

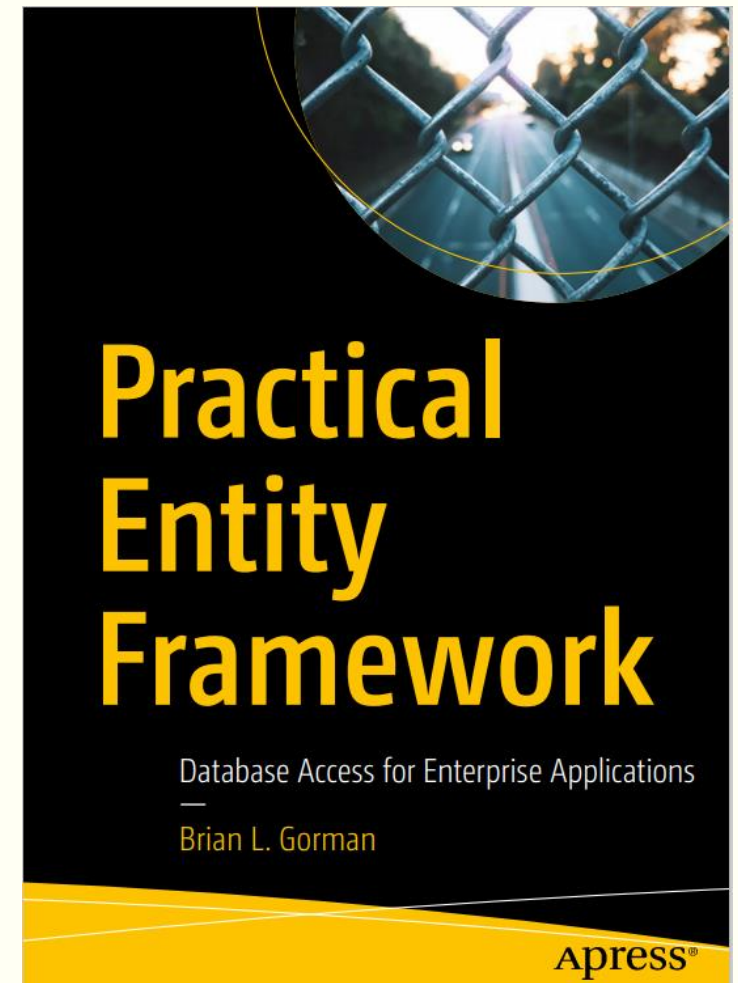


ОСНОВИ РОБОТИ З ТЕХНОЛОГІЄЮ ENTITY FRAMEWORK

Лекція 15
Об'єктно-орієнтоване програмування

План лекції

- Огляд технології Entity Framework. Робота з існуючою БД.
- Підхід Entity Framework: Code First





ОГЛЯД ТЕХНОЛОГІЇ ENTITY FRAMEWORK

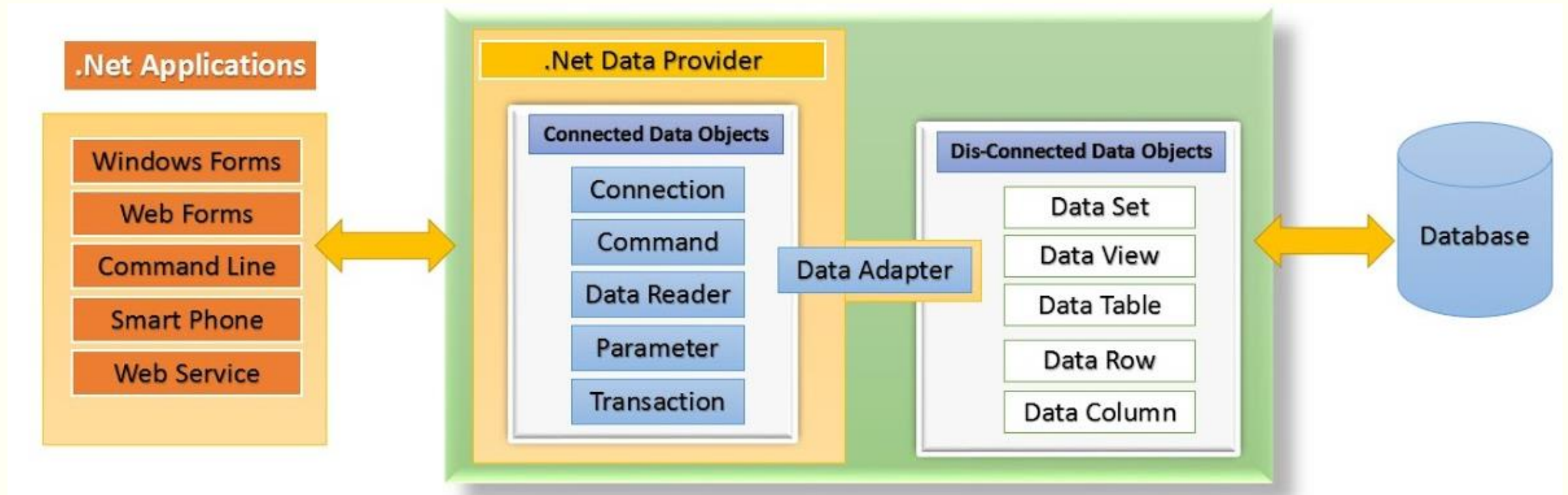
Питання 15.1.

Еволюція взаємодії додатків з базами даних у .NET

- Бази даних Microsoft існували ще до появи .NET.
 - Раніше підключення до БД (database connection) відбувалось у коді за допомогою Object Linking and Embedding Database Object (OLEDB).
 - Розробникам часто доводилось писати вбудовані SQL-запити, а потім підключатись до БД і виконувати дії за допомогою таких інструментів.
 - Крім того, часто запити не мали достатньої організованості чи безпечності: подібні чи однакові виклики могли виконуватись одночасно з багатьох сторінок.
 - Тоді SQL-запити (в найгірших випадках – інформація щодо БД та дані користувачів) могли навіть міститись у HTML-розмітці.
 - Розкриття запитів та даних для підключення (credentials) призводить до величезних безпекових проблем: атаки типу SQL Injection досі популярні.

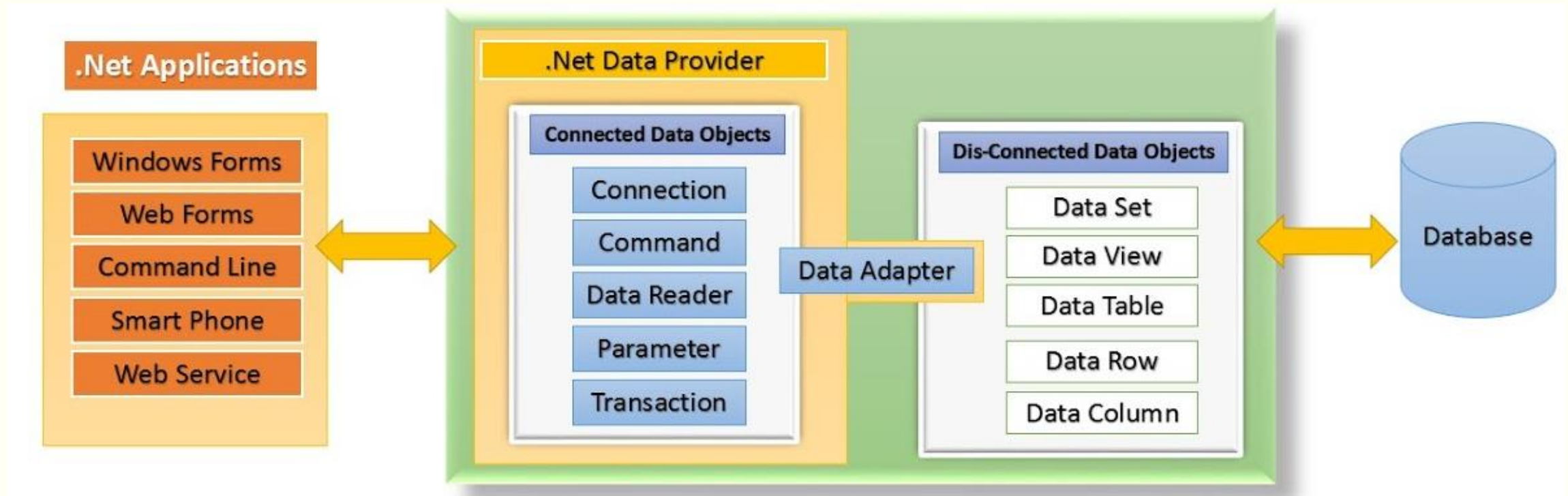
ADO.Net – кращий засіб для взаємодії з БД в додатку

- Для .NET-розробників наступним кроком стало використання технології ADO.Net.
 - Вона розроблялась для попередження деяких вище згаданих проблем.
 - Одна з основних особливостей бібліотеки ADO.Net – проста **параметризація запитів**: створення базового об'єкта для підключення (connection object) з прихованими обліковими даними та рядком підключення, збереженими в спільному безпечному місці.



