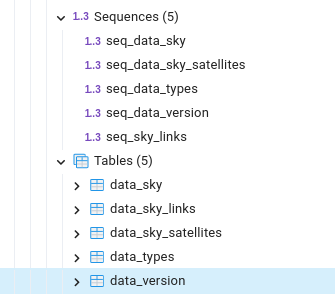
|  |  |
| --- | --- |
| 7 | Совместное проектирование структуры базы данных. Каждый слушатель курсов самостоятельно проектирует собственную базу данных. |
| 8 | Создание подключения к базе данных с помощью технологии ADO.net. Построение системы чтения данных с использованием пуля подключений. Построение системы записи данных с использованием Reflection. |
| 9 | Разработка полной загрузки данных (два сценария). Создание простого ASP.net приложения. Подключение таймера для загрузки и обработки данных. |
| 10 | Создание простого контроллера для получения сферических координат. |

|  |
| --- |
| **Совместное проектирование структуры базы данных. Каждый слушатель курсов самостоятельно проектирует собственную базу данных.** |

Задание: Спректировать собственную базу данных для хранения информации по следующим моделям: GpsdVersionModel, GpsdSkyModel, GpsdSkySatelliteModel

Создадим **5 таблиц** и перечислений. Так же, свяжем каждую таблицу с перечислением по ключу id

Получим:  
  


|  |  |
| --- | --- |
| Таблица | Назначение |
| data\_types | Тип данных, справочник, который мы будем заполнять самостоятельно. В нем есть поле Class – оно уникально. |
| data\_version | Данные, которые будут создаваться из модели GpsdVersionModel |
| data\_sky | Данные, которые будут создаваться из модели GpsdSkyModel |
| data\_sky\_satellites | Табличные данные из модели GpsdSkySatelliteModel |
| data\_sky\_links | Таблица – связка между моделью GpsdSkyModel и GpsdSkySatelliteModel |

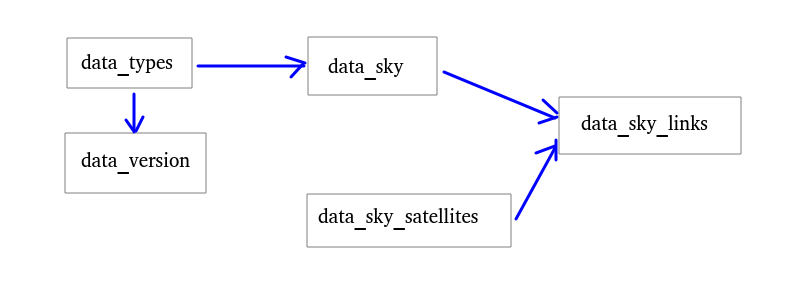
Создадим связи между таблицами. Сделаем связь между таблицей data\_types

|  |
| --- |
| alter table public.data\_sky add constraint fk\_data\_sky\_data\_type  FOREIGN KEY (type\_id) REFERENCES data\_types (id);  alter table public.data\_version add constraint fk\_data\_sky\_satellites\_data\_type  FOREIGN KEY (type\_id) REFERENCES data\_types (id); |

Затем, связываем спутники

|  |
| --- |
| alter table public.data\_sky\_links add constraint fk\_data\_sky\_links\_sky\_id  FOREIGN KEY (sky\_id) REFERENCES data\_sky (id);  alter table public.data\_sky\_links add constraint fk\_data\_sky\_links\_satellite\_id  FOREIGN KEY (satellite\_id) REFERENCES data\_sky\_links (id); |

В итоге, мы получаем следующие связи



Так же, заполняем справочник

|  |
| --- |
| insert into data\_types(class,description) values('SKY', 'Информация по местоположении');  insert into data\_types(class,description) values('VERSION', 'Версия протокола'); |

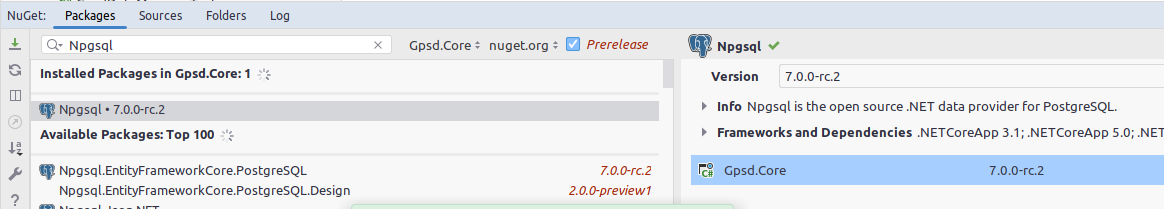
Задание:  
Написать SQL скрипт для записи данных в таблицы:  
data\_version, data\_sky, data\_sky\_satellites, data\_sky\_links

|  |
| --- |
| **Создание подключения к базе данных с помощью технологии ADO.net. Построение системы чтения данных с использованием пула подключений. Построение системы записи данных с использованием Reflection.** |

