ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Утверждаю

Заместитель директора по УМР

\_\_\_\_\_\_\_\_\_\_\_\_/Е.А. Родзик

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Методические рекомендации по выполнению   
учебной-практической работы №10

учебной практики   
УП 05. Программирование

Тема: «UWP работа с базой данных»

г. Томск – 2020 г

РАССМОТРЕННО

на заседании ПЦК

«Информационные системы и программирование»

протокол №\_\_\_\_\_\_

от «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Председатель ПЦК

\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Фунтиков М.Н.Рекомендации разработаны на основе Федерального государственного образовательного стандарта (далее – ФГОС) по специальностям среднего профессионального образования (далее СПО) 09.02.07 – «Информационные системы и программирование» и в соответствии с примерной основной образовательной программой.

Организация-разработчик:

ОГБОУ СПО «Томский техникум информационных технологий»

Разработчики:

\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сидиков И.Д.

\_\_\_\_\_\_\_\_\_\_\_\_\_/ Владимировна А. В.

**СОДЕРЖАНИЕ**

[1. Общая характеристика учебной-практической работы №10 4](#_Toc22557890)

[2. Краткие теоретические сведения 5](#_Toc22557891)

[3. Задания для выполнения 32](#_Toc22557892)

[4. Индивидуальные задания 33](#_Toc22557893)

[5. Контрольные вопросы 34](#_Toc22557894)

[6. Рекомендованная литература 35](#_Toc22557895)

[ПРИЛОЖЕНИЕ А 36](#_Toc22557896)

[ПРИЛОЖЕНИЕ Б 37](#_Toc22557897)

# ОБЩАЯ ХАРАКТЕРИСТИКА УЧЕБНОЙ-ПРАКТИЧЕСКОЙ РАБОТЫ №10

**Тема:** «UWP работа с базой данных».

**Цель работы:** получение навыков по взаимодействию базы данных с UWP на языке программирования C#

**Проверяемые компетенции:**

ОК 1, ОК 2, ОК 4, ОК 9, ПК11.2.

**Инструкция по выполнению:**

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения практической работы.
2. Выполните предложенное практическое задание.
3. Оформите отчет по учебной практике (шаблон представлен в приложении)
4. Отправьте отчет на проверку.

**Время выполнения заданий:** 6 часов.

**Критерии оценки:**

|  |  |  |
| --- | --- | --- |
| **№** | **Критерий** | **Баллы** |
| 1 | Отчет соответствует предложенному шаблону | 1 |
| 2 | Задание выполнено правильно | 2 |
| 3 | Представлен код программы и интерфейс | 4 |
| 4 | Составлена блок схема к программе | 2 |
| 5 | Программа работает правильно без сбоев | 2 |
| 6 | Студент ответил на контрольные вопросы по лабораторной работе | 2 |
| 7 | Оформление отчета соответствует требованиям (шрифт, поля, отступы, интервалы, оформление рисунков, автоматическое оглавление) | 1 |
| 8 | Своевременность выполнения задания | 1 |
| Итого | | 15 |

**Перевод в пятибалльную систему оценивания:**

«отлично» - 13 - 15 баллов

«хорошо» - 10 -12 баллов

«удовлетворительно» - 7-9

«неудовлетворительно» <7 баллов

# КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

# Использование базы данных SQLite в приложении UWP

**Некоторые преимущества использования SQLite для локального хранения данных**

✔️ SQLite легкая и автономная. Это библиотека кода без других зависимых компонентов. Ее не требуется настраивать.

✔️ Тут нет сервера базы данных. Клиент и сервер работают в одном процессе.

✔️ База данных SQLite находится в общедоступном домене, поэтому вы можете свободно использовать и распространять ее в вашем приложении.

✔️ SQLite работает на разных платформах и архитектурах.

Подробнее о SQLite см. [здесь](https://sqlite.org/about.html).

**Выбор слоя абстрагирования**

Мы рекомендуем использовать Entity Framework Core или [библиотеку SQLite](https://github.com/aspnet/Microsoft.Data.Sqlite/) с открытым исходным кодом, встроенные корпорацией Майкрософт.

**Entity Framework Core**

Entity Framework (EF) — это объектно-реляционный модуль сопоставления, позволяющий разработчикам работать с реляционными данными с помощью объектов, специализированных для доменов. Если вы уже использовали эту платформу для работы с данными в других приложениях .NET, можно перенести этот код в приложение UWP и он будет работать с соответствующими изменениями в строке подключения.

Чтобы опробовать эту платформу, см. раздел [Начало работы с EF Core на универсальной платформе Windows (UWP) с новой базой данных](https://docs.microsoft.com/ef/core/get-started/uwp/getting-started).

**Библиотеке SQLite**

Библиотека [Microsoft.Data.Sqlite](https://docs.microsoft.com/dotnet/api/microsoft.data.sqlite?view=msdata-sqlite-2.0.0) реализует интерфейсы в пространстве имен [System.Data.Common](https://docs.microsoft.com/dotnet/api/system.data.common). Майкрософт активно поддерживает эти реализации и предоставляет интуитивно понятную оболочку для низкоуровневых собственных API-интерфейсов SQLite.

В остальных разделах этого руководства приведены инструкции по работе с этой библиотекой.

**Настройка решения для использования библиотеки Microsoft.Data.SQlite**

Мы начнем с базового проекта UWP, добавим библиотеку классов, а затем установим соответствующие пакеты NuGet.

Тип библиотеки классов, добавляемой в решение, и специальные пакеты, которые вы устанавливаете, зависят от минимальной версии Windows SDK, для которого предназначено ваше приложение. Эти сведения можно найти на странице свойств проекта UWP.

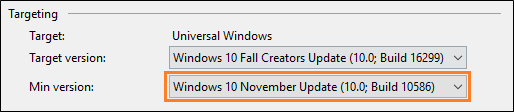


Рисунок 1. Выбор целевой версии OS

Используйте один из следующих разделов в зависимости от минимальной версии Windows SDK, для которого предназначен вам проект UWP.

