**Модульное тестирование.**

Как ранее было сказано, основной задачей модульного тестирования является проверка корректности работы отдельных методов (модулей). Данные методы не должны иметь каких-либо зависимостей с другими методами, а также непосредственно с базой данных.

**Написание тестовых сценариев.**

Класс, содержащий тестовые сценарии, помечается атрибутом *[TestFixture]*. Данный атрибут отмечает то, что следующий класс содержит тесты:

[TestFixture]

public class UnitTestDataBase

{

}

Попробуем написать первый тестовый сценарий. У нас имеется метод, который принимает два параметра: table и name, а возвращает строку string:

public string QueryPassengerOrCompany(string table, string name) =>

$"insert into {table} (name) values('{name}')";

Напишем для него тестовый сценарий. Перед тестом мы помечаем его атрибутом *[Test]*. Для проверки работы метода нам необходимо инициализировать тестируемый класс, а также создать тестовые данные, т.е. *table* и *name*:

[Test]

public void TestQueryAdd()

{

    DataBase db = new DataBase();

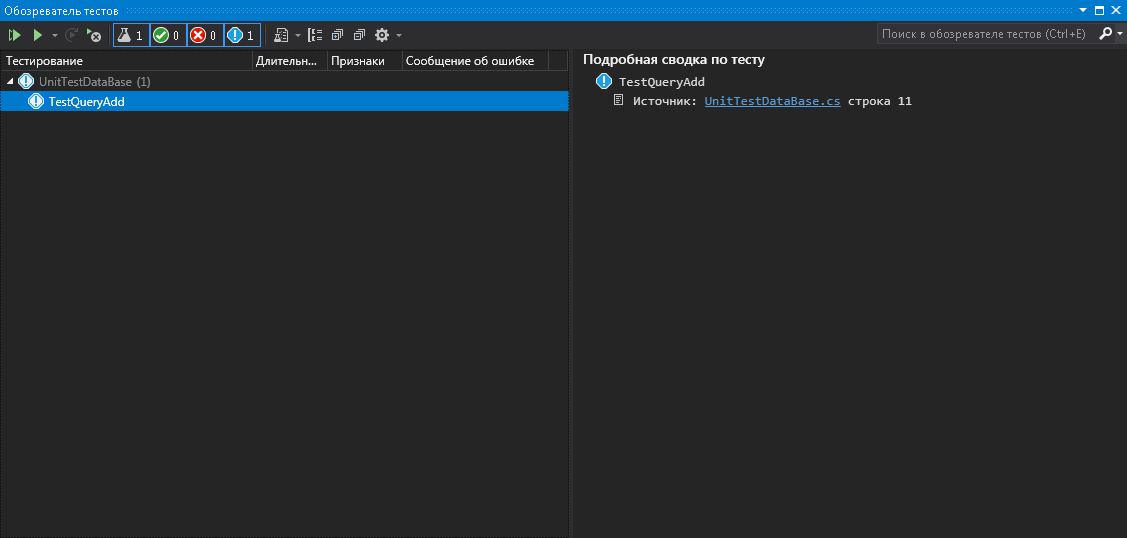
    string table = "Passenger", name = "Russkih V.D.";

    Assert.AreEqual($"insert into {table} (name) values('{name}')",

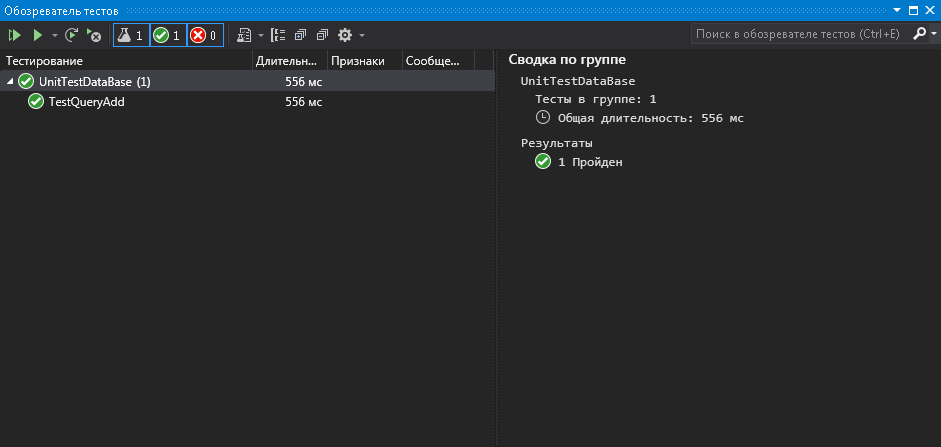
        db.QueryPassengerOrCompany(table, name));

