНЯТИЯ РЕШЕНИЙ В САПР

Лабораторный практикум
для студентов специальности 1-40 01 02-01 «Информационные системы и технологии в проектировании и производстве»

В 3 частях

Ч а с т ь 2

С о с т а в и т е л и :
КОЧУРОВ Вадим Александрович
ГЕРМАН Юлия Олеговна

Технический редактор *О. В. Песенько*

Подписано в печать 24.12.2013. Формат $60\times84^{1}/_{16}$. Бумага офсетная. Ризография.
Усл. печ. л. 2,27. Уч.-изд. л. 1,77. Тираж 100. Заказ 437.
Издатель и полиграфическое исполнение: Белорусский национальный технический университет. ЛИ № 02330/0494349 от 16.03.2009. Пр. Независимости, 65. 220013, г. Минск.