Для подключения к базе данных, создадим специальный класс с описанием параметров подключения. Далее, переопределим метод ToString

|  |
| --- |
| https://www.connectionstrings.com/ |
| /// <summary> /// Класс - описание настроек подключение к базе данных PostgreSql /// </summary> public class PostgreConnect {    #region Свойства   /// <summary>  /// Хост  /// </summary>  public string Host { get; set; } = "localhost";   /// <summary>  /// Порт подключения  /// </summary>  public long Port { get; set; } = 5432;   /// <summary>  /// Имя пользователя  /// </summary>  public string Login { get; set; } = "gpsd";   /// <summary>  /// Пароль  /// </summary>  public string Password { get; set; } = "123456";   /// <summary>  /// Наименование базы данных  /// </summary>  public string Database { get; set; } = "gpsd";   #endregion   public override string ToString()  => string.Format(  "User ID={0};Password={1};Host={2};Port={3};Database={4};Pooling=true;",  Login, Password, Host, Port, Database); } |

Подключим Nuget пакет Ngpsql (https://www.npgsql.org/)



Далее, добавим в проект статический класс расширение. В котором реализуем два статических метода: выборка данных, создание параметров для выборки данных

|  |
| --- |
| public static class PostgreConnectExtension {  /// <summary>  /// Получить выборку данных из PostgreSql  /// </summary>  /// <param name="connect"> Информация о соединении </param>  /// <param name="sql"> SQL запрос </param>  /// <param name="parameters"> Набор параметров </param>  /// <returns></returns>  /// <exception cref="ArgumentException"></exception>  /// <exception cref="InvalidOperationException"></exception>  public static DataTable GetTable(this PostgreConnect connect, string sql, Dictionary<string, object>? parameters = null)  {  ArgumentNullException.ThrowIfNull(connect);  if (string.IsNullOrEmpty(sql))  throw new ArgumentException("Не указан SQL запрос для выполнения выборки данных");   try  {  using var database = new NpgsqlConnection(connect.ToString());  database.Open();  var command = new NpgsqlCommand(sql, database);  command.Parameters.AddRange(parameters.GetPostgreParameters().ToArray());   using var reader = command.ExecuteReader(CommandBehavior.CloseConnection);  var result = new DataTable();  result.Load(reader);  return result;  }   catch (Exception ex)  {  throw new InvalidOperationException(  $"Ошибка при выполнении Sql запроса! \n{sql}. Соединение: {connect.ToString()}", ex);  }   }   /// <summary>  /// Сформировать набор параметров для SQL запроса  /// </summary>  /// <param name="parameters"> Исходный словарь с параметрами </param>  /// <returns></returns>  private static IEnumerable<NpgsqlParameter> GetPostgreParameters(this Dictionary<string, object>? parameters)  {  if (parameters is null) return Enumerable.Empty<NpgsqlParameter>();  var result = parameters.Select(x => new NpgsqlParameter  {  ParameterName = x.Key, Value = x.Value  });  return result;  } |

Для проверки работы с базой данных, добавим новый Nunit тест с двумя тестами: обычная выборка данных, выборка данных с параметрами.

|  |
| --- |
| public class PostgreTests {  /// <summary>  /// Проверить подключение и выборку данных  /// </summary>  [Test]  public void Check\_Connect\_DataBase()  {  // Подготовка  var connect = new PostgreConnect();  var sql = "select \* from data\_types";   // Действие  var result = connect.GetTable(sql);   // Проверки  Assert.IsNotNull(result);  }   /// <summary>  /// Проверить подключение и выборку данных с использованием параметров  /// </summary>  [Test]  public void Check\_Connect\_Database\_WithParameters()  {  // Подготовка  var connect = new PostgreConnect();  var sql = "select \* from data\_types where id = @Id" ;  var parameters = new Dictionary<string, object?>()  {  { "Id", 1 }  };   // Действие  var result = connect.GetTable(sql, parameters!);   // Проверки  Assert.IsNotNull(result);  Assert.That(result.Rows.Count, Is.EqualTo(1));  } |

Задание.

