

Мета : Розробити програму для керування даними, що зберігаються в SQL-базі даних.

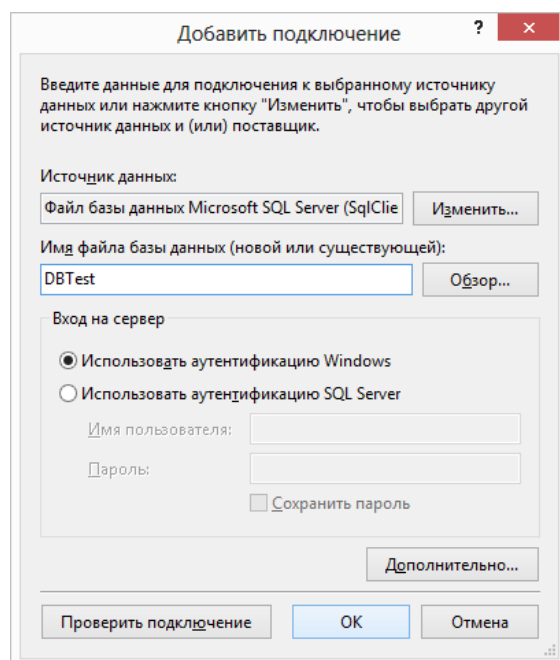
Завдання 1. Створення моделі даних

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

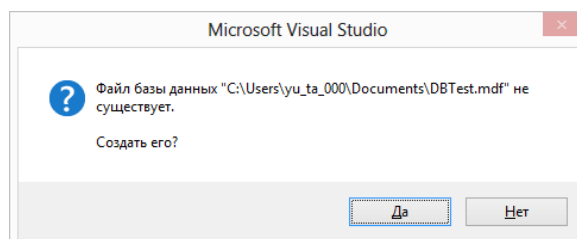
Створення локальної бази даних

Порядок створення локальної SQL бази даних:

1. Запустити **Visual Studio**.
2. Відкрити **браузер серверів**, скориставшись однойменною командою з меню **View**.
3. Клацніть правою кнопкою миші на вузлі **Підключення даних** та виберіть у контекстному меню команду **Додати підключення...**
4. У діалозі вкажіть ім'я створюваної бази даних і натисніть кнопку **OK**.



5. Підтвердьте необхідність створення нової бази даних.