**Минимальная версия проекта не предназначена для обновления Fall Creators Update**

Если вы используете Visual Studio 2015, щелкните **Справка**->**Сведения о Microsoft Visual Studio**. Затем в списке установленных программ убедитесь, что версия диспетчера пакетов NuGet — **3.5** или более поздняя версия. Если номер версии ниже, установите более позднюю версию NuGet. Скачать ее можно [здесь](https://www.nuget.org/downloads). На этой странице вы найдете все версии Nuget, перечисленные под заголовком **Visual Studio 2015**.

Затем добавьте библиотеку классов в свое решение. Использовать библиотеку классов для размещения вашего кода доступа к данным не требуется, но мы используем ее в нашем примере. Мы назовем библиотеку **DataAccessLibrary**, а класс в библиотеке — **DataAccess**.

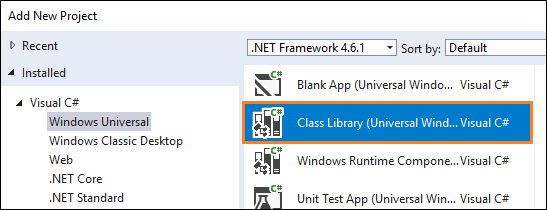


Рисунок 2. Создание класса

Щелкните правой кнопкой мыши решение, а затем нажмите кнопку **Управление пакетами NuGet для решения**.

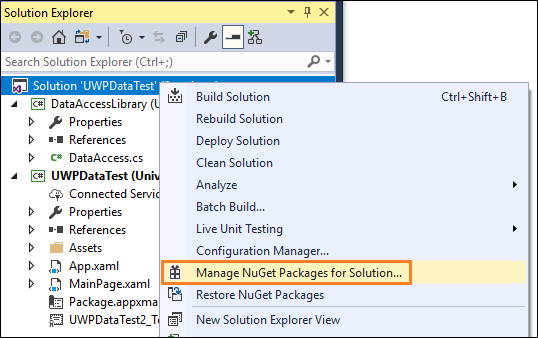


Рисунок 3. Менеджер пакетов NuGet

Если вы используете Visual Studio 2015, выберите вкладку **Установленные** и убедитесь, что версия пакета **Microsoft.NETCore.UniversalWindowsPlatform** — **5.2.2** или более поздняя версия.

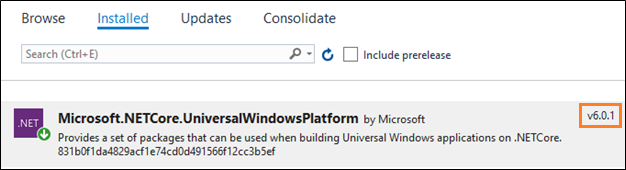


Рисунок 4. Установленные пакеты в проекте

Если это не так, обновите пакет до более новой версии.

Выберите вкладку **Обзор** и выполните поиск пакета **Microsoft.Data.SQLite**. Установите версию пакета **1.1.1** (или ниже).

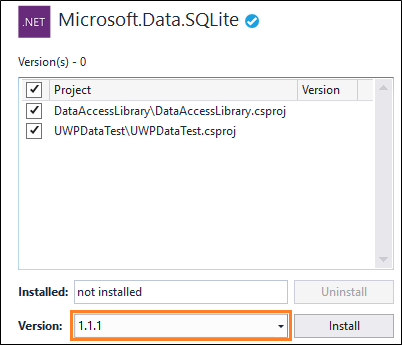


Рисунок 5. Выбор версии пакета

Перейдите к разделу [Добавление данных в базу данных SQLite и их извлечение](https://docs.microsoft.com/ru-ru/windows/uwp/data-access/sqlite-databases#use-data) данного руководства.

**Добавление данных в базу данных SQLite и их извлечение**

Мы выполним следующие действия.

1. Подготовка класса доступа к данным.
2. Инициализация базы данных SQLite.
3. Вставка данных в базу данных SQLite.
4. Извлечение данных из базы данных SQLite.
5. Добавление базового пользовательского интерфейса.

**Подготовка класса доступа к данным**

В проекте UWP добавьте ссылку на проект **DataAccessLibrary** в своем решении.

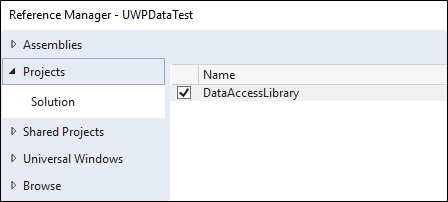


Рисунок 6. Добавление ссылок в проект

Добавьте следующий оператор using в файлы **App.xaml.cs** и **MainPage.xaml.cs** в проекте UWP.

|  |
| --- |
| С# |
| using DataAccessLibrary; |

Откройте класс **DataAccess** в вашем решении **DataAccessLibrary** и сделайте этот класс статическим.

|  |
| --- |
| Примечание!  В нашем примере мы размещаем код доступа к данным в статическом классе, однако это лишь проектное решение, которое не является обязательным. |

|  |
| --- |
| С# |
| namespace DataAccessLibrary  {  public static class DataAccess  {  }  } |

Добавьте следующие операторы использования в начало этого файла.

|  |
| --- |
| С# |
| using Microsoft.Data.Sqlite;  using System.Collections.Generic; |

**Инициализация базы данных SQLite**

Добавьте метод в класс **DataAccess**, который инициализирует базу данных SQLite.

|  |
| --- |
| С# |
| public async static void InitializeDatabase()  {  await ApplicationData.Current.LocalFolder.CreateFileAsync("sqliteSample.db", CreationCollisionOption.OpenIfExists);  string dbpath = Path.Combine(ApplicationData.Current.LocalFolder.Path, "sqliteSample.db");  using (SqliteConnection db =  new SqliteConnection($"Filename={dbpath}"))  {  db.Open();  String tableCommand = "CREATE TABLE IF NOT " +  "EXISTS MyTable (Primary\_Key INTEGER PRIMARY KEY, " +  "Text\_Entry NVARCHAR(2048) NULL)";  SqliteCommand createTable = new SqliteCommand(tableCommand, db);  createTable.ExecuteReader();  }  } |

Этот код создает базу данных SQLite и сохраняет ее в локальном хранилище данных приложения.