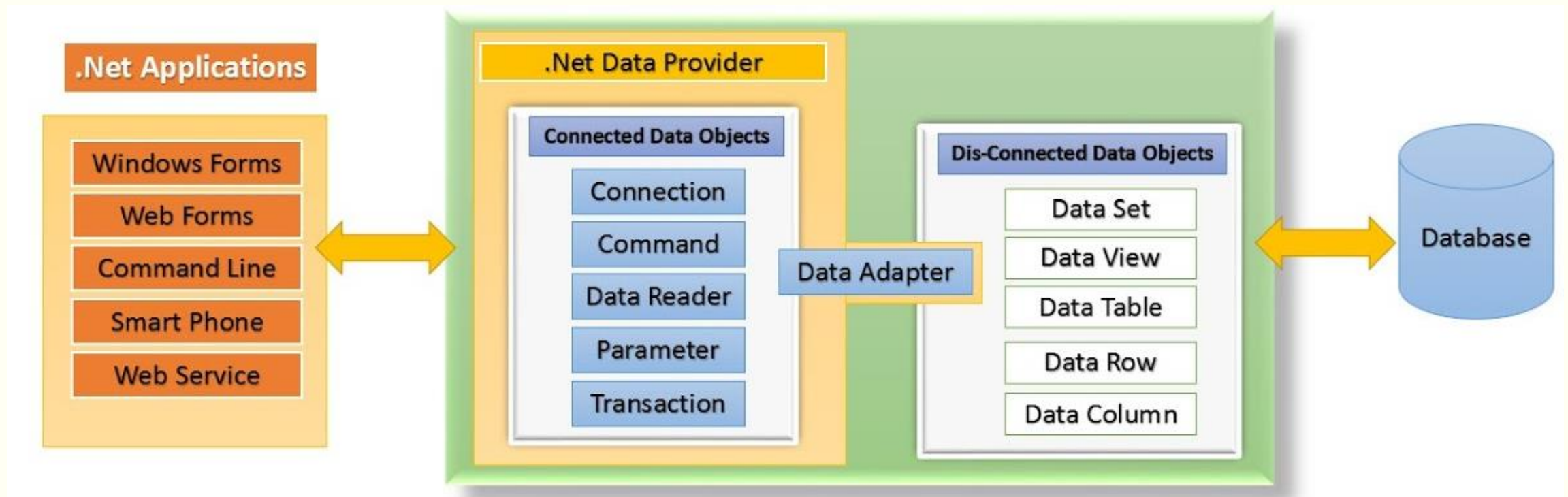
ADO.Net – кращий засіб для взаємодії з БД в додатку

- На об'єкт для підключення (connection object) відбувалось пряме посилення через командний об'єкт (SQL command object).
 - Командний об'єкт SQL мав налаштування, що дозволяли переключати команду на роботу в якості звичайного запиту або на виконання database object, зокрема збереженої процедури.
 - Запит дозволяє параметрам визначатись та обмежуватись своїм типом, а також автоматично замінити некоректні символи, які часто застосовувались в атаках виду SQL Injection.

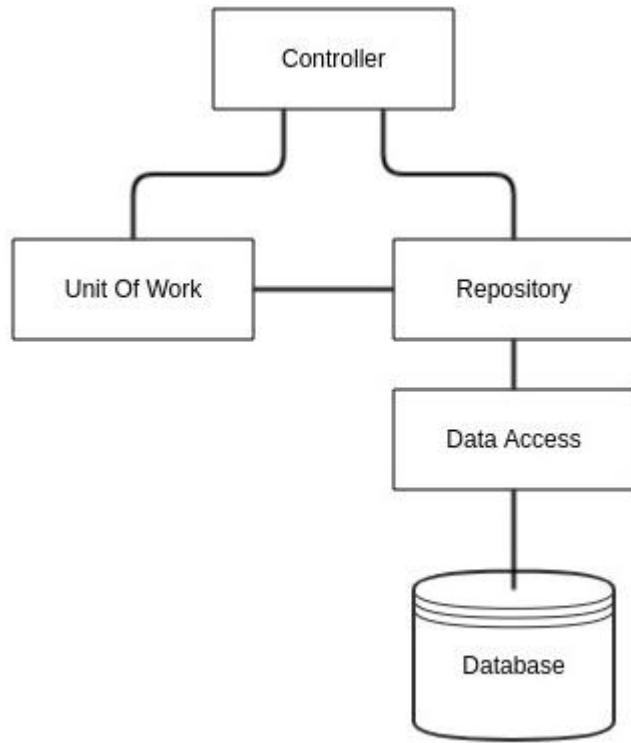


ADO.Net – кращий засіб для взаємодії з БД в додатку

- Як тільки запити виконувались з команди, їх можна було застосовувати для формування результуючого набору даних, зокрема `DataReader` чи `DataSet`.
 - Ці об'єкти потім використовувались з метою отримання релевантних даних та їх відображення кінцевому користувачеві.
 - Цей підхід був найкращий в екосистемі до Entity Framework, NHibernate чи інших ORM.

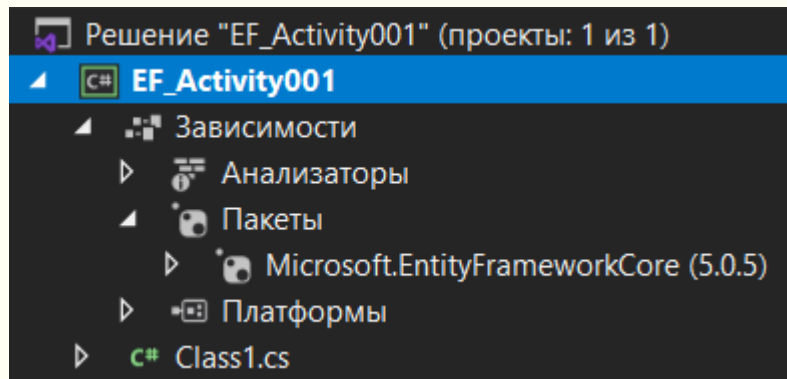
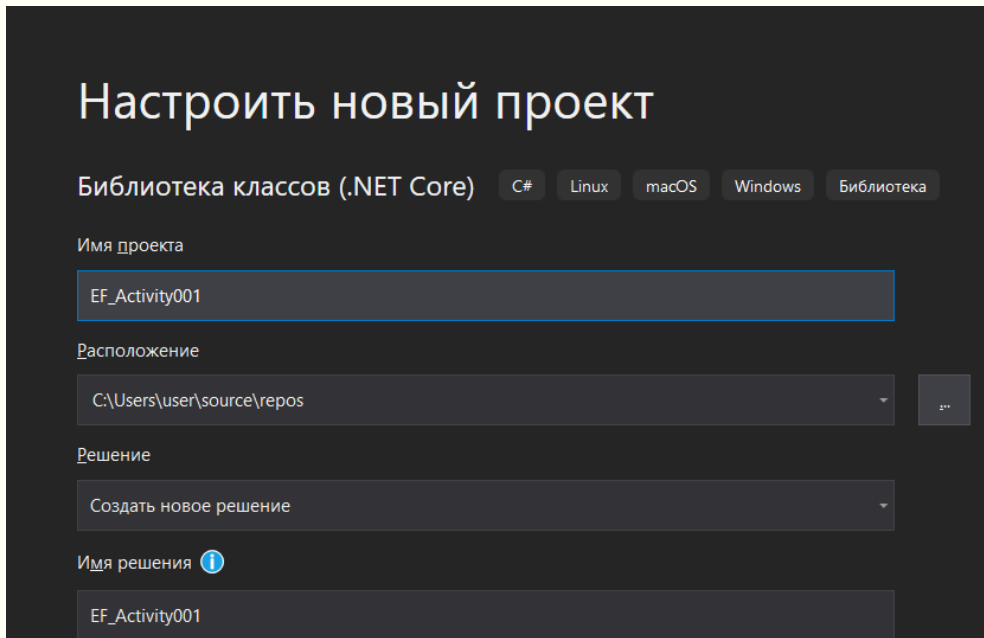