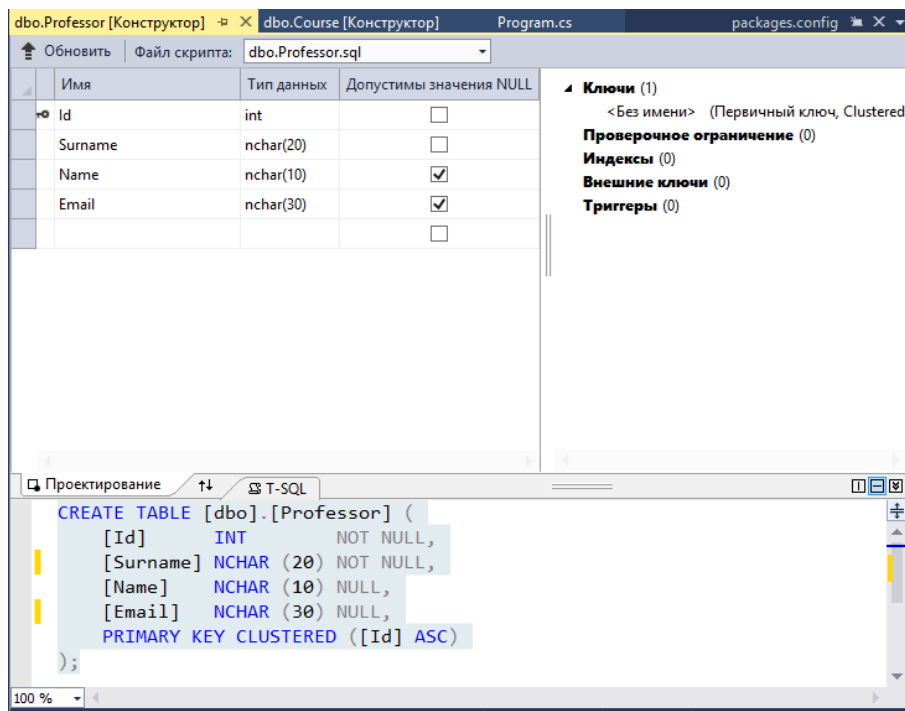


Додавання таблиці до бази даних

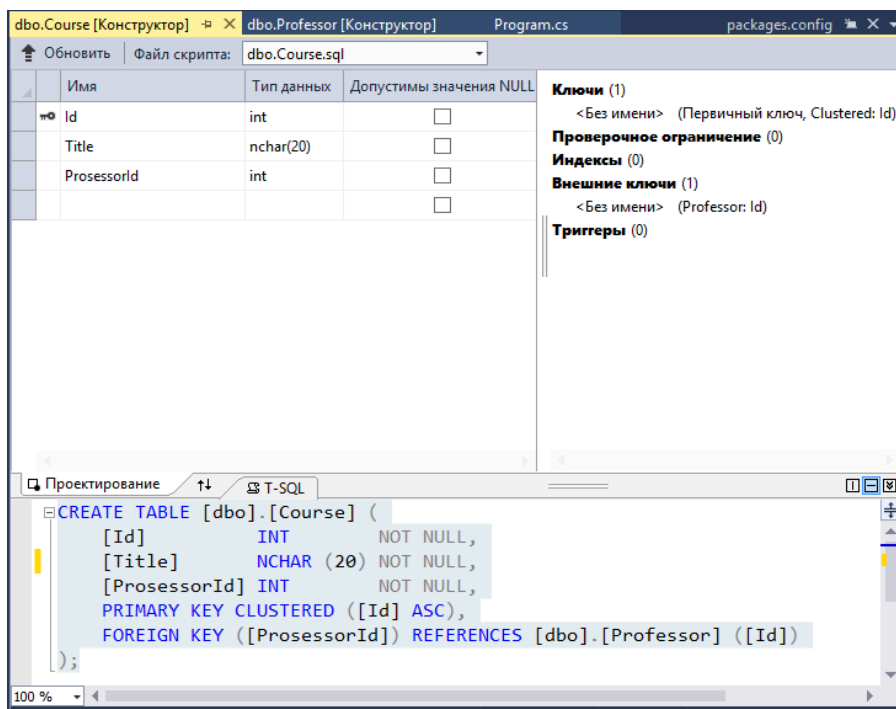
Щоб додати таблицю до бази даних, виконайте такі дії:

1. У **браузері серверів** розгорніть вузол створеної бази даних.

2. Клацніть правою кнопкою миші по вузлу **Таблиці** та виберіть у контекстному меню команду **Додати нову таблицю**.
3. У вікні редактора таблиць вкажіть для кожного поля назву, тип та інші властивості (за потреби).



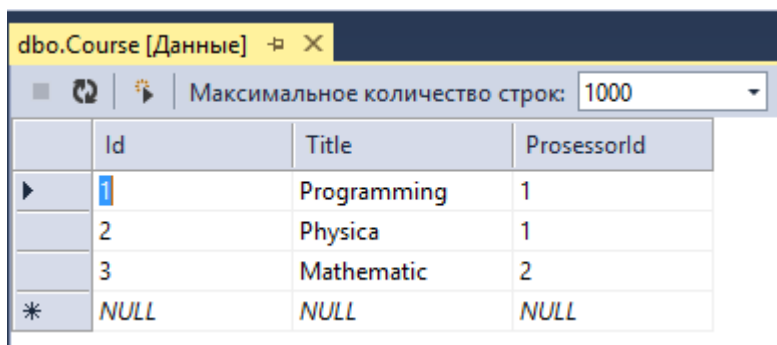
4. Введіть назву таблиці та оновіть базу даних.
5. Повторіть п.3-4 кожної таблиці бази даних.
6. Встановіть зв'язок між таблицями, редагуючи код скрипта T-SQL. Наприклад, зв'язок типу «один-до багатьох» створюється за допомогою поля зовнішнього ключа:



Редагування даних

Для введення та редагування даних у таблицю виконайте такі дії:

1. У **браузері серверів** розгорніть вузол **таблиці**.
2. Клацніть правою кнопкою миші на вузлі таблиці та виберіть у контекстному меню команду **Показати таблицю даних**.
3. Відредагуйте дані таблиці.



	Id	Title	ProsessorId
▶	1	Programming	1
	2	Physica	1
	3	Mathematic	2
*	NULL	NULL	NULL

4. Закрийте таблицю.
5. Аналогічно заповніть даними інші таблиці бази даних.