}

Проверка корректности работы тестируемого модуля происходит при помощи метода *AreEqual* класса *Assert*. Если значения равны, значит тест выполнился успешно, иначе *Failed.*

Для выполнения теста, необходимо открыть обозреватель тестов: *Тест* → *Обозреватель* *тестов* или сочетанием клавиш *ctrl+E,T*.

Выполнив тест, увидим следующий результат:

Но это лишь один тест. Чтобы метод действительно был корректным и работал, как нужно, необходимо провести тесты с разными исходными данными. Но чтобы не писать каждый тест с разными данными, или не вызывать несколько раз *Assert.AreEqual(…)*, мы можем воспользоваться аттрибутом *[TestCase()]*. Данный атрибут позволяет выполнить один и тот же тест несколько раз, с разными данными. Пример написания такого тестового сценария:

[TestCase("Passenger", "Kireev D.A.")]

[TestCase("Company", "Kireev D.A.")]

public void TestQueryAdd(string table, string name)

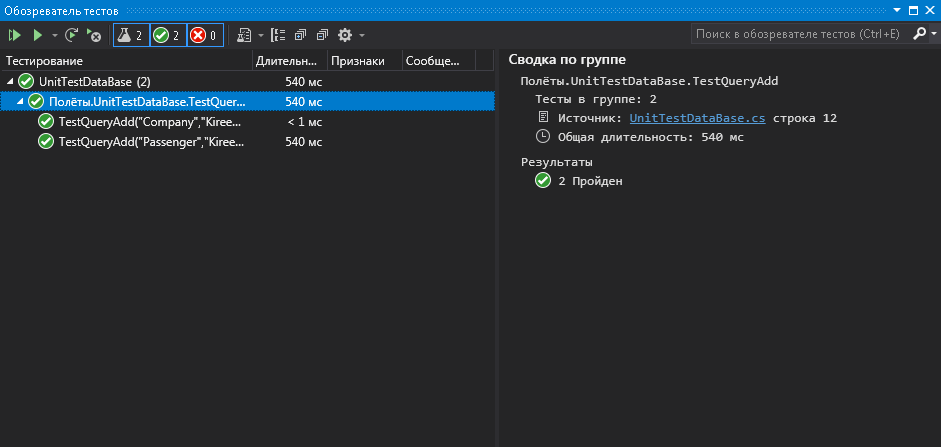
{

    DataBase db = new DataBase();

    Assert.AreEqual($"insert into {table} (name) values('{name}')",

        db.QueryPassengerOrCompany(table, name));

}

Чтобы использовать данные из *TestCase(),* необходимо описать входные данные функции. В нашем случае это *string table* и *string name*. Выполним тест:

Создадим ещё один *Test*, только теперь входными данными будут *null.* Метод должен вернуть значение *null*. Заметим, что нам снова приходиться объявлять экземпляр класса *DataBase*. Для такого мы можем объявить метод *SetUp()* с атрибутом *[SetUp]*. Данный атрибут говорит о том, что этот метод будет выполняться перед выполнением каждого тестового сценария. Опишем этот метод, также объявим в классе поле класса *DataBase*, чтобы оно было доступно в каждом тестовом сценарии:

DataBase db;

[SetUp]

public void InitDataBase()

{

    db = new DataBase();

}

Вернёмся к нашему тесту со значениями *null*.