Entity Framework та архітектура додатку з БД

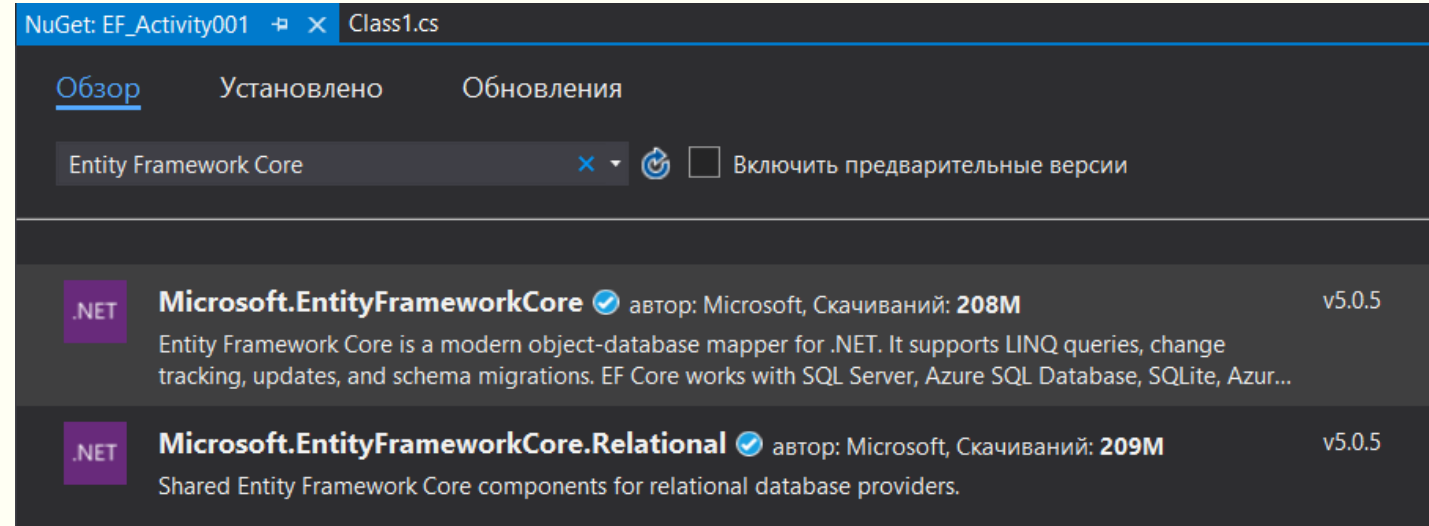


- Розробники практично завжди бажають обгорнути операції з БД згідно з паттерном «Unit of Work», а також надавати доступ до БД через репозиторії (інкапсулює логіку роботи з БД).
 - Паттерн Unit of Work допомагає упростити роботу з різними репозиторіями і дає впевненість, що всі репозиторії будуть використовувати один і той же DbContext.
 - Entity Framework перевершує ADO.Net, надаючи таку обгортку для нас.
- У тандемі, EF та LINQ дозволяють не лише працювати з об'єктами баз даних за допомогою C# чи VB.Net, але й мати змогу визначати структуру БД в коді напряму.
 - Можливість прямого відстежування змін даних з бази в пам'яті сприяє поглибленому вивченню питань конкурентного доступу для роботи з від'єднаними (disconnected) даними.
- Наприкінці 2016 року Microsoft анонсувала .Net з відкритим кодом.
 - Разом з переписуванням .Net Framework на .Net Core прийшла нова версія EF – Entity Framework Core.
 - У найближчий час буде мінімум 2 активні версії Entity Framework: EF6 (орієнтовно до початку 2029 року) та EFCore (версія 3.1 буде підтримуватись орієнтовно до 2030 року).

Розпочинаємо роботу з Entity Framework Core



- Створимо нову бібліотеку класів .NET Core в проєкті EF_Activity001 та підключимо в нього свіжу версію EFCore.



- Наступним кроком буде створення DbContext-об'єкта для роботи з БД.

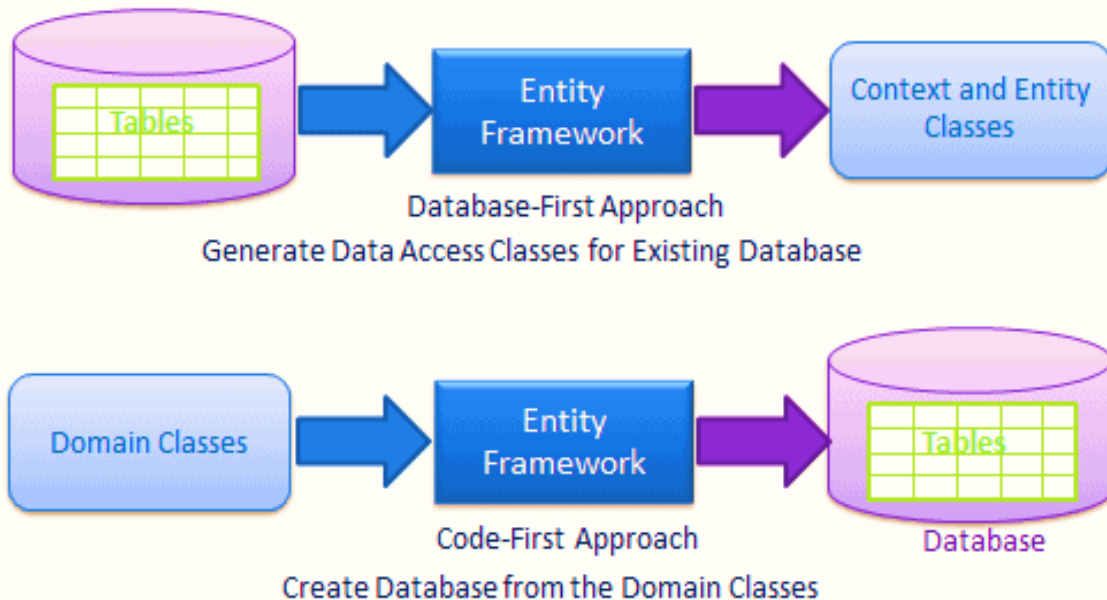
Крок 2. Створення DbContext-об'єкта

- Об'єкт DbContext (контекст БД) працює як інтерпретатор між вашим кодом і реальною БД.
 - У контексті визначаються всі сутності (entity sets), а також можна замістити (override) певну частину схеми БД за допомогою Fluent API.
- Надамо класу змістовну назву, наприклад ApplicationDbContext та будемо успадковувати його від класу DbContext.
 - Також згенеруємо options-конструктор:

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext([NotNullAttribute] DbContextOptions options) : base(options) { }
}
```

- Цей параметр міститиме критичну інформацію, зокрема connection string до БД.

Підходи до інженерії баз даних: database-first vs code-first

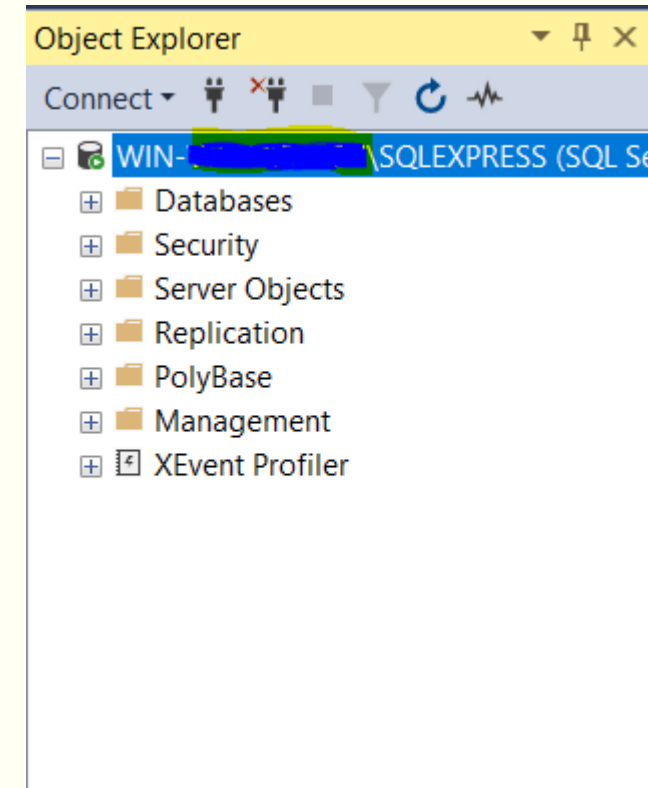
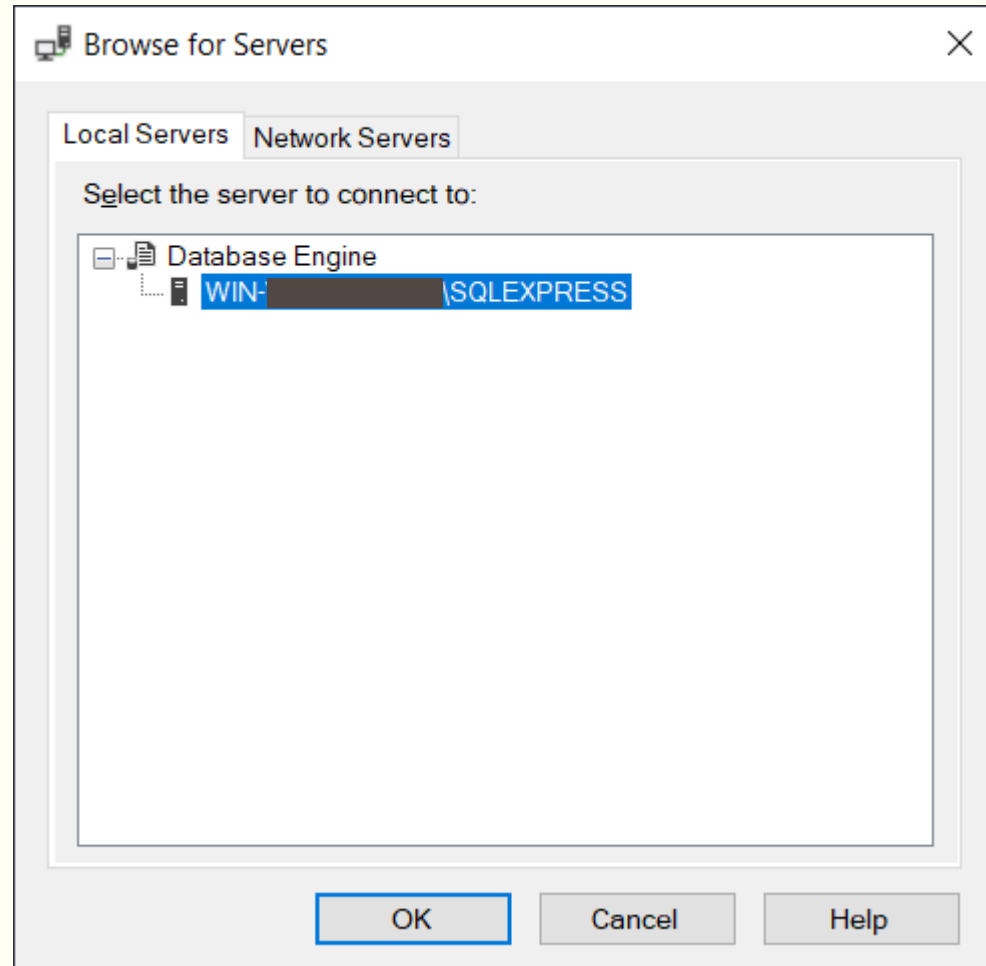
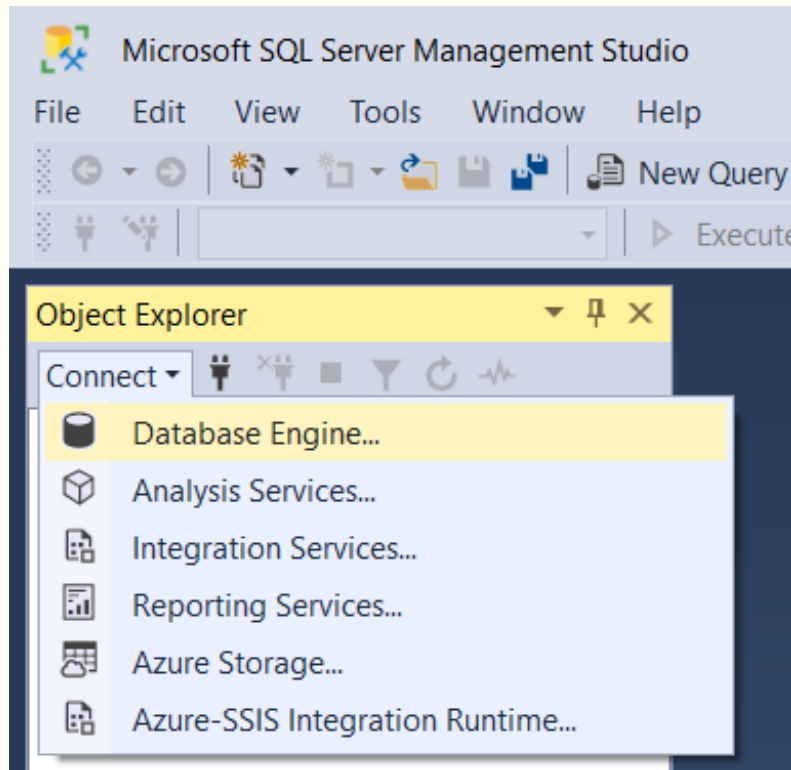


- У Entity Framework підхід Database First надає альтернативу підходу Code First, створюючи РОСО-класи з існуючої БД.
 - Підхід Database first використовується, коли БД вже готова.
 - Змінювати БД можна вручну та оновлювати модель з БД.
 - entity framework може створити класи рівня моделі на основі таблиць та стовпців реляційної БД.
- Підхід code first було представлено в Entity Framework 4.1.
 - Він націлений на БД, якої ще не існує, і Code First API створить її на льоту на основі ваших класів сутностей (entity classes) та налаштувань (configuration).
 - Також можливе оновлення БД, якщо модель змінюється, використовуючи *міграції* (Code First Migrations).
 - Підхід Code First дуже популярний та має, в першу чергу, повний контроль над кодом.
 - Тут можемо виконувати всі операції з БД на стороні коду, зникає ручне внесення змін у базу.

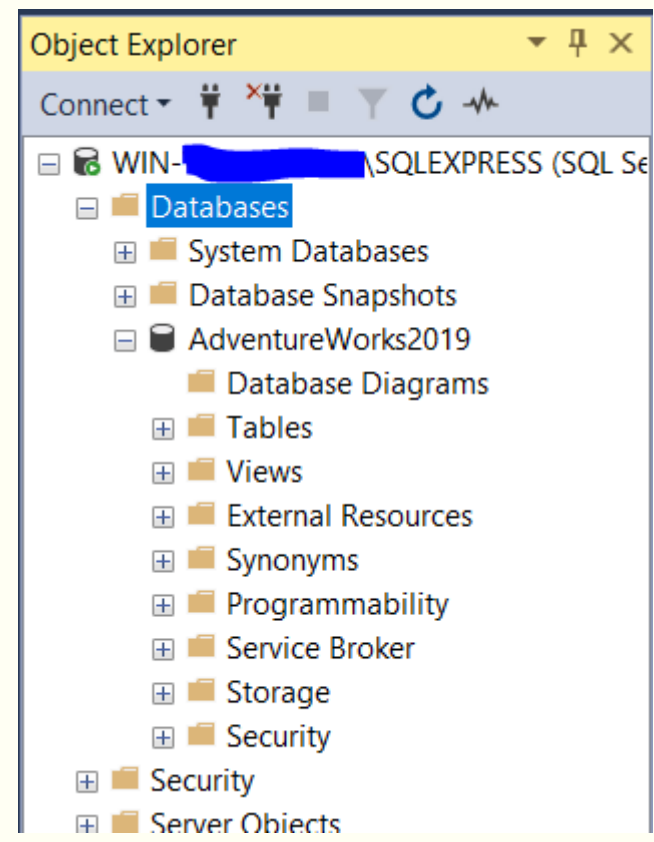
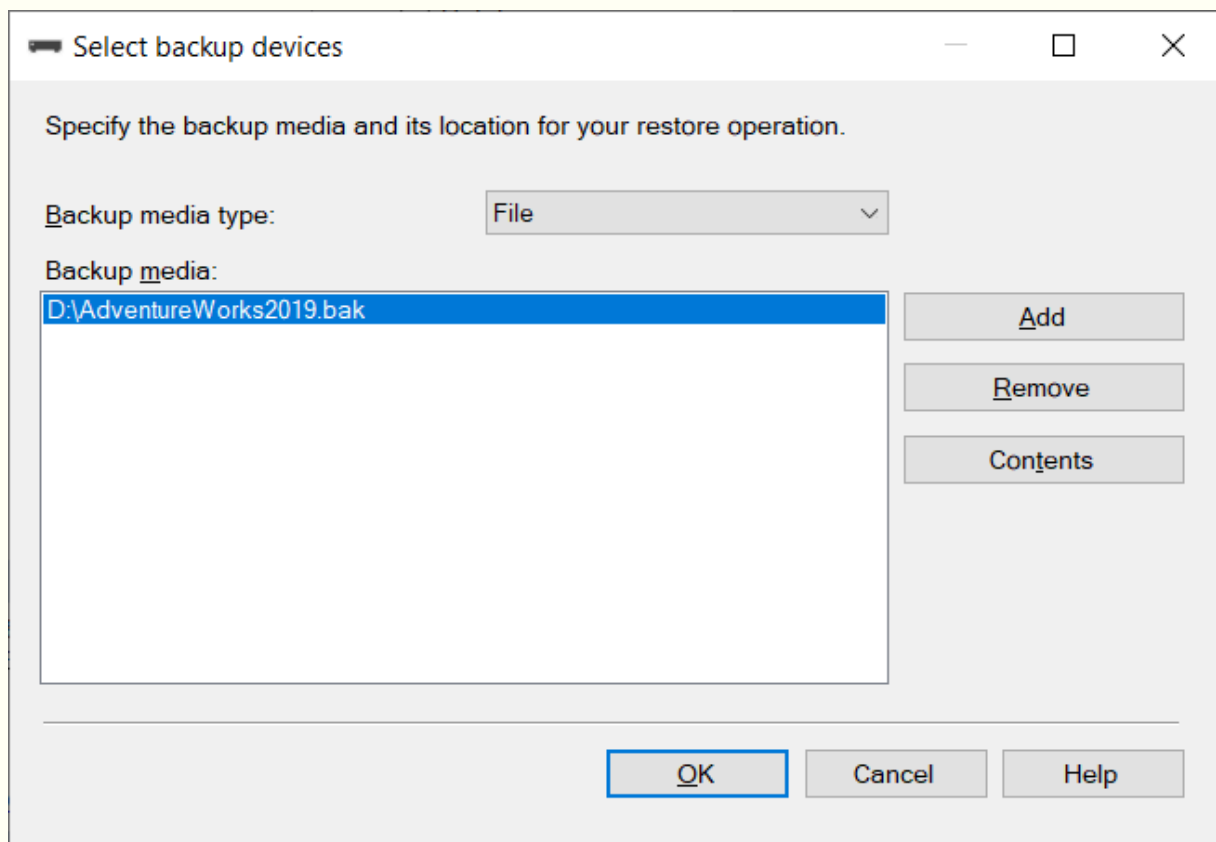
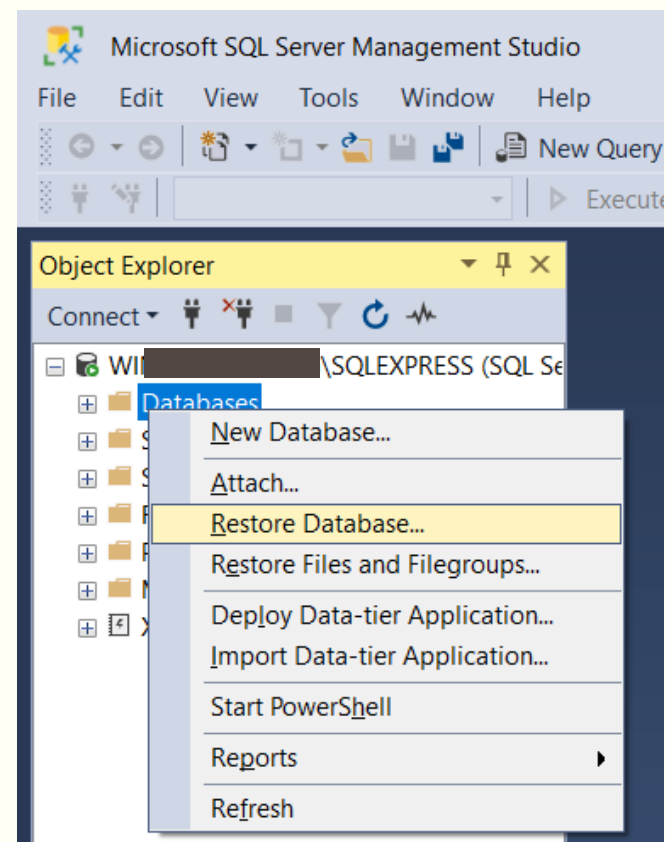
Підхід database-first (reverse-engineered)

- Існують інструменти, які дозволяють швидко генерувати код, необхідний для роботи з БД.
 - Проте цей код не дуже гнучкий.
 - Загалом підхід потребує регенерації коду кожного разу, коли БД змінюється при взаємодії додатку з її об'єктами.
 - Потрібний новий стовпчик? Необхідно додати його в БД напямую та регенерувати ваш контекст БД.