В этом примере мы используем имя базы данных sqlliteSample.db, но вы можете использовать любое имя при условии, что оно будет использоваться во всех объектах [SqliteConnection](https://docs.microsoft.com/dotnet/api/microsoft.data.sqlite.sqliteconnection?view=msdata-sqlite-2.0.0), экземпляры которых необходимо создать.

В конструкторе файла **App.xaml.cs** проекта UWP вызовите метод InitializeDatabase класса **DataAccess**.

|  |
| --- |
| С# |
| public App()  {  this.InitializeComponent();  this.Suspending += OnSuspending;  DataAccess.InitializeDatabase();  } |

**Вставка данных в базу данных SQLite**

Добавьте метод в класс **DataAccess**, который вставляет данные в базу данных SQLite. Этот код использует параметры в запросе для предотвращения атак путем внедрения кода SQL.

|  |
| --- |
| С# |
| public static void AddData(string inputText)  {  string dbpath = Path.Combine(ApplicationData.Current.LocalFolder.Path, "sqliteSample.db");  using (SqliteConnection db =  new SqliteConnection($"Filename={dbpath}"))  {  db.Open();  SqliteCommand insertCommand = new SqliteCommand();  insertCommand.Connection = db;  // Use parameterized query to prevent SQL injection attacks  insertCommand.CommandText = "INSERT INTO MyTable VALUES (NULL, @Entry);";  insertCommand.Parameters.AddWithValue("@Entry", inputText);  insertCommand.ExecuteReader();  db.Close();  }  } |

**Извлечение данных из базы данных SQLite**

Добавьте метод, который возвращает строки данных из базы данных SQLite.

|  |
| --- |
| С# |
| public static List<String> GetData()  {  List<String> entries = new List<string>();  string dbpath = Path.Combine(ApplicationData.Current.LocalFolder.Path, "sqliteSample.db");  using (SqliteConnection db =  new SqliteConnection($"Filename={dbpath}"))  {  db.Open();  SqliteCommand selectCommand = new SqliteCommand  ("SELECT Text\_Entry from MyTable", db);  SqliteDataReader query = selectCommand.ExecuteReader();  while (query.Read())  {  entries.Add(query.GetString(0));  }  db.Close();  }  return entries;  } |

Метод [Read](https://docs.microsoft.com/dotnet/api/microsoft.data.sqlite.sqlitedatareader.read?view=msdata-sqlite-2.0.0" \l "Microsoft_Data_Sqlite_SqliteDataReader_Read) считывает строки возвращаемых данных. Он возвращает значение **true**, если остались строки. В противном случае он возвращает значение **false** .

Метод [GetString](https://docs.microsoft.com/dotnet/api/microsoft.data.sqlite.sqlitedatareader.getstring?view=msdata-sqlite-2.0.0" \l "Microsoft_Data_Sqlite_SqliteDataReader_GetString_System_Int32_) возвращает значение конкретного столбца в виде строки. Он принимает целое число, представляющее собой отсчитываемый от нуля порядковый номер столбца необходимых данных. Можно использовать похожие методы, такие как [GetDataTime](https://docs.microsoft.com/dotnet/api/microsoft.data.sqlite.sqlitedatareader.getdatetime?view=msdata-sqlite-2.0.0" \l "Microsoft_Data_Sqlite_SqliteDataReader_GetDateTime_System_Int32_) и [GetBoolean](https://docs.microsoft.com/dotnet/api/microsoft.data.sqlite.sqlitedatareader.getboolean?view=msdata-sqlite-2.0.0#Microsoft_Data_Sqlite_SqliteDataReader_GetBoolean_System_Int32_). Выберите метод в зависимости от типа данных в столбце.

Порядковый номер параметра не так важен в этом примере, так как мы выбираем все записи в одном столбце. Тем не менее, если несколько столбцов являются частью запроса, используйте порядковое значение для получения столбца, из которого требуется извлечь данные.

**Добавление базового пользовательского интерфейса**

В файл **MainPage.xaml** проекта UWP добавьте следующий XAML-код.

|  |
| --- |
| XML |
| <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">  <StackPanel>  <TextBox Name="Input\_Box"></TextBox>  <Button Click="AddData">Add</Button>  <ListView Name="Output">  <ListView.ItemTemplate>  <DataTemplate>  <TextBlock Text="{Binding}"/>  </DataTemplate>  </ListView.ItemTemplate>  </ListView>  </StackPanel>  </Grid> |

Этот базовый пользовательский интерфейс предоставляет пользователю поле TextBox, которое можно использовать для ввода строки, которую мы добавим в базу данных SQLite. Мы подключимся Buttonв этом пользовательском интерфейсе к обработчику событий, который будет получать данные из базы данных SQLite и затем отображать их в ListView.

В файл **MainPage.xaml.cs** добавьте следующий обработчик. Это метод, который мы связали с кнопкой Click событие Button в пользовательском интерфейсе.

|  |
| --- |
| C# |
| private void AddData(object sender, RoutedEventArgs e)  {  DataAccess.AddData(Input\_Box.Text);  Output.ItemsSource = DataAccess.GetData();  } |

Готово! Изучите [Microsoft.Data.Sqlite](https://docs.microsoft.com/dotnet/api/microsoft.data.sqlite?view=msdata-sqlite-2.0.0) чтобы узнать, что еще можно сделать с вашей базой данных SQLite. Перейдите по ссылкам ниже, чтобы узнать о других способах использования данных в приложении UWP.

# Использование базы данных SQL Server в приложении UWP

Ваше приложение может подключаться напрямую к базе данных SQL Server и затем хранить и извлекать данные с помощью классов в пространстве имен [System.Data.SqlClient](https://docs.microsoft.com/dotnet/api/system.data.sqlclient).

