Темы урока

[**Подготовка: Создание БД OnlineStore**](#_r8dwlw946bhp) **1**

[**Теория и практика**](#_dblr7fhd1skw) **1**

[Создаём новое консольное приложение](#_92x4sl72w0va) 2

[SqlConnection](#_sosw12qdf9ut) 2

[SQL connection string](#_eg05pdt1gxwy) 2

[SqlCommand (CommandType = Text)](#_qgbgegz3tmd6) 2

[SqlDataReader](#_pvq11ke6ewa4) 2

[Небольшой рефакторинг](#_o47f6km2o705) 3

[Самостоятельная работа #1](#_ngqzojtv7vvf) 3

[SqlCommand (CommandType = StoredProcedure)](#_e3hzhu31vpau) 3

[Самостоятельная работа #2](#_7fo6zr73t71d) 4

[SqlTransaction (если есть время и “тянут” :)](#_1e9vgg90d7nf) 4

[Problem Statement](#_pvz86qjltizs) 4

[Solution](#_472pq6a03ht2) 4

[Implementation](#_gnkjgn5ulq61) 5

[**Домашнее задание**](#_xliovyhyekrq) **5**

На этом уроке мы учимся работать с базой данных программно — на языке C# используя классы .NET.

Работать будем с готовой БД OnlineStore, которую мы создадим в самом начале на базе БД прошлого урока (скрипты лежат в папке OnlineStore\_SQL\_scripts). По сравнению с прошлым уроком здесь есть внешние ключи, чтобы в самом конце занятия, если останется время, показать как программно работать с транзакциями.

# Подготовка: Создание БД OnlineStore

Создаём новую БД с названием OnlineStore

1. Накатываем скрипт со схемой **L30\_C01\_OnlineStore.Schema.sql**
2. Накатываем скрипт с данными **L30\_C02\_OnlineStore.Data.sql**Можно вскользь заметить, что в скрипте на данные мы, чтобы вызвать TRUNCATE TABLE убиваем внешние ключи, потом (после вызова TRUNCATE TABLE) пересоздаём их (как и в схеме). По идее, можно было бы не очищать таблицы, но тогда наш скрипт перестанет быть запускабельным несколько раз, я осознанно решил расплатится за такое его качетво дублированием кода по созданию внешних ключей.
3. (Скрипт с хранимками не трогаем, лучше даже его не слать им, первую написать онлайн, вторую и третью пусть напишут сами в рамках самостоятельной работы)

# Теория и практика

Как обычно, работаем в проекте **L30\_C01\_working\_with\_sql\_db** постепенно приводя его к **L30\_C02\_working\_with\_sql\_db\_final**.

## Создаём новое консольное приложение

Создаём новое приложение

Пространство имён System.Data.SqlClient. Для этого устанавливаем NuGet-пакет System.Data.SqlClient.

## SqlConnection

Немного теории по слайду.

Подчеркнуть, что

* Класс реализует интерфейс IDisposable и лучшей практикой использования является конструкция:  
    
  using (var connection = new SqlConnection(connectionString)) { … }
* Нужно не забывать открывать соединение методом Open()

## SQL connection string

Немного теории по слайду.

Создаём класс OnlineStoreRepository со строкой подключения в конструкторе.

Создаём метод private SqlConnection GetOpenedSqlConnection() { … }.

## SqlCommand (CommandType = Text)

Ещё немного теории по слайду.

Создаём метод public int GetProductCount() { … }.

Рассказываем про метод SqlCommand.ExecuteScalar().

## SqlDataReader

Ещё немного теории по слайду.

Создаём метод public List<string> GetProductList().

Говорим, что здесь конечно должен быть лист dto-объектов по-хорошему, но для них надо писать классы, так что мы не отвлекаемся и пока тупо всё упаковываем в строку.

Рассказываем про члены SqlDataReader:

* Свойство HasRows
* Метод Read() и практику его использования в while(sqlReader.Read()) { … }
* Метод SqlDataReader.GetOrdinal(string)
* Методы GetInt32(int), GetString(int), GetDecimal(int), и т.д.

## Небольшой рефакторинг

Немного реорганизуем наш код.

Выделяем интерфейс: IProductRepository с двумя нашими методами:

* int GetProductCount();
* List<string> GetProductList();

Разбиваем класс OnlineStoreRepository на 2 файла, класс становится public partial class:

* Файл OnlineStoreRepository.cs
  + Объявление реализации интерфейса остаётся здесь, хотя реализовывать интерфейс будем во втором файле.
* Файл OnlineStoreRepository.Product.cs
  + Здесь два наших метода, реализующих интерфейс IProductRepository.

## Самостоятельная работа #1

Создать новый интерфейс IOrderRepository с двумя методами:

* int GetOrderCount();
* List<string> GetOrderIdList();

Расширить класс OnlineStoreRepository созданным интерфейсом и реализовать его в отдельном файле OnlineStoreRepository.Order.cs.

## SqlCommand (CommandType = StoredProcedure)

Теория по слайду.

Рассказать, что параметры это круто, даже и для обычных команд, не только для хранимок.  
Показать как можно дропнуть базу через SQL-инъекцию, это может их впечатлить.

Пишем хранимую процедуру:

CREATE PROCEDURE dbo.AddProduct(

@name AS VARCHAR(300),

@price AS SMALLMONEY,

@id AS INT OUTPUT)

AS

BEGIN

INSERT INTO [dbo].[Product] ([Name], [Price]) VALUES (@name, @price);

SELECT @id = SCOPE\_IDENTITY();

END

GO

Добавляем в интерфейс OnlineStoreRepository новый декларацию нового метода AddProduct(string name, decimal price).

Показываем работу с хранимыми процедурами, обычными и output-параметрами на примере реализации этого метода.

## Самостоятельная работа #2

Написать SQL-запросы для создания двух хранимых процедур:

* dbo.AddOrder(  
   @customerId AS INT,  
   @orderDate AS DATETIMEOFFSET,  
   @discount AS FLOAT = NULL,  
   @id AS INT OUTPUT)
* dbo.AddOrderItem(  
   @orderId AS INT,  
   @productId AS INT,  
   @numberOfItems AS INT)

Расширить интерфейс IOrderRepository новым методом AddOrder:

public interface IOrderRepository

{

...

int AddOrder(

int customerId,

DateTimeOffset orderDate,

float? discount,

List<Tuple<int, int>>

productIdCountList);

}

Реализовать новый метод в классе OnlineStoreRepository.

## SqlTransaction (если есть время и “тянут” :)

На базе их реализации метода AddOrder можно показать как работать с транзакциями через C#.

### Problem Statement

У таблицы позиций (OrderItem) внешний ключ на таблицу заказов (Order).

Это накладывает на нас следующие ограничения: мы сначала должны создать пустой заказ, а только потом будем заполнять записи позиций, связанных с этим заказом. При этом у позиции (OrderItem) есть внешний ключ на продукт (Product). И если продукта с заданным ID не окажется в базе, то позиция не вставится, вызовется исключение SqlException, **а в базе останется пустой заказ (без привязанных к нему позиций)**.

### Solution

Этого можно избежать, если воспользоваться транзакцией, причем, объединить ей вызов нескольких хранимых процедур.

### Implementation

Показать как это сделано по готовому примеру метода AddOrder в классе OnlineStoreRepository.Order.cs.

# Домашнее задание

Попытаться самостоятельно реализовать методы интерфейса IReminderStorage в новой сборке Class Library (.NET Standard) Reminder.Storage.SqlServer.ADO.