- Додатковий недолік підходу – database code часто не зберігається в репозиторії.
 - Маючи згенеровані моделі залучених об'єктів, код, який насправді створив їх у БД, часто відсутній.
 - Також маємо нехорошу історію об'єктів та їх стану в БД: складно при спробі відновлення попереднього patch отримати БД в тому ж стані, що і в момент того patch-a.

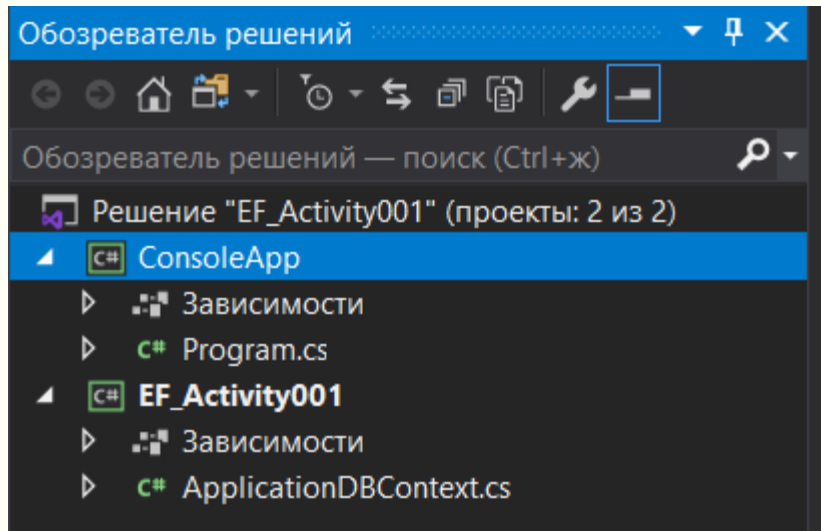
Підключення до локального серверу баз даних



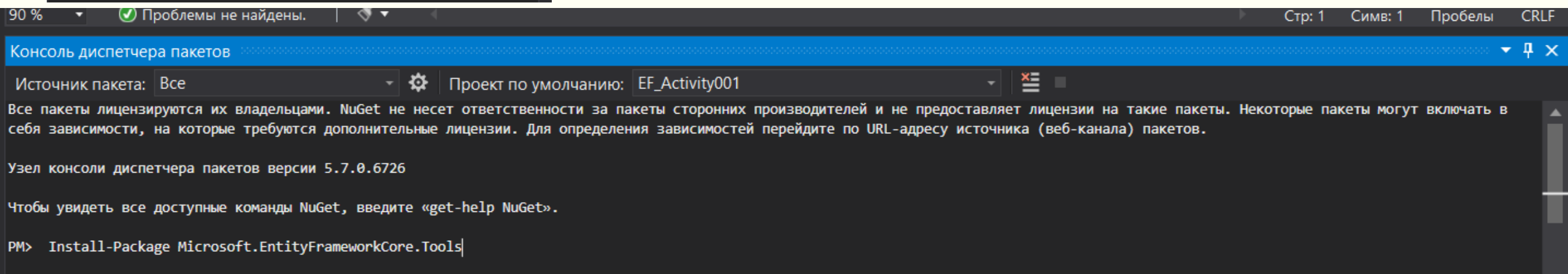
Відновлення БД з бекапу



Створення reverse-engineered БД в Entity Framework Core

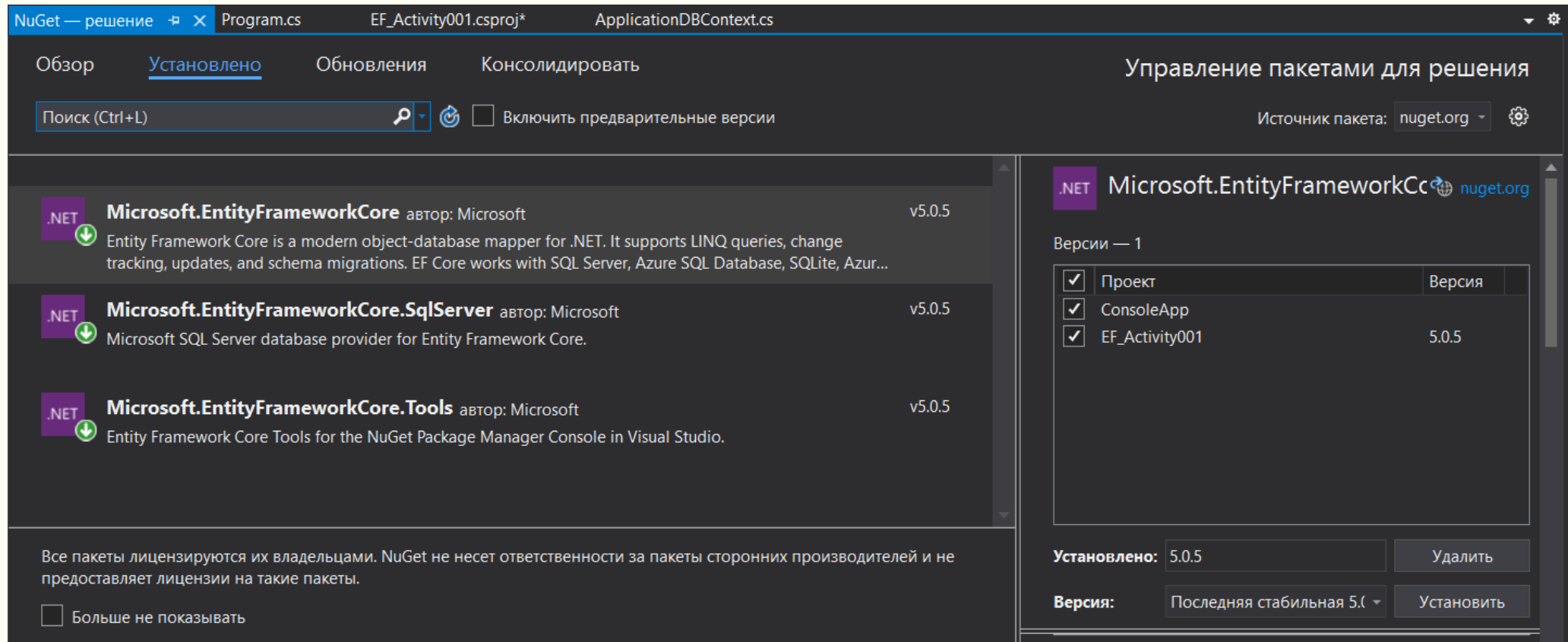


- Основная задача – создать работающий контекст БД для работы с существующей БД.
 - Добавим в старое решение новый проект ConsoleApp (клиентский додаток)
- Для реверс-инжиниринга БД нужны инструменты Entity Framework, які встановлюються за допомогою NuGet.



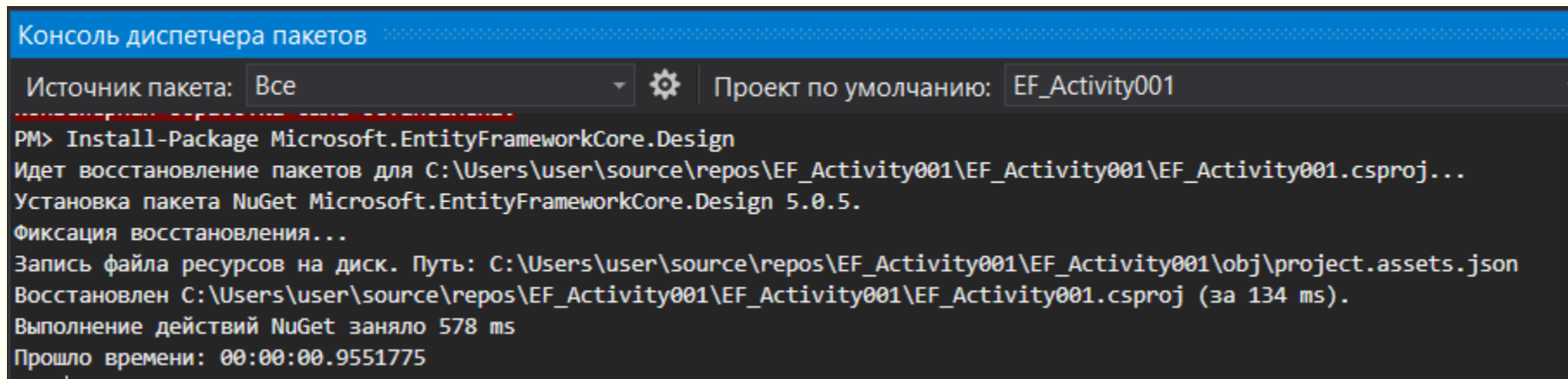
Встановлюємо на проєкт Entity Framework for SQL Server

- Команда Install-Package Microsoft.EntityFrameworkCore.SqlServer
- Посилаємось у менеджері NuGet-пакетів (*ПКМ на рішенні*) та натискаємо на кнопку встановлення:



Скаффолдинг нового контексту за допомогою команди Scaffold-Context

- **Скаффолдинг** – метод метапрограмування для створення веб-застосунків, які взаємодіють з БД.
 - Розробник задає специфікації, за якими надалі генерується програмний код для операцій створення певних записів у БД, їх читання, оновлення і вилучення (CRUD).
 - При скаффолдингу ми підключаємось напряму до БД.
- Щоб виконати scaffold-операцію, додамо ще один NuGet-пакет:
 - Install-Package Microsoft.EntityFrameworkCore.Design
 - Зауважте, що за умовчанням усі таблиці та схеми підлягають скаффолдингу, якщо не вказано іншого.
 - Для цього запустіть команду з connection string та вказаним провайдером даних.