В данном руководстве приведен один из способов выполнения этой задачи. Если вы установите пример базы данных [Northwind](https://docs.microsoft.com/dotnet/framework/data/adonet/sql/linq/downloading-sample-databases) поверх экземпляра SQL Server, а затем используете эти фрагменты кода, у вас получится базовый пользовательский интерфейс, в котором демонстрируются продукты из примера базы данных Northwind.

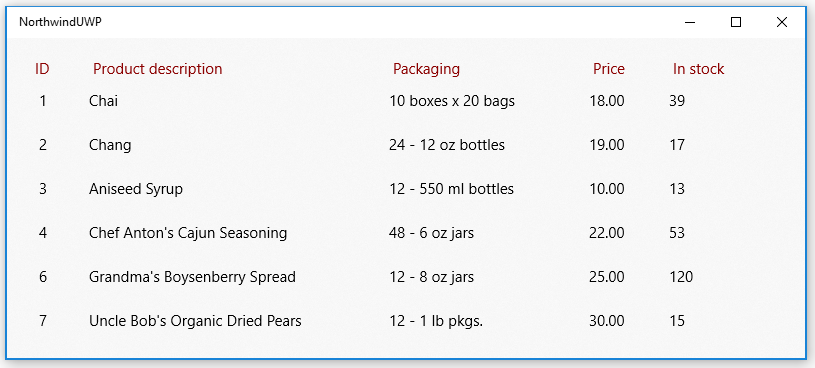


Рисунок 7. Пример работы UWP c базой данных

В основе фрагментов кода, представленных в этом руководстве, лежит более [полный пример кода](https://github.com/StefanWickDev/IgniteDemos/tree/master/NorthwindDemo).

**Сначала следует настроить решение**

Чтобы подключить приложение напрямую к базе данных SQL Server, убедитесь, что минимальная версия вашего проекта поддерживает Fall Creators Update. Эти сведения можно найти на странице свойств проекта UWP.

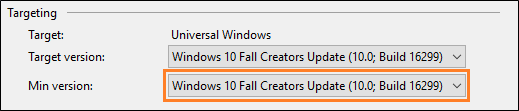


Рисунок 8. Настройка целевой версии OS

Откройте файл **Package.appxmanifest** в конструкторе манифестов проекта UWP.

Если для проверки подлинности SQL Server вы используете проверку подлинности Windows, то на вкладке **Возможности** следует выбрать **Корпоративная проверка подлинности**.

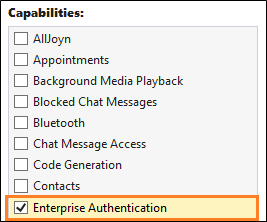


Рисунок 9. Корпоративная проверка пользователей

**Добавление данных в базу данных SQL Server и их извлечение**

В этом разделе будут выполнены следующие действия.

* Добавление строки подключения.
* Создание класса для хранения данных продукта.
* Получение продуктов из базы данных SQL Server.
* Добавление базового пользовательского интерфейса.
* Заполнение пользовательского интерфейса продуктами.

|  |
| --- |
| Примечание!  В этом разделе показан один из способов организации кода доступа к данным. Его задачей является показать пример того, как можно использовать System.Data.SqlClient для хранения данных в базе данных SQL Server и их извлечения. Код можно организовывать таким образом, который лучше всего подходит для проекта приложения. |

**Добавление строки подключения**

В файле **App.xaml.cs** добавьте свойство в класс App, который предоставляет другим классам в решении доступ к строке подключения.

Строка подключения указывает на базу данных Northwind в экземпляре SQL Server Express.

|  |
| --- |
| C# |
| sealed partial class App : Application  {  // Connection string for using Windows Authentication.  private string connectionString =  @"Data Source=YourServerName\SQLEXPRESS;Initial Catalog=NORTHWIND;Integrated Security=SSPI";  // This is an example connection string for using SQL Server Authentication.  // private string connectionString =  // @"Data Source=YourServerName\YourInstanceName;Initial Catalog=DatabaseName; User Id=XXXXX; Password=XXXXX";  public string ConnectionString { get => connectionString; set => connectionString = value; }  ...  } |

**Создание класса для хранения данных продукта**

Создадим класс, реализующий событие [INotifyPropertyChanged](https://docs.microsoft.com/dotnet/api/system.componentmodel.inotifypropertychanged), чтобы можно было привязать атрибуты в пользовательском интерфейсе XAML к свойствам этого класса.

|  |
| --- |
| C# |
| public class Product : INotifyPropertyChanged  {  public int ProductID { get; set; }  public string ProductCode { get { return ProductID.ToString(); } }  public string ProductName { get; set; }  public string QuantityPerUnit { get; set; }  public decimal UnitPrice { get; set; }  public string UnitPriceString { get { return UnitPrice.ToString("######.00"); } }  public int UnitsInStock { get; set; }  public string UnitsInStockString { get { return UnitsInStock.ToString("#####0"); } }  public int CategoryId { get; set; }  public event PropertyChangedEventHandler PropertyChanged;  private void NotifyPropertyChanged(string propertyName)  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }  } |

**Получение продуктов из базы данных SQL Server**

Создайте метод, который будет получать продукты из примера базы данных Northwind и возвращать их в качестве коллекции [ObservableCollection](https://docs.microsoft.com/dotnet/api/system.collections.objectmodel.observablecollection-1) экземпляров Product.

|  |
| --- |
| C# |
| public ObservableCollection<Product> GetProducts(string connectionString)  {  const string GetProductsQuery = "select ProductID, ProductName, QuantityPerUnit," +  " UnitPrice, UnitsInStock, Products.CategoryID " +  " from Products inner join Categories on Products.CategoryID = Categories.CategoryID " +  " where Discontinued = 0";  var products = new ObservableCollection<Product>();  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  conn.Open();  if (conn.State == System.Data.ConnectionState.Open)  {  using (SqlCommand cmd = conn.CreateCommand())  {  cmd.CommandText = GetProductsQuery;  using (SqlDataReader reader = cmd.ExecuteReader())  {  while (reader.Read())  {  var product = new Product();  product.ProductID = reader.GetInt32(0);  product.ProductName = reader.GetString(1);  product.QuantityPerUnit = reader.GetString(2);  product.UnitPrice = reader.GetDecimal(3);  product.UnitsInStock = reader.GetInt16(4);  product.CategoryId = reader.GetInt32(5);  products.Add(product);  }  }  }  }  }  return products;  }  catch (Exception eSql)  {  Debug.WriteLine("Exception: " + eSql.Message);  }  return null;  } |

**Добавление базового пользовательского интерфейса**

Добавьте следующий XAML-код в файл **MainPage.xaml** проекта UWP.