По аналогии, добавить в класс расширение вызовы для вставки и обновления данных в PostgreSql

Свяжем модели данных с базой данных путем добавления поля – уникальный код записи. Для этого мы созданим специальный интерфейс IdataEntity и абстрактны класс DataEntity

|  |
| --- |
| /// <summary> /// Специальный интерфейс для связывания произвольной модели в базой данных /// </summary> public interface IDataEntity {  public long Id { get; set; } } |
| /// <summary> /// Абстрактный класс для наследования. Указывает уникальный код записи в базе данных /// </summary> public abstract class DataEntity: IDataEntity {  [JsonIgnore]  public long Id { get; set; } } |

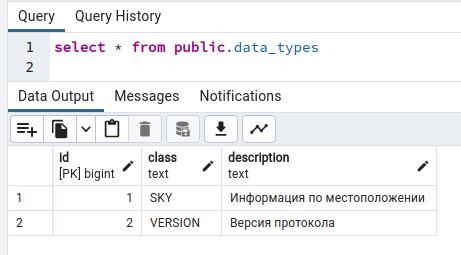
Так же, создадим собственный атрибут для создания описаний для каждой модели. Данный атрибут будет в себе хранить наименование таблицы в базе данных.

|  |
| --- |
| /// <summary> /// Класс атрибут для хранения информации о наименовании таблицы базы данных /// </summary> public class TableNameAttribute: Attribute {  private string \_tableName = string.Empty;   public TableNameAttribute(string tableName)  {  if (string.IsNullOrEmpty(tableName))  throw new ArgumentException("Некорректно переданы параметры!", nameof(tableName));    \_tableName = tableName;  } } |
| [TableName("data\_sky")] public class GpsdSkyModel: GpsdDataModel |

Задание:  
Для моделей: GpsdVersionModel, GpsdSkySatelliteModel в коде указать наименование таблиц.

Вставка данных:

В абстрактный класс добавим свойство, которое будет отвечать за передачу данных о типе модели. В базе данных – это поле type\_id



|  |
| --- |
| /// <summary> /// Специальный код с типом данных /// </summary> public virtual long DataType => 0; |

Далее, для классов с моделями, переопределим данное свойство, согласно записям таблицы data\_types. Например, для класса GpsdDataModel

|  |
| --- |
| [JsonPropertyName("type\_id")] public override long DataType => 2; |

Теперь, доработаем расширение PostgreConnectExtension. Добавим метод InsertData

|  |
| --- |
| /// <summary> /// Вставить данные в таблицу из модели <see cref="IGpsDataModel"/> /// </summary> /// <param name="connect"></param> /// <param name="model"> Исходная модель </param> public static long InsertData(this PostgreConnect connect, IGpsDataModel model) {  ArgumentNullException.ThrowIfNull(connect);  ArgumentNullException.ThrowIfNull(model);   var tableAttributes = model.GetType().GetCustomAttribute(typeof(TableNameAttribute)) as TableNameAttribute;  if (tableAttributes is null)  throw new InvalidCastException();   // Наименование таблицы  var tableName = tableAttributes.TableName;    // Собираем поля для вставвки  var fields = model.GetType().GetProperties()  .Select(x => new  {  Name = x.Name,  JsonName = x.GetCustomAttribute(typeof(JsonPropertyNameAttribute)) as JsonPropertyNameAttribute,  Value = x.GetValue(model)  }).Where(x => x.JsonName is not null) .ToList();   var fieldsNames = fields.Select(x => x.JsonName?.Name ?? "").ToArray();  var fieldsValues = fields.Select(x => $"'{x.Value}'").ToArray();  var sql =  $"insert into {tableName} ({string.Join(",", fieldsNames)}) values({string.Join(",", fieldsValues)});";    // Выполним запрос и получим код вставки  try  {  using var database = new NpgsqlConnection(connect.ToString());  database.Open();  var command = new NpgsqlCommand(sql, database);   var result = command.ExecuteNonQuery();  return result;  }   catch (Exception ex)  {  throw new InvalidOperationException(  $"Ошибка при выполнении Sql запроса! \n{sql}. Соединение: {connect.ToString()}", ex);  }  } |

Данный метод формирует Sql запрос на вставку данных.  
Пример

|  |
| --- |
| insert into data\_version (release,rev,proto\_major,proto\_minor,type\_id) values('1','1','1','1','2'); |

Так же, для проверки добавим автоматический тест. Выполним его и убедимся в том, что вставка данных выполнена.

|  |
| --- |
| /// <summary> /// Проверка вставки данных /// </summary> [Test] public void Check\_Connect\_InsertData() {  // Подготовка  var connect = new PostgreConnect();  var model = new GpsdVersionModel()  {  Release = "1", Class = "Version", Major = 1, Minor = 1, Revision = "1"  };    // Действие  var result = connect.InsertData(model);    // Проверки  Assert.IsTrue(result > 0); } |

Задание. Написать самостоятельно автоматический тест. Выполнить его. Далее, сделать тест на выборку данных. Получить результат вставки.