```
Консоль диспетчера пакетов
Источник пакета: Все [gear icon] Проект по умолчанию: EF_Activity001
PM> Install-Package Microsoft.EntityFrameworkCore.Design
Идет восстановление пакетов для C:\Users\user\source\repos\EF_Activity001\EF_Activity001\EF_Activity001.csproj...
Установка пакета NuGet Microsoft.EntityFrameworkCore.Design 5.0.5.
Фиксация восстановления...
Запись файла ресурсов на диск. Путь: C:\Users\user\source\repos\EF_Activity001\EF_Activity001\obj\project.assets.json
Восстановлен C:\Users\user\source\repos\EF_Activity001\EF_Activity001\EF_Activity001.csproj (за 134 ms).
Выполнение действий NuGet заняло 578 ms
Прошло времени: 00:00:00.9551775
```

Скаффолдинг нового контексту за допомогою команди Scaffold-Context

Добавить подключение

Введите данные для подключения к выбранному источнику данных или нажмите кнопку "Изменить", чтобы выбрать другой источник данных и (или) поставщик.

Источник данных:
Microsoft SQL Server (SqlClient) Изменить...

Имя сервера:
WIN-... \SQLEXPRESS Обновить

Вход на сервер
Проверка подлинности: Проверка подлинности Windows

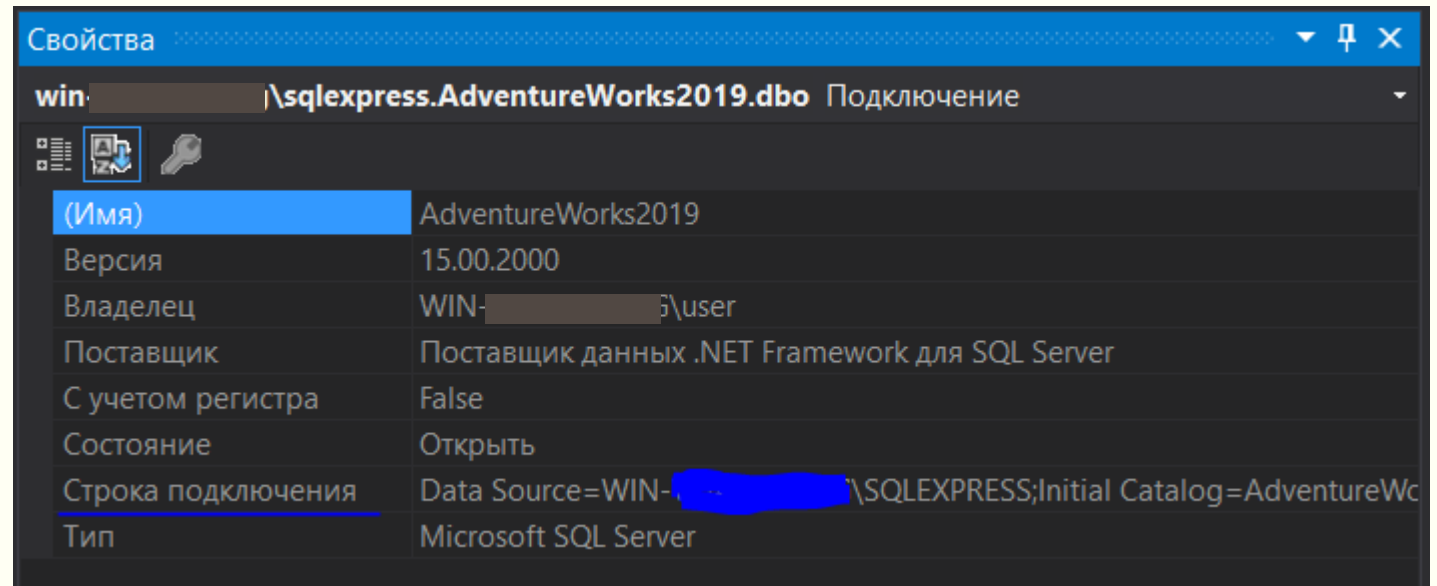
Имя пользователя:
Пароль:
☐ Сохранить пароль

Подключение к базе данных
☒ Выберите или введите имя базы данных:
AdventureWorks2019
☐ Прикрепить файл базы данных:
Обзор...
Логическое имя:

Дополнительно...

Проверить подключение OK Отмена

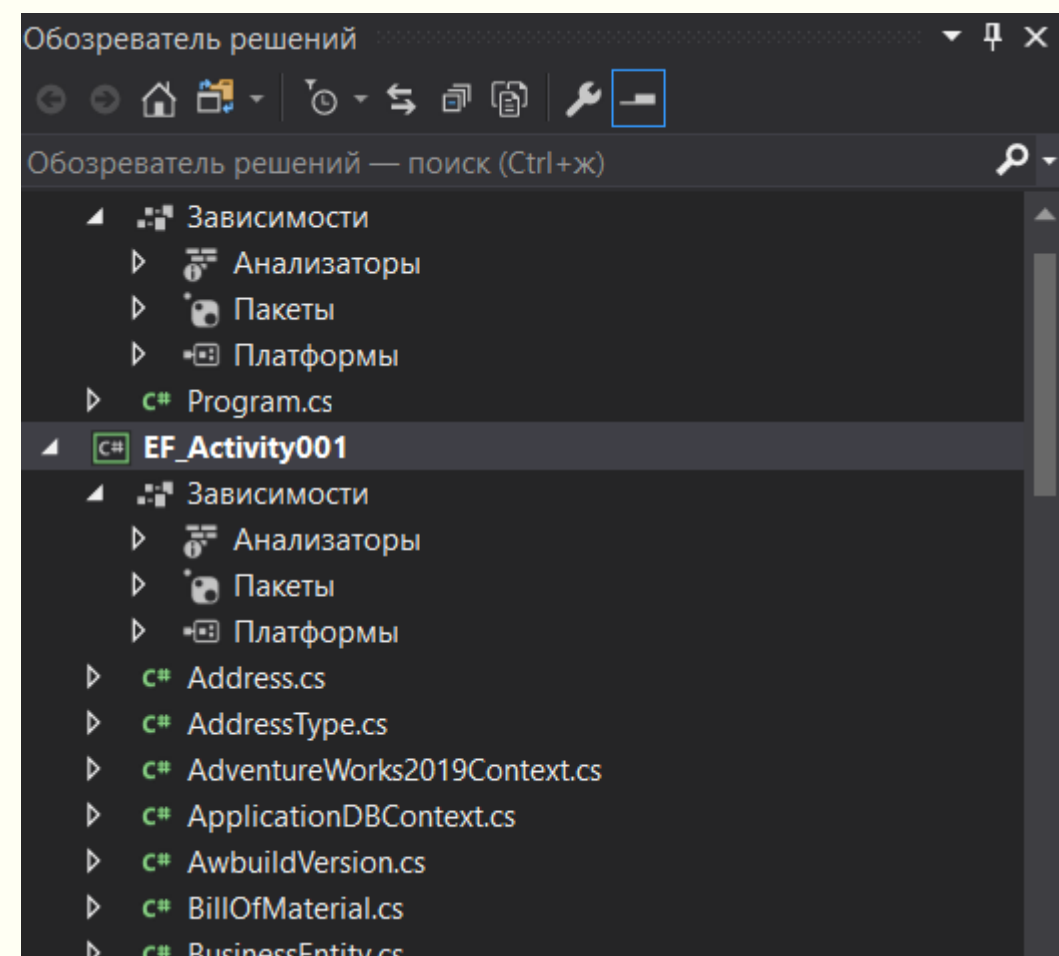
- Переконайтесь, що знову обрано проєкт EF_Activity001 у Package Manager Console.
 - Запускатимемо команду скаффолдингу на рівні database project.
 - Обравши EF_Activity001, запустіть команду Scaffold-DbContext <connection_string> [optional params].
 - Підключимо отриману БД в проєкт та отримаємо в її властивостях рядок підключення.



Скаффолдинг нового контексту за допомогою команди Scaffold-Context

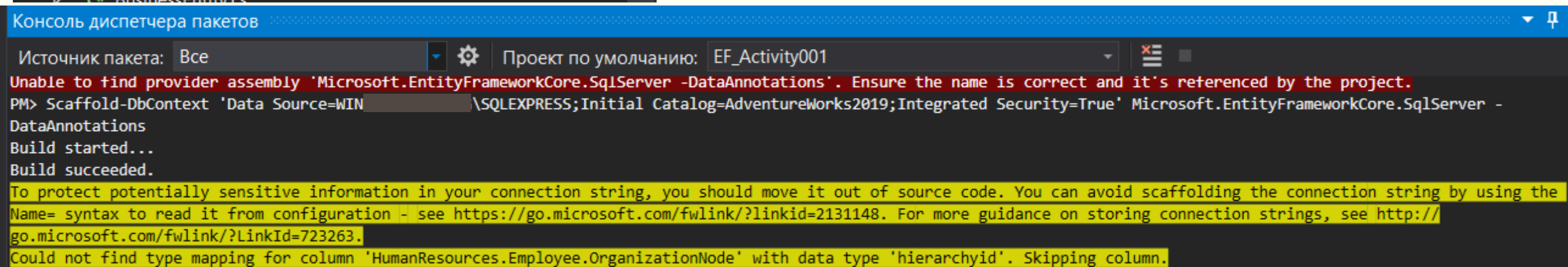
- Викличемо команду менеджера NuGet-пакетів Scaffold-DbContext, для якої в одинарних лапках передамо рядок підключення.
 - У якості провайдера визначаємо Microsoft.EntityFrameworkCore.SqlCore
 - При запуску команди отримаємо повний DbContext, згенерований та готовий до використання.
 - Зауважте, що присутні багато попереджень в ході генерації коду через складність схеми БД.