Этот XAML-код создает [ListView](https://docs.microsoft.com/uwp/api/windows.ui.xaml.controls.listview) для отображения каждого продукта, возвращенного в предыдущем фрагменте кода, и привязывает атрибуты каждой строки в [ListView](https://docs.microsoft.com/uwp/api/windows.ui.xaml.controls.listview) к свойствам, определенным в классе Product.

|  |
| --- |
| XML |
| <Grid Background="{ThemeResource SystemControlAcrylicWindowBrush}">  <RelativePanel>  <ListView Name="InventoryList"  SelectionMode="Single"  ScrollViewer.VerticalScrollBarVisibility="Auto"  ScrollViewer.IsVerticalRailEnabled="True"  ScrollViewer.VerticalScrollMode="Enabled"  ScrollViewer.HorizontalScrollMode="Enabled"  ScrollViewer.HorizontalScrollBarVisibility="Auto"  ScrollViewer.IsHorizontalRailEnabled="True"  Margin="20">  <ListView.HeaderTemplate>  <DataTemplate>  <StackPanel Orientation="Horizontal" >  <TextBlock Text="ID" Margin="8,0" Width="50" Foreground="DarkRed" />  <TextBlock Text="Product description" Width="300" Foreground="DarkRed" />  <TextBlock Text="Packaging" Width="200" Foreground="DarkRed" />  <TextBlock Text="Price" Width="80" Foreground="DarkRed" />  <TextBlock Text="In stock" Width="80" Foreground="DarkRed" />  </StackPanel>  </DataTemplate>  </ListView.HeaderTemplate>  <ListView.ItemTemplate>  <DataTemplate x:DataType="local:Product">  <StackPanel Orientation="Horizontal" >  <TextBlock Name="ItemId"  Text="{x:Bind ProductCode}"  Width="50" />  <TextBlock Name="ItemName"  Text="{x:Bind ProductName}"  Width="300" />  <TextBlock Text="{x:Bind QuantityPerUnit}"  Width="200" />  <TextBlock Text="{x:Bind UnitPriceString}"  Width="80" />  <TextBlock Text="{x:Bind UnitsInStockString}"  Width="80" />  </StackPanel>  </DataTemplate>  </ListView.ItemTemplate>  </ListView>  </RelativePanel>  </Grid> |

**Отображение продуктов в ListView**

Откройте файл **MainPage.xaml.cs** и добавьте код в конструктор класса MainPage, который задает свойство **ItemSource** в [ListView](https://docs.microsoft.com/uwp/api/windows.ui.xaml.controls.listview) для [ObservableCollection](https://docs.microsoft.com/dotnet/api/system.collections.objectmodel.observablecollection-1) экземпляров Product.

|  |
| --- |
| C# |
| public MainPage()  {  this.InitializeComponent();  InventoryList.ItemsSource = GetProducts((App.Current as App).ConnectionString);  } |

Запустите проект и посмотрите, как продукты из примера базы данных Northwind отображаются в пользовательском интерфейсе.

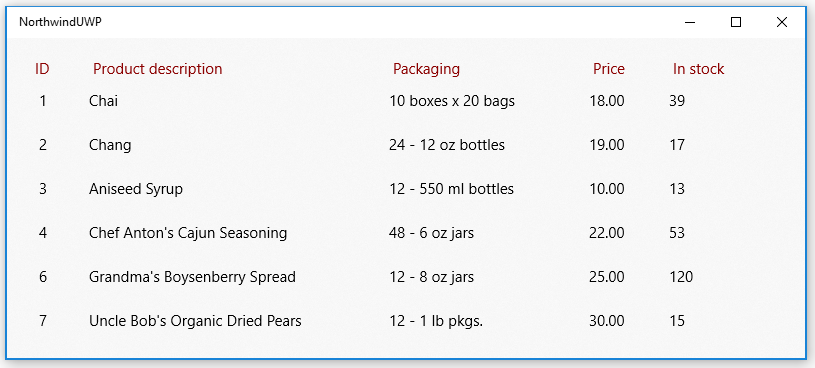


Рисунок 10. Пример работы базы данных с UWP

Чтобы узнать, что еще можно сделать с данными в базе данных SQL Server, изучите пространство имен [System.Data.SqlClient](https://docs.microsoft.com/dotnet/api/system.data.sqlclient).

# Основные операции с базой данных SQLite

После создания моделей и добавления Entity Framework создадим собственно элементы управления, которые позволят нам добавлять, обновлять, удалять и просматривать данные из базы данных.