Додавання бази даних до проекту та створення файлу конфігурації

Щоб додати базу даних до проекту, виконайте такі дії:

1. У **браузері рішень** виділіть корінь проекту і командою **Додати/Створити папку** контекстного меню створіть підкаталог для розміщення бази даних, наприклад, **Data**.
2. У **браузері рішень** за допомогою команди контекстного меню **Додати/Існуючий елемент...** додайте файл бази даних *.mdf до проекту (як правило, вона знаходиться в каталозі C:\Users\Admin).
3. Виділіть у **браузері** файл доданої бази даних (*.mdf) і через панель **Властивості** перевірте, що його властивості мають такі значення: **Дія при складанні** = **Зміст**, **Копіювати у вихідний каталог** = **Завжди копіювати**.

Для створення конфігураційного файлу виконайте такі дії:

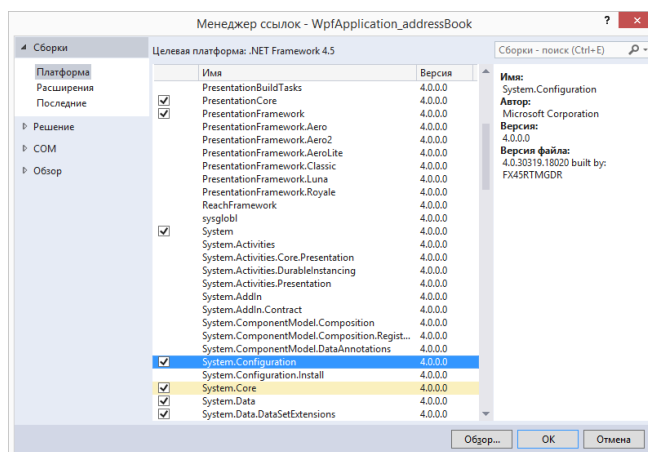
1. Відкрийте файл конфігурації **App.config** у проекті .
2. Вставте в розділ <configuration/> розділ <connectionStrings/>, що містить відомості про рядок з'єднання з базою даних (ім'я та значення):

```
<connectionStrings>
  <add name="connectionString_ADO"
    connectionString="Data Source=R501N10;Initial Catalog=Clients;Integrated Security=True"/>
</connectionStrings>
```

Тут *name* - це ім'я з'єднання, *Initial Catalog* - це назва бази даних, *Data Source* - це ім'я сервера бази даних (SQL Server name).

Значення рядка з'єднання можна скопіювати з властивостей бази даних.

3. У **браузері рішень** викличте для вузла **References** проекту контекстне меню та командою **Додати посилання** додайте посилання на бібліотечну збірку **System.Configuration.dll**, в якій знаходиться клас **ConfigurationManager** для роботи з конфігураційним файлом **App.config** з процедурного коду:



Тепер рядок з'єднання з файлу **App.config** можна отримати таким чином:

```
String connectionString = System.Configuration.ConfigurationManager.ConnectionStrings["  
connectionStringName "].ConnectionString;
```

Хід виконання

1. Запустіть **браузер серверів**, скориставшись однойменною командою з меню **View**.
2. Створіть SQL базу даних, що складається з кількох пов'язаних таблиць (дані наведені в Таблиці 1).
3. Встановіть зв'язки між таблицями.
4. Збережіть файли сценаріїв створення таблиць та скопіюйте їх у звіт про виконання завдання 1.
5. Додайте дані до таблиць.