```
Консоль диспетчера пакетов
Источник пакета: Все
Проект по умолчанию: EF_Activity001
Прошло времени: 00:00:00.9551775
PM> Scaffold-DbContext 'Data Source=WIN-1\SQLEXPRESS;Initial Catalog=AdventureWorks2019;Integrated Security=True'
Командлет Scaffold-DbContext в конвейере команд в позиции 1
Укажите значения для следующих параметров:
Provider: Microsoft.EntityFrameworkCore.SqlServer
Build started...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the user+password authentication method. For more information, see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723443.
Could not find type mapping for column 'HumanResources.Employee.OrganizationNode' with data type 'hierarchyid'. Skipping column.
The column 'HumanResources.Employee.SalariedFlag' would normally be mapped to a non-nullable bool property, but it has a default constraint. Such a constraint is not supported by the provider. See https://go.microsoft.com/fwlink/?linkid=851278 for details.
The column 'HumanResources.Employee.CurrentFlag' would normally be mapped to a non-nullable bool property, but it has a default constraint. Such a constraint is not supported by the provider. See https://go.microsoft.com/fwlink/?linkid=851278 for details.
Unable to scaffold the index 'IX_Employee_OrganizationLevel_OrganizationNode'. The following columns could not be scaffolded: OrganizationNode.
Unable to scaffold the index 'IX_Employee_OrganizationNode'. The following columns could not be scaffolded: OrganizationNode.
```

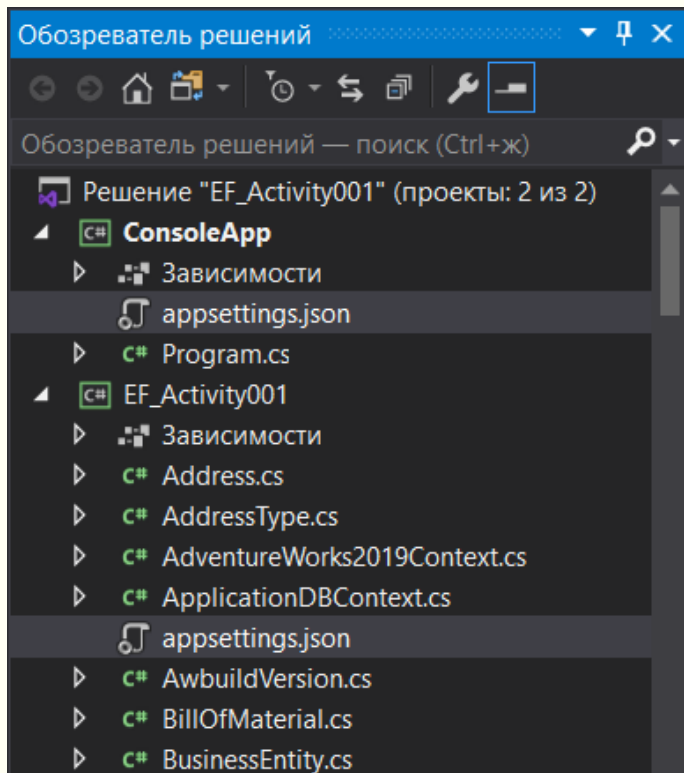


Формування остаточного контексту та конфігураційних файлів для підключення

- Тепер можемо розпочати роботу з БД в консольному додатку.
 - Для підключення до БД виконаємо деякі налаштування.
 - Оскільки проект уже існує, слід додати файли з налаштуваннями, в яких буде встановлюватись connection string для БД, який потім передаватиметься в контекст з метою підключення до неї.
 - Перед цим зробимо reset та будемо використовувати повний контекст БД з анотаціями даних (дає найкращий обсяг доступних даних).
 - Можливо, доведеться застосувати прапорець -f (forced).



Формування остаточного контексту та конфігураційних файлів для підключення



- Маючи готовий контекст, додамо файл з назвою `appsettings.json` в поточні проекти.
 - Додатково визначимо режим роботи з файлом як «Content and Copy».
 - Без цього `connection string` не буде зчитуватись при виконанні коду.
- Далі додамо ще три NuGet-пакети за допомогою Package Manager Console для консольного проєкту:
 - 1) `Install-Package Microsoft.Extensions.Configuration`
 - 2) `Install-Package Microsoft.Extensions.Configuration.FileExtensions`
 - 3) `Install-Package Microsoft.Extensions.Configuration.Json`
- Переконайтесь, що `connection string` заданий коректно і використовуються подвійні слеші (`\\`):
 - ```
{
 "ConnectionStrings": {
 "AdventureWorks2019": "WIN-◇ ◇ ◇ \\SQLEXPRESS;Initial
Catalog=AdventureWorks2019;Integrated Security=True"
 }
}
```
  - Якщо `appsettings.json` вже містить вузол, наприклад `Exclude`, і потрібно додати `connection string`, розташуйте кому в кінці існуючого вузла та додайте елемент `ConnectionStrings` після неї.

# Файл Program.cs

---