По умолчанию у нас уже есть страница MainPage.xaml. Пусть она будет выводить список компаний, то есть объектов класса Company, а также будет содержать элементы для управления этими объектами. Итак, изменим MainPage.xaml следующим образом:

|  |
| --- |
| XML |
| <Page      x:Class="SQLiteApp.MainPage"      xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"      xmlns:local="using:SQLiteApp"      xmlns:d="<http://schemas.microsoft.com/expression/blend/2008>"      xmlns:mc="<http://schemas.openxmlformats.org/markup-compatibility/2006>"      mc:Ignorable="d">      <Page.BottomAppBar>          <CommandBar>              <AppBarButton x:Name="saveButton" Label="Добавить" Icon="Add" Click="Add\_Click" />              <AppBarButton x:Name="editButton" Label="Изменить" Icon="Edit" Click="Edit\_Click" />              <AppBarButton x:Name="deleteButton" Label="Удалить" Icon="Delete" Click="Delete\_Click" />          </CommandBar>      </Page.BottomAppBar>      <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">          <ListView x:Name="companiesList">              <ListView.ItemTemplate>                  <DataTemplate  x:DataType="local:Company">                      <StackPanel>                          <TextBlock Text="{Binding Name}"  FontSize="26" />                      </StackPanel>                  </DataTemplate>              </ListView.ItemTemplate>          </ListView>      </Grid>  </Page> |

Для отображения списка определен элемент ListView, а для управления компаниями в нижней панели определены три кнопки. Теперь в файле кода MainPage.xaml.cs определим для них обработчики нажатия:

|  |
| --- |
| C# |
| public sealed partial class MainPage : Page  {      public MainPage()      {          this.InitializeComponent();          this.Loaded += MainPage\_Loaded;      }        private void MainPage\_Loaded(object sender, RoutedEventArgs e)      {          using (MobileContext db = new MobileContext())          {              companiesList.ItemsSource = db.Companies.ToList();          }      }        private void Add\_Click(object sender, RoutedEventArgs e)      {          Frame.Navigate(typeof(CompanyPage));      }        private void Edit\_Click(object sender, RoutedEventArgs e)      {          // получаем выделеный пункт меню          if (companiesList.SelectedItem != null)          {              Company company = companiesList.SelectedItem as Company;              if (company != null)                  Frame.Navigate(typeof(CompanyPage), company.Id);          }      }        private void Delete\_Click(object sender, RoutedEventArgs e)      {          // получаем выделеный пункт меню          if (companiesList.SelectedItem != null)          {              Company company = companiesList.SelectedItem as Company;              if (company != null)              {                  using (MobileContext db = new MobileContext())                  {                      db.Companies.Remove(company);                      db.SaveChanges();                      companiesList.ItemsSource = db.Companies.ToList();                  }              }          }      }  } |

При загрузке страницы срабатывает обработчик MainPage\_Loaded, в котором получаем список компаний из базы данных и устанавливаем его в качестве источника данных для ListView:

|  |
| --- |
| C# |
| companiesList.ItemsSource = db.Companies.ToList(); |

Обработчик кнопки добавления Add\_Click просто перенаправляет на страницу CompanyPage, которую мы чуть позже создадим.

Обработчик кнопки редактирования Edit\_Click получает выделенный объект Company в списке и передает его id на страницу CompanyPage.

Обработчик кнопки удаления Delete\_Click получает выделенный объект Company и передает его в метод Remove:

|  |
| --- |
| C# |
| db.Companies.Remove(company); |

Этот метод позволяет сгенерировать SQL-выражение DELETE для удаления данных из таблицы компаний. После этого надо сохранить все изменения, то есть выполнить это SQL-выражение с помощью вызова:

|  |
| --- |
| C# |
| db.SaveChanges(); |

И далее переустанавливаем источник данных для ListView.

В итоге главная страница будет выглядеть примерно так:

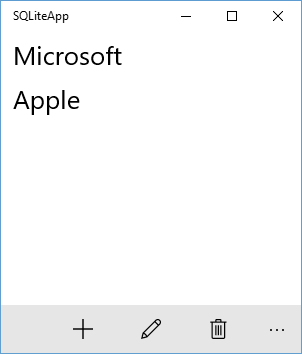


Рисунок 11. Отображение данных из базы данных

Для добавления/редактирования добавим новую страницу по типу BlankPage, которую назовем CompanyPage:

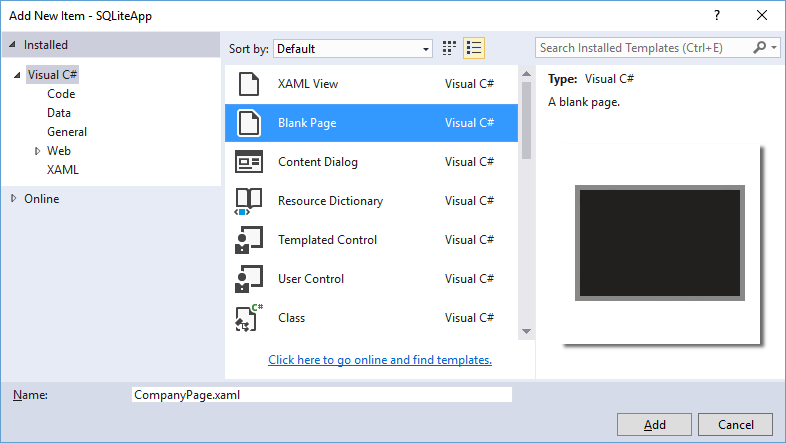


Рисунок 12. Добавление новой страницы

Так как модель Company имеет, по сути, одно значимое свойство - Name, то много нам добавлять и редактировать не надо, и для обеих операций нам хватит одной страницы. Тем более эти операции у нас будут отличаться только наличием Id при редактировании.

Изменим начальный код страницы CompanyPage.xaml на следующий:

|  |
| --- |
| XML |
| <Page      x:Class="SQLiteApp.CompanyPage"      xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"      xmlns:local="using:SQLiteApp"      xmlns:d="<http://schemas.microsoft.com/expression/blend/2008>"      xmlns:mc="<http://schemas.openxmlformats.org/markup-compatibility/2006>"      mc:Ignorable="d">        <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">          <StackPanel Margin="10 0 10 0">              <TextBlock x:Name="headerBlock" Text = "Добавление компании"                         Style="{ThemeResource HeaderTextBlockStyle}" />              <TextBlock>Название компании</TextBlock>              <TextBox x:Name="nameBox" />              <StackPanel Orientation="Horizontal">                  <Button Content="Сохранить" Click="Save\_Click" Margin="10" />                  <Button Content="Отмена" Click="Cancel\_Click" Margin="10" />              </StackPanel>          </StackPanel>      </Grid>  </Page> |

Здесь одно текстовое поле для редактирования названия компании и две кнопки: для сохранения изменений и отмены действий.