Таблиця 1. Дані для розробки баз даних

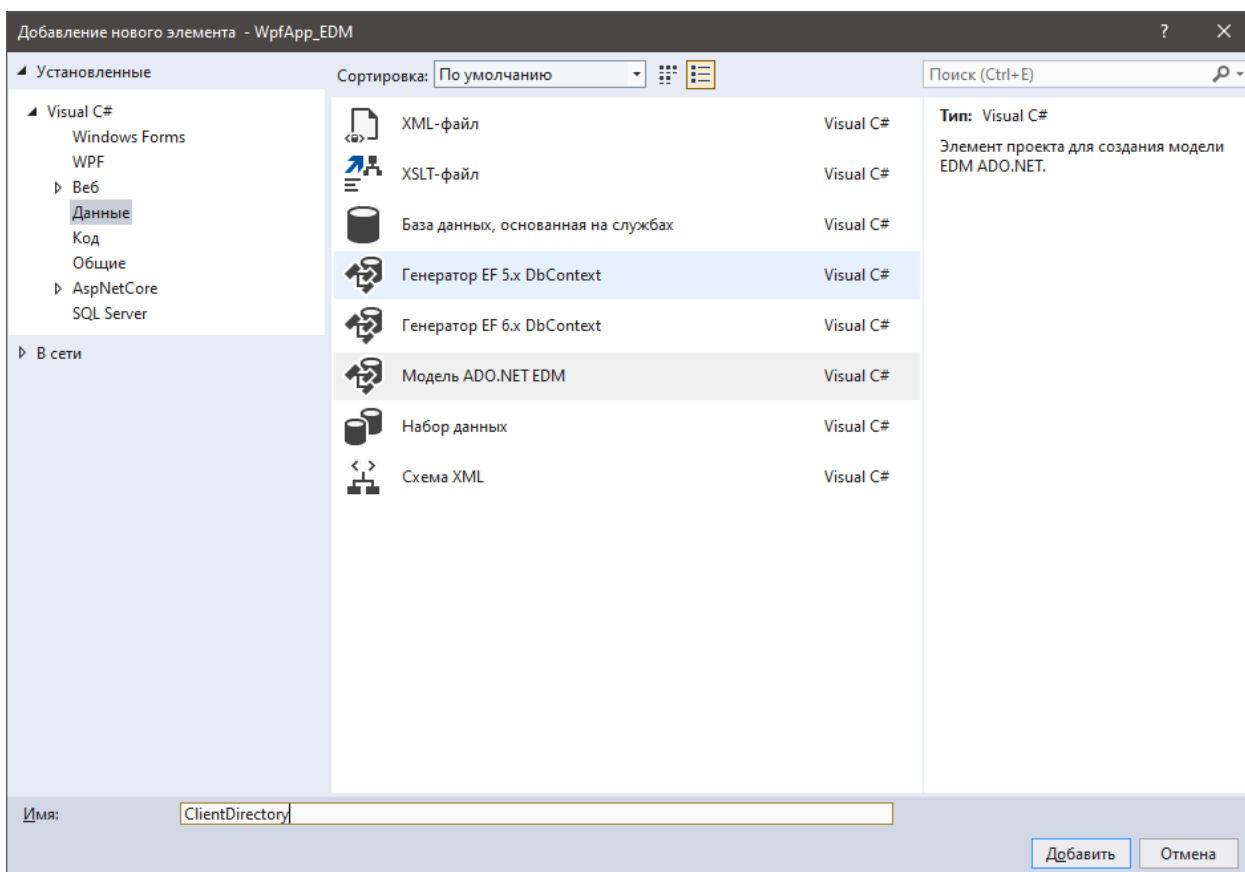
Варіанти	База даних	Таблиці БД	Поля таблиць
1-5	Довідник клієнтів	Клієнти Компанії	ID, Назва, Телефон, Код компанії, Надходження, Витрати Код компанії, Назва компанії
6-10	Довідник товарів	Товари Одиниці виміру	Артикул, Код одиниці виміру, Кількість, Ціна Код одиниці виміру, Одиниця виміру
11-15	Довідник студентів	Студенти Група	Номер залікової книги, Код групи, Адреса Код групи, Група
16-20	Довідник книг	Книги Видавництва	ISBN, Автори, Код видавництва, Рік видання Код видавництва, Видавництво
21-25	Довідник співробітників	Співробітник и Підрозділи	ID, Підрозділ, Посада, Телефон Код підрозділу, Підрозділ
Свій варіант			