```
class Program
{
 static IConfigurationRoot _configuration;

 static void Main(string[] args)
 {
 BuildConfiguration();
 Console.WriteLine("Hello World!");
 }

 static void BuildConfiguration()
 {
 var builder = new ConfigurationBuilder()
 .SetBasePath(Directory.GetCurrentDirectory())
 .AddJsonFile("appsettings.json", optional: true,
 reloadOnChange: true);
 _configuration = builder.Build();
 }
}
```

- У Program.cs консольного додатку потрібно додати connection string у білдер (builder).
  - Включимо відповідний метод і статичну змінну:  
static IConfigurationRoot \_configuration;
- Після цього виправте проблему з IConfigurationRoot, обравши «Show Potential Fixes» та додавши простір імен Microsoft.Extensions.Configuration.
  - Далі додамо виклик методу BuildConfiguration() в Main().
  - <https://www.thecodebuzz.com/configurationbuilder-does-not-contain-definition-for-setbasepath-addjsonfile/>

# Підключення до БД та відображення результатів

---

- На фінальному кроці додамо іншу статичну змінну для збереження опцій білдера.
  - Це узагальнений об'єкт `DbContextOptionsBuilder` з типом згенерованого `DbContext`-а.
  - У даному випадку цей тип `AdventureWorksContext2019`, тому оголошуємо так:  
`static DbContextOptionsBuilder<AdventureWorks2019Context> _optionsBuilder;`
- Далі додаємо новий метод, щоб зібрати опції контексту БД:
  - ```
static void BuildOptions() {  
    _optionsBuilder = new DbContextOptionsBuilder<AdventureWorks2019Context>();  
    // _optionsBuilder.UseSqlServer(_configuration.GetConnectionString("AdventureWorks2019"));  
}
```
 - Даний метод задає налаштування білдера у вигляді нового екземпляру `DbContextOptionsBuilder`, а потім вказує білдеру використовувати SQL Server з налаштуваннями для connection string, визначеними в файлі `appsettings.json`.
 - Розташуйте код методу `BuildOptions()` у класі `Program` після методу `BuildConfiguration()`.

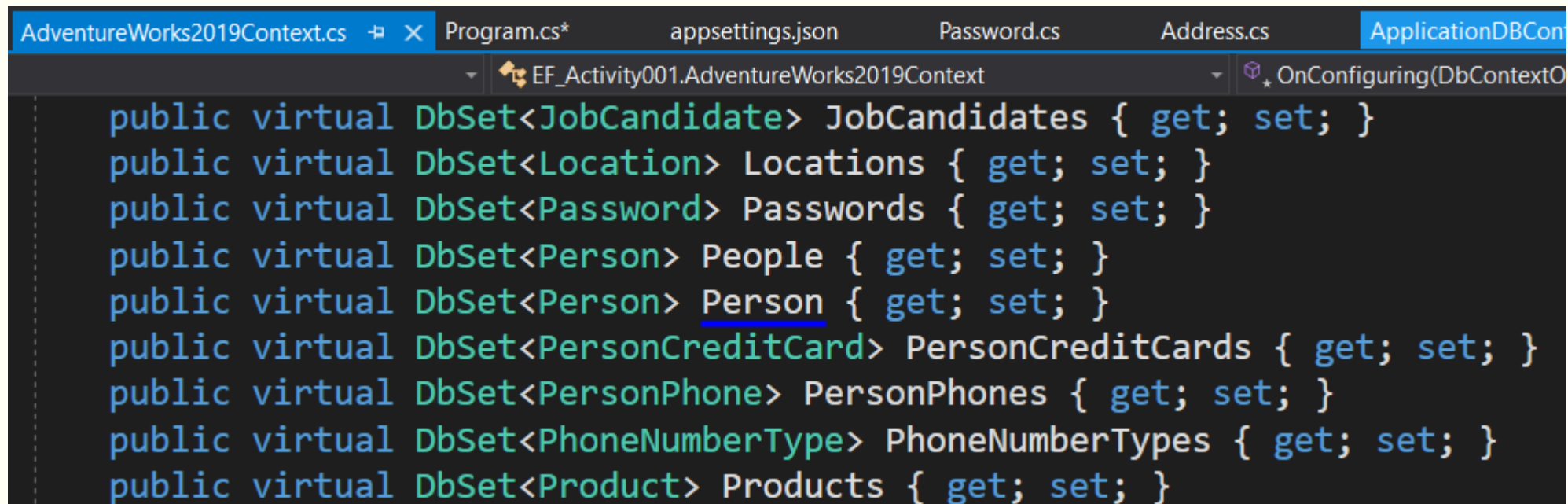
Підключення до БД та відображення результатів

- Для отримання даних потрібно мати запит і метод для виводу результатів.
 - Створимо метод ListPeople() у класі Program:

```
static void ListPeople()
{
    using (var db = new AdventureWorks2019Context(_optionsBuilder.Options))
    {
        var people = db.Person.OrderByDescending(x => x.LastName).Take(20).ToList();
        foreach (var person in people)
        {
            Console.WriteLine($"{person.FirstName} {person.LastName}");
        }
    }
}
```

Підключення до БД та відображення результатів

- Зауважте, що залежно від ваших налаштувань кодогенерації може знадобитись використання Person або People в назві entity set (entity set напряду визначається в контексті).
 - Якщо не бачите згенерованої властивості Person, можна створити інший DbSet або деякі опції були некоректно задані, і слід повністю регенерувати контекст БД.



```
AdventureWorks2019Context.cs Program.cs* appsettings.json Password.cs Address.cs ApplicationDBCon
EF_Activity001.AdventureWorks2019Context OnConfiguring(DbContextO

public virtual DbSet<JobCandidate> JobCandidates { get; set; }
public virtual DbSet<Location> Locations { get; set; }
public virtual DbSet<Password> Passwords { get; set; }
public virtual DbSet<Person> People { get; set; }
public virtual DbSet<Person> Person { get; set; }
public virtual DbSet<PersonCreditCard> PersonCreditCards { get; set; }
public virtual DbSet<PersonPhone> PersonPhones { get; set; }
public virtual DbSet<PhoneNumberType> PhoneNumberTypes { get; set; }
public virtual DbSet<Product> Products { get; set; }
```

Підключення до БД та відображення результатів

Консоль отладки Microsoft Visual Studio

```
Michael Zwilling  
Michael Zwilling  
Jake Zukowski  
Judy Zugelder  
Patricia Zubaty  
Carla Zubaty  
Karin Zimprich  
Karin Zimprich  
Tiffany Zimmerman  
Marc Zimmerman  
Krystal Zimmerman  
Kimberly Zimmerman  
Juanita Zimmerman  
Jo Zimmerman  
Jenny Zimmerman  
Jack Zimmerman  
Henry Zimmerman  
Curtis Zimmerman  
Christy Zimmerman  
Candice Zimmerman
```

- Формуємо метод Main() та запускаємо код на виконання:

```
static void Main(string[] args)  
{  
    BuildConfiguration();  
    BuildOptions();  
    ListPeople();  
}
```




ДЯКУЮ ЗА УВАГУ!

Наступне питання: Підхід Entity Framework: Code First