Далее изменим код C#, добавив в него обработчики кнопок:

|  |
| --- |
| C# |
| public sealed partial class CompanyPage : Page  {  Company company;    public CompanyPage()  {  this.InitializeComponent();  }    protected override void OnNavigatedTo(NavigationEventArgs e)  {  if (e.Parameter != null)  {  int id = (int)e.Parameter;  using (MobileContext db = new MobileContext())  {  company = db.Companies.FirstOrDefault(c => c.Id == id);  }  }    if (company != null)  {  headerBlock.Text = "Редактирование компании";  nameBox.Text = company.Name;  }  }    private void Save\_Click(object sender, RoutedEventArgs e)  {  using (MobileContext db = new MobileContext())  {  if(company!=null)  {  company.Name = nameBox.Text;  db.Companies.Update(company);  }  else  {  db.Companies.Add(new Company { Name = nameBox.Text });  }  db.SaveChanges();  }  GoToMainPage();  }    private void Cancel\_Click(object sender, RoutedEventArgs e)  {  GoToMainPage();  }    private void GoToMainPage()  {  if (Frame.CanGoBack)  Frame.GoBack();  else  Frame.Navigate(typeof(MainPage));  }  } |

При переходе на страницу срабатывает метод OnNavigatedTo(). В нем мы смотрим на переданный параметр. Если он не равен null, значит, нам надо отредактировать сушность, идентификатор которой равен параметру. В ином случае надо выполнить добавление. По этому идентифкатору мы получаем компанию из бд:

|  |
| --- |
| C# |
| company = db.Companies.FirstOrDefault(c => c.Id == id); |

При нажатии на кнопку сохранения смотрим, была ли получена ранее компания. Если она была получена, значит, нам надо отредактировать ее. Для этого объект компании передается в метод Update():

|  |
| --- |
| C# |
| db.Companies.Update(company); |

Он генерирует выражение SQL UPDATE для обновления записей в базе данных.

Если же компания при загрузке страницы не была установлена, то выполняем добавление в базе данных с помощью метода Add():

|  |
| --- |
| C# |
| db.Companies.Add(new Company { Name = nameBox.Text }); |

Далее выполняем сгенерированное SQL-выражение с помощью метода db.SaveChanges(). И после всех изменений возвращаемся на главную страницу.

В итоге CompanyPage будет выглядеть примерно так:

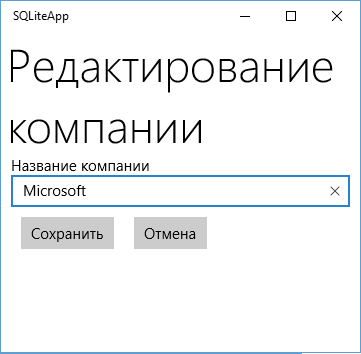


Рисунок 13. Редактирование компании

Таким образом, мы создали простейшее приложение, которое выполняет все базовые операции с данными: чтение, добавление, обновление и удаление.

# Работа со сложными данными

В прошлой теме мы рассмотрели работу с простой моделью Company, объекты которой хранятся в базе данных. Но у нас еще есть другая модель - Phone, которая связана с Company отношением "один-ко-многим". Рассмотрим вкратце, как мы можем работать и с этой моделью.

Для работы с моделью Phone добавим в проект новую страницу. Назовем ее PhonesListPage.xaml и установим у нее следующее содержимое:

|  |
| --- |
| XML |
| <Page      x:Class="SQLiteApp.PhonesListPage"      xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"      xmlns:local="using:SQLiteApp"      xmlns:d="<http://schemas.microsoft.com/expression/blend/2008>"      xmlns:mc="<http://schemas.openxmlformats.org/markup-compatibility/2006>"      mc:Ignorable="d">      <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">          <Grid.RowDefinitions>              <RowDefinition Height="Auto" />              <RowDefinition />          </Grid.RowDefinitions>          <StackPanel>              <TextBox Name="nameBox" PlaceholderText="Введите модель" Margin="10" />              <TextBox Name="priceBox" PlaceholderText="Введите цену" Margin="10" />              <ComboBox x:Name="companiesList" DisplayMemberPath="Name" SelectedValuePath="Id"                      Margin="10" PlaceholderText="Выберите компанию" />              <StackPanel Orientation="Horizontal">                  <Button Content="Добавить" Click="Add\_Click" Margin="10" />                  <Button Content="Назад" Click="Back" Margin="10" />              </StackPanel>          </StackPanel>          <ListView x:Name="phonesList" Grid.Row="1" Margin="10">              <ListView.ItemTemplate>                  <DataTemplate  x:DataType="local:Phone">                      <StackPanel>                          <TextBlock FontSize="16" Text="{x:Bind Title}" FontWeight="Bold"  />                          <TextBlock FontSize="16" Text="{x:Bind Company.Name}" />                          <TextBlock FontSize="16" Text="{x:Bind Price}" />                      </StackPanel>                  </DataTemplate>              </ListView.ItemTemplate>          </ListView>      </Grid>  </Page> |

Сначала на странице идет набор полей для добавления нового объекта Phone. И под ними располагается элемент ListView, который выводит все добавленные объекты. Для упрощения примера добавление и вывод происходят на одной и той же страницу.