[TestCase(null, null)]

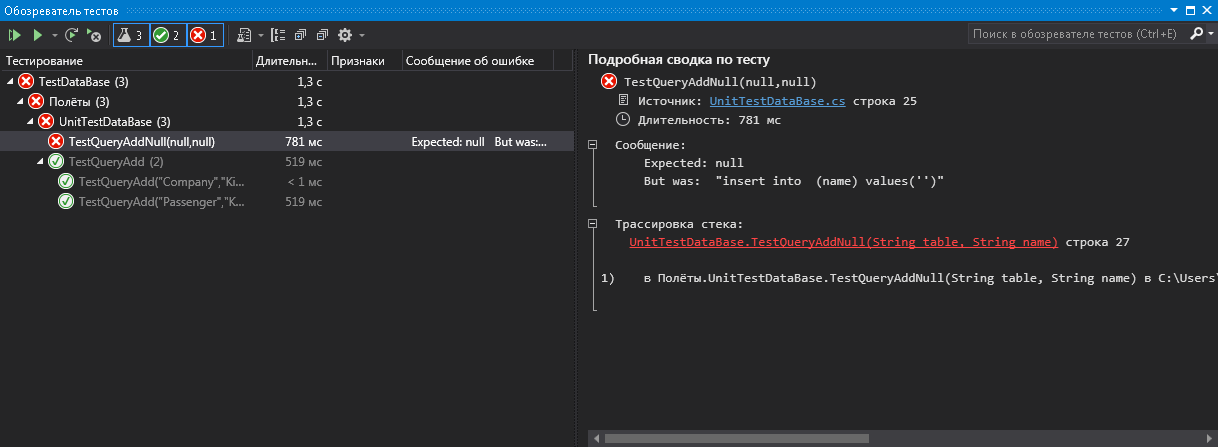
public void TestQueryAddNull(string table, string name)

{

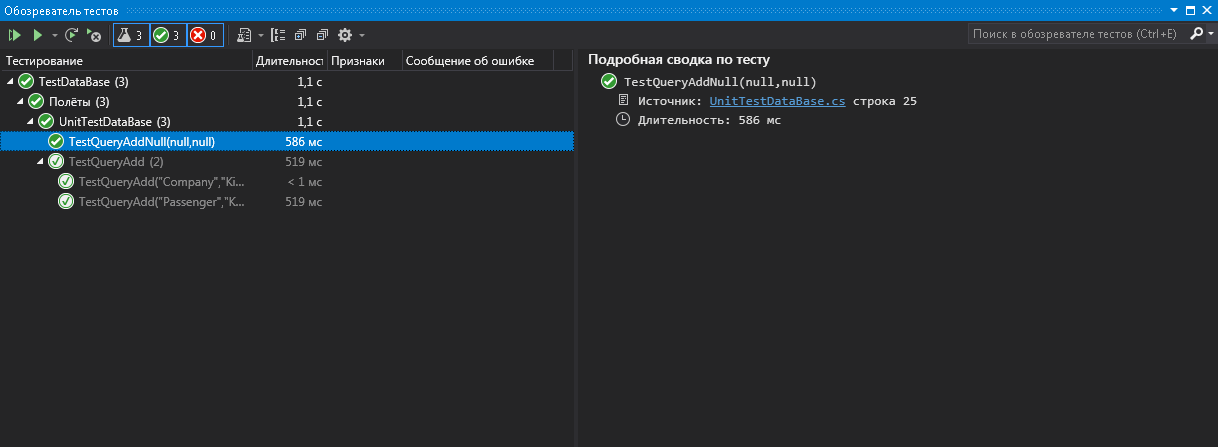
    Assert.IsNull(db.QueryPassengerOrCompany(table, name));

}

Попробуем выполнить.

Тест завершился ошибкой. Подробности об ошибке мы можем увидеть в подробной сводке по тесту, она находится в обозревателе тестов. Попробуем исправить ошибку и выполнить тесты.

public string QueryPassengerOrCompany(string table, string name) =>

    (name == null) ? null : $"insert into {table} (name) values('{name}')";

Класс Assert содержит ещё множество других функций, такие как *.IsFalse*, *.IsNull*, *.Throws* и т.д. Метод *.Throws* позволяет обрабатывать исключения. Метод имеет следующую конструкцию:

*Assert.Throws(new <Тип исключения>, new TestDelegate(<Метод, который вызовет исключение>))*.