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

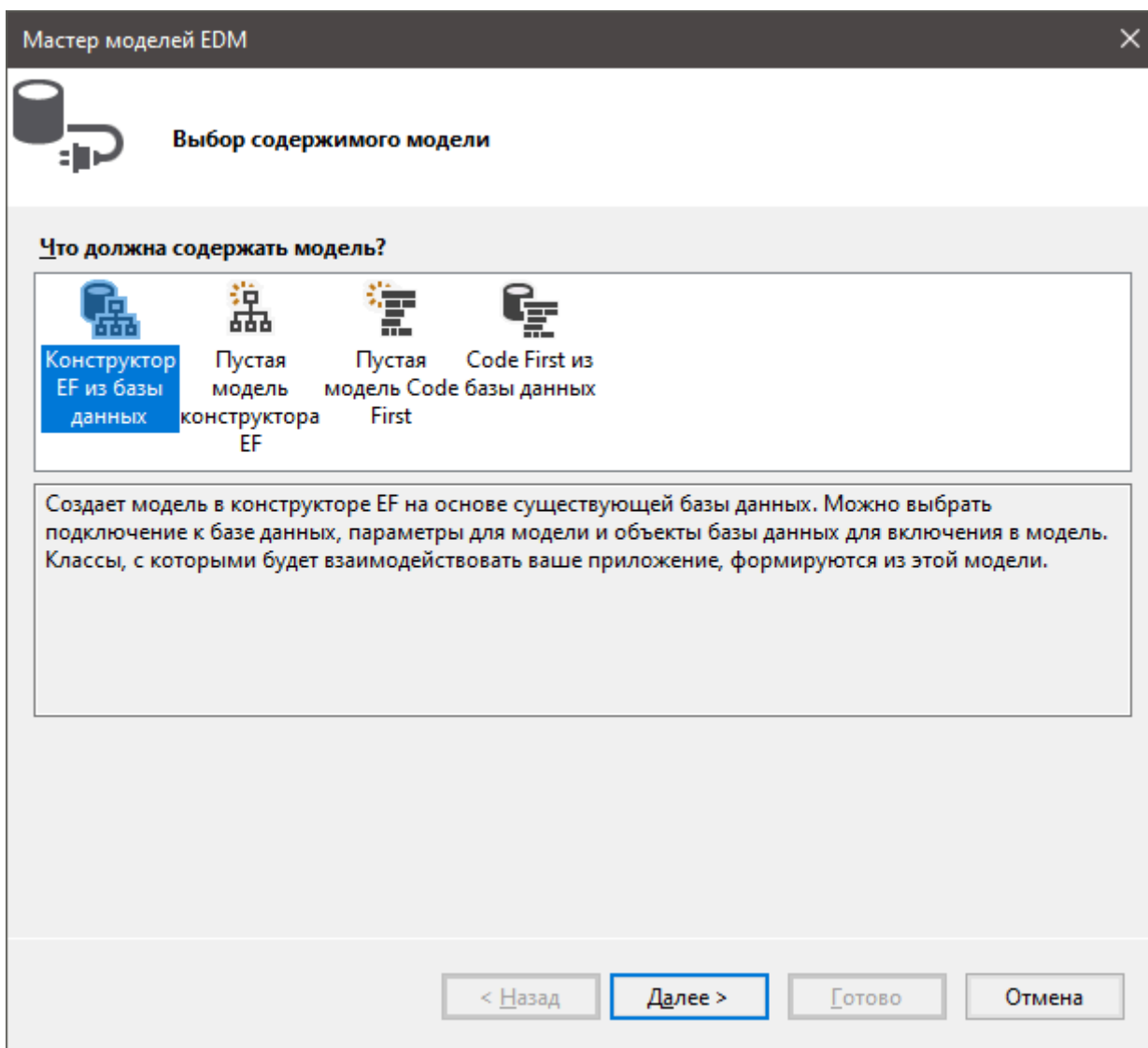
Взаємодія програми із джерелом даних може базуватися на моделі "сутність-зв'язок" – **Entity Data Model (EDM)**. Модель EDM описує структури даних на основі сутностей та зв'язків, які є незалежними від схем зберігання. В результаті такого підходу форма зберігання даних відокремлюється від застосунка та не впливає на його розробку. Це забезпечується тим, що сутності та зв'язки описують структуру даних так, як вона використовується у застосунку.

Entity Framework — це технологія об'єктно-реляційного відображення, яка дозволяє розробникам .NET працювати з реляційними даними за допомогою предметно-орієнтованих об'єктів. Це усуває потребу у більшості коду доступу до даних, який зазвичай потрібно написати розробникам. Entity Framework — це рекомендована технологія моделювання об'єктно-реляційного відображення (ORM) для нових програм .NET.

Для створення EDM-моделі у складі Visual Studio включено **ADO.NET Entity Data Model**. Для його підключення до поточного проекту необхідно скористатися командою **Проект-Додати новий елемент...** у вузлі **Дані** вибрати **Модель ADO.NET EDM**, задати ім'я моделі та натиснути кнопку **Додати**.




В результаті відкриється вікно для запуску майстра створення моделі:



За наявності бази даних слід залишити вибраною піктограму **Створити з бази даних** та натиснути кнопку **Далі**.

На наступному етапі роботи майстра потрібно задати ім'я моделі. За промовчанням пропонується зручний варіант імені у вигляді **<Ім'я_бази_даних>Entities**, який недоцільно змінювати.

Мастер моделей EDM

 **Выбор подключения к данным**

Какое подключение к данным будет использоваться приложением для подключения к базе данных?

r501n10.ClientDirectory.dbo Создать соединение...

Возможно, эта строка подключения содержит конфиденциальные данные (например, пароль), которые требуются для подключения к базе данных. Хранение конфиденциальных данных в строке подключения может представлять угрозу безопасности. Включить конфиденциальные данные в строку подключения?

☒ Нет, исключить конфиденциальные данные из строки подключения. Они будут заданы в коде приложения.

☐ Да, включить конфиденциальные данные в строку подключения.

Строка подключения:

```
metadata=res://*/Model1.csdl|res://*/Model1.ssdl|
res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string="data
source=R501N10;initial catalog=ClientDirectory;integrated
security=True;pooling=False;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Сохранить параметры соединения в App.Config как:


ClientDirectoryEntities

< Назад **Далее >** Готово Отмена

На цьому кроці пропонується зберегти модель (разом із файлом бази даних та файлом конфігурації) у проєкті.

Потім пропонується вибрати версію Entity Framework:


Мастер моделей EDM

 **Выберите версию**

Какую версию Entity Framework вы хотите использовать?

☒ Entity Framework 6.0

☐ Entity Framework 5.0

 Можно также установить и использовать другие версии Entity Framework. [Получить дополнительные сведения об этом](#)

< Назад **Далее >** Готово Отмена

Насамкінець слід вибрати необхідні таблиці бази даних та ім'я простору імен моделі, в якості якого за умовчанням рекомендується <Ім'я_бази_даних>Model:

Мастер моделей EDM

Выберите параметры и объекты базы данных

Какие объекты базы данных нужно включить в модель?

- ☒ Таблицы
 - ☒ dbo
 - ☒ Clients
 - ☒ Companies
 - ☐ Представления
 - ☐ Хранимые процедуры и функции

☐ Формировать имена объектов во множественном или единственном числе

☒ Включить столбцы внешних ключей в модель

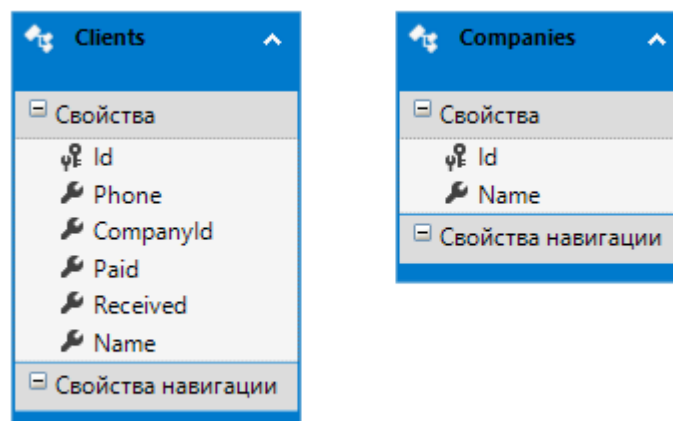
☐ Импортировать выбранные хранимые процедуры и функции в модель сущностей

Пространство имен модели:

ClientDirectoryModel

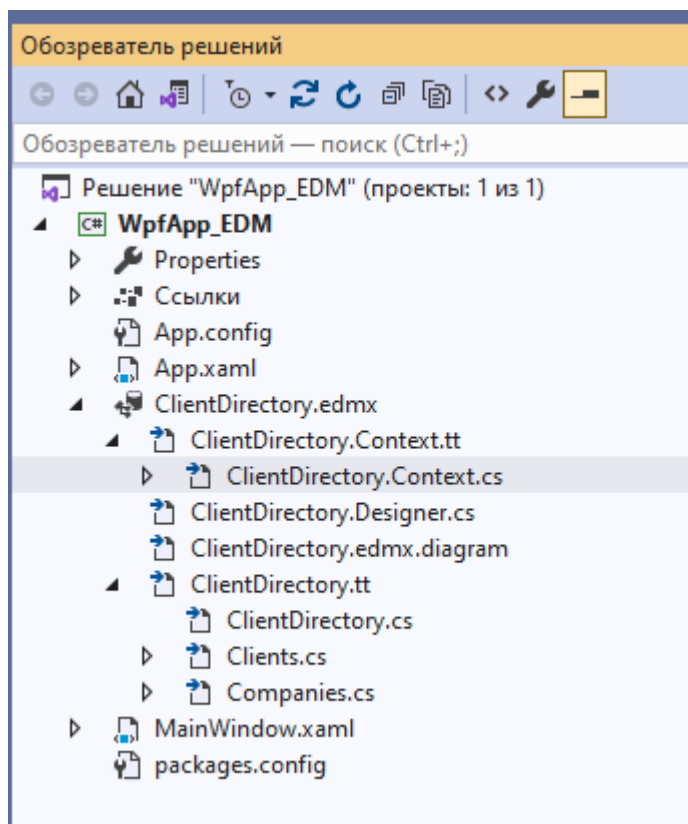
< Назад Далее > Готово Отмена

Після завершення роботи майстра відкривається схема сутностей:



Важливо, що у сформованих класах створюються навігаційні властивості, які містять колекції з пов'язаних елементів іншого класу.

Одночасно до браузера рішення додається вузол моделі з розширенням `<Ім'я_моделі>.Context.edmx` :



Ключовим об'єктом тут є контейнер або контекст даних - `<ім'я_моделі>. Context.cs`:

```

namespace WpfApp_EDM
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    ссылка: 1
    public partial class ClientDirectoryEntities1 : DbContext
    {
        Ссылка: 0
        public ClientDirectoryEntities1()
            : base("name=ClientDirectoryEntities1")
        {
        }

        Ссылка: 0
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        Ссылка: 0
        public virtual DbSet<Clients> Clients { get; set; }
        Ссылка: 0
        public virtual DbSet<Companies> Companies { get; set; }
    }
}

```

Об'єкти цього класу сприймаються як сховище даних, тому що через них можна взаємодіяти з базою даних. У ньому створюються екземпляри класів із простору імен **System.Data.Entity**, що забезпечують основу функціональності Entity Framework:

- **DbContext** : визначає контекст даних, що використовується для взаємодії з базою даних.
- **DbModelBuilder** : зіставляє класи мовою C# із сутностями у базі даних.
- **DbSet/DbSet<TEntity>** : представляє набір сутностей, що зберігаються в базі даних

У додатку, що працює з БД через Entity Framework, доступ до даних здійснюється через об'єкт контексту - класу, похідного від **DbContext**, та його властивості типу **DbSet**. Наприклад, у C#-кодi застосунку це можна було б зробити так:

```

namespace WpfApp_EDM
{
    /// <summary>
    /// Логика взаємодії для MainWindow.xaml
    /// </summary>
    Ссылка: 3
    public partial class MainWindow : Window
    {
        ClientDirectoryEntities1 context;
        List<Clients> clients;
        Ссылка: 0
        public MainWindow()
        {
            InitializeComponent();

            context = new ClientDirectoryEntities1();...
            clients = new List<Clients>();
        }

        ссылка: 1
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            foreach (var client in context.Clients)
            {
                clients.Add(client);
            }

            clientDir.ItemsSource = clients;
        }
    }
}

```

Тут створюється об'єкт контексту **context** та за допомогою циклу елементи колекції **context.Clients** перетворюються в колекцію типу **List**, яку вже можна використовувати в якості джерела даних для елемента **DataGrid**.

Хід виконання

1. Додайте в проект EDM-модель на основі бази даних.
2. В XAML-код додайте елемент **TabControl**.
3. Створіть в ньому нову вкладку **TabItem**.
4. Додайте на вкладку елемент **DataGrid** та присвойте йому ім'я.
5. В C#-коді сформууйте джерело для відображення даних з таблиці бази даних (довільної).
6. Замініть заголовок вкладки на назву таблиці.
7. Зробіть скріншот екрана і вставте його у звіт про виконання завдання 2.
8. Повторіть п.п.3-7 для кожної таблиці бази даних.

В Entity Framework можна створювати запити двома способами:

1. За допомогою методів LINQ.
2. За допомогою запитів SQL.

Використання LINQ

Language-Integrated Query (LINQ) — потужна мова запитів, представлена у Visual Studio. Як впливає з назви, запити LINQ-to-Entities працюють із набором сутностей (типу DbSet-властивості), щоб отримати доступ до даних із бази даних.

Методи LINQ неявно генерують вираз SQL для створення запитів до бази даних. Розглянемо їх.

Метод **Where** дозволяє конкретизувати вибірку, накладаючи умови на значення властивостей об'єкта. Наприклад, вираз **Course course=context.Courses.Where(c=>c.Id==1)** забезпечує звернення до екземпляра колекції з ідентифікатором 1.

Метод **Find** використовується, якщо треба знайти в базі даних один об'єкт за його ID. Так, вираз **Course course = context.Courses.Find(3)** забезпечує знаходження елемента колекції з ідентифікатором 3. Якщо об'єкт з таким ID не знайдений, то змінна *course* отримає значення *null*.

Метод **Count** підраховує кількість об'єктів у вибірці: **int num = context.Courses.Count()**. Метод також може приймати лямбда-вираз, що вказує на умову вибірки: **int num = context.Courses.Count(c=>c.Id>2)**.

Методи **Min** та **Max** дозволяють отримати мінімальне та максимальне значення властивості: **int num = context.Courses.Max(c=>c.Id)**.

Метод **Select** подібний до **Where**, але дозволяє обмежити результати запиту певними умовами. Наприклад:

```
List<Professor> professors = context.Courses.Select(c => c.Professor).ToList();
```

```
foreach (var item in professors)
```

```
    Console.WriteLine(item.Id + "\t" +item.Surname);
```

В даному випадку вибірка йде лише за властивістю **Professor** об'єкта **Courses**, і результатом вибірки буде колекція рядків, а не об'єктів **Courses**.

Для поєднання таблиць за певним критерієм використовується метод **Join**. Наприклад, у разі коли таблиця телефонів і таблиця компаній має загальний критерій - id компанії, за яким можна провести об'єднання таблиць:

```
using(PhoneContext db = new PhoneContext())
```

```
{
```

```
    var phones = db.Phones.Join(db.Companies, p => p.CompanyId, c => c.Id, (p, c) => new
```

```
    {
```

```
        Name=p.Name,
```

```
        Company = c.Name,
```

```
        Price=p.Price
```

```

    });

    foreach (var p in phones)

        Console.WriteLine("{0} ({1}) - {2}", p.Name, p.Company, p.Price);

}

```

Синтаксис методу LINQ:

```

using (var context = new SchoolDBEntities())

{

    var query = context.Students

        .where(s => s.StudentName == "Bill")

        .FirstOrDefault<Student>();

}

```

Синтаксис запиту LINQ:

```

using (var context = new SchoolDBEntities())

{

    var query = from st in context.Students

        where st.StudentName == "Bill"

        select st;

    var student = query.FirstOrDefault<Student>();

}

```

Рекомендується створювати його екземпляр контексту у using(), щоб при виході за межі тіла він автоматично видалявся.

Використання Sql

Для написання явних запитів на SQL використовуються методи **SqlQuery()** та **ExecuteSqlCommand()**.

За допомогою **SqlQuery** можна зробити вибірку:

```

var professors = context.Professors.SqlQuery("SELECT * FROM PROFESSOR").ToList();

foreach (var item in professors)

    Console.WriteLine(item.Id + "\t" + item.Surname);

```

Можна також виконати запит на вибір окремих властивостей (як за допомогою методу **Select**):

```
var names = context.Database.SqlQuery<string>("SELECT Name FROM PROFESSOR").ToList();
```

```
foreach (var item in names)
```

```
    Console.WriteLine(item);
```

Метод **ExecuteSqlCommand** застосовується, коли потрібно видаляти, оновлювати вже існуючі або вставляти нові записи. Метод повертає кількість оброблених командою рядків:

```
context.Database.ExecuteSqlCommand("INSERT INTO Professor (Id) VALUES (5)");
```

```
context.Database.ExecuteSqlCommand("UPDATE Professor SET Surname='Vorobyaninov' WHERE Id=5");
```

```
context.Database.ExecuteSqlCommand("DELETE FROM Professor WHERE Id=5");
```

Хід виконання

1. Виходячи з контексту бази даних підготуйте команду/запит LINQ чи SQL для відбору даних.
2. Додайте в коді C# метод з реалізацією команди/запиту.
3. Додайте на головну форму нову вкладку.
4. Розмістіть в ній елемент **DataGrid** для відображення результату і елементи для визначення параметрів команди