Также изменим код C# у страницы, чтобы добавлять объекты Phone и выводить их в списке:

|  |
| --- |
| C# |
| using System;  using System.Collections.Generic;  using System.Linq;  using Windows.UI.Xaml;  using Windows.UI.Xaml.Controls;  using System.Collections.ObjectModel;  using Microsoft.EntityFrameworkCore;    namespace SQLiteApp  {      public sealed partial class PhonesListPage : Page      {          ObservableCollection<Phone> phones;          List<Company> companies;          public PhonesListPage()          {              this.InitializeComponent();                this.Loaded += PhonesListPage\_Loaded;          }            private void PhonesListPage\_Loaded(object sender, RoutedEventArgs e)          {              using (MobileContext db = new MobileContext())              {                  phones = new ObservableCollection<Phone>(db.Phones.Include(x => x.Company).ToList());                  companies = db.Companies.ToList();              }                companiesList.ItemsSource = companies;              phonesList.ItemsSource = phones;          }            private void Back(object sender, RoutedEventArgs e)          {              if (Frame.CanGoBack) Frame.GoBack();          }            private void Add\_Click(object sender, RoutedEventArgs e)          {              Company company = companiesList.SelectedItem as Company;              if (company == null) return;                // создаем объект Phone              Phone phone = new Phone              {                  Title = nameBox.Text,                  Price = Int32.Parse(priceBox.Text),                  Company = company              };                using(MobileContext db = new MobileContext())              {                  db.Companies.Attach(company);                  db.Phones.Add(phone);                  if(db.SaveChanges() > 0)                  {                      phones.Add(phone);                      // либо так                      //phonesList.ItemsSource = db.Phones.Include(x=>x.Company).ToList();                  }              }          }      }  } |

При загрузке страницы мы получаем два списка - список телефонов для их отображения в ListView и список компаний для выпадающего списка ComboBox, чтобы при добавлении пользователь мог выбрать компанию.

В обработчике кнопки добавления создаем новый объект Phone по введенным в поля данным и добавляем его в базу данных. В общем-то тут добавление проходит также, как и добавление обычной модели. Только в данном случае нам надо еще установить внешний ключ на связанную модель Company с помощью свойства CompanyId.

В итоге страница будет выглядеть следующим образом:

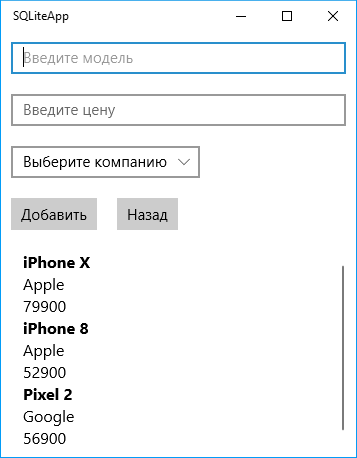


Рисунок 14. Работа программы

И чтобы эту страницу можно было вызвать с главной, добавим в MainPage.xaml еще одну кнопку:

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | XML | | <Page.BottomAppBar>      <CommandBar>          <AppBarButton x:Name="phonesListButton" Label="Телефоны" Icon="PhoneBook" Click="PhonesList\_Click" />            <!-- остальные кнопки-->      </CommandBar>  </Page.BottomAppBar> | |  |

И определим для этой кнопки обработчик:

|  |
| --- |
| XML |
| private void PhonesList\_Click(object sender, RoutedEventArgs e)  {      Frame.Navigate(typeof(PhonesListPage));  } |

# ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. Создайте программу с базой данной и интерфейсом на UWP по заданию
2. Спроектируйте блок схему к программе
3. Отобразить в отчете проделанные шаги
4. Ответить на контрольные вопросы
5. Написать вывод о проделанной работе

# ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

|  |
| --- |
| Вариант 1 – Реализуйте программу автомастерская. Создайте БД по теме автомастерская, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 2 – Реализуйте программу авиабилеты. Создайте БД по теме авиабилеты, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 3 – Реализуйте программу автомастерская. Создайте БД по теме автомастерская, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 4 - Реализуйте программу растения. Создайте БД по теме садоводство, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 5 – Реализуйте программу склад. Создайте БД по теме склад товаров, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 6 - Реализуйте программу сведенья о студентах. Создайте БД по теме студенты и сведенья о них, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 7 - Реализуйте программу клиентская база. Создайте БД по теме клиентская база, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 8 - Реализуйте программу сведенья об учебный организациях. Создайте БД по теме сведенья об учебный организациях, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 9 - Реализуйте программу изделия для производства. Создайте БД по теме изделия для производства, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |
| Вариант 10 - Реализуйте программу реставрация мебели. Создайте БД по теме реставрация мебели, данные и параметры используете по усмотрению. Задача: создайте интерфейс, который отображает данные из БД, может редактировать ее, удалять и добавлять данные в нее. |

# КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как реализовывали задание?
2. Какие есть плюсы у SQLite?
3. Какие есть операции с БД?

# РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1. Официальная документация Microsoft C# — Текст: электронный // Microsoft [сайт]. — URL: https://docs.microsoft.com/ (дата обращения: 12.03.2020).
2. Сообщество IT-специалистов — Текст: электронный // Habr [сайт]. — URL: https://habr.com/ (дата обращения: 12.03.2020).
3. Сайт о программирование / — Текст: электронный // Metanit [сайт]. — https://metanit.com/ (дата обращения: 12.03.2020).
4. С# Tutotial — Текст: электронный // Tutotial [сайт]. — URL: https://csharp.net-tutorials.com/ (дата обращения: 13.03.2020).

# ПРИЛОЖЕНИЕ А

**Требования к отчету**

Общие требования:

1. Шрифт – Times New Roman, 14 пт.
2. Интервалы: междустрочный – 1,5 строки, интервал до и после абзаца – 0 пт.
3. Отступ первой строки – 1,25
4. Рисунки и подписи к ним выравниваются по центру.

Требования к структуре отчета:

1. Титульный лист
2. Содержание
3. Основная часть:
   1. Задачи
   2. Код
   3. Блок-схемы
   4. Результат работы
4. Выводы по работе
5. Ответы на контрольные вопросы

# ПРИЛОЖЕНИЕ Б

ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Отчет по учебной-практической работе №10

учебной практике   
УП 05. Программирование

Тема: «UWP работа с базой данных»

Выполнил:

студент \_\_\_\_\_\_\_ группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:  
преподаватель

Владимирова А.В.

г. Томск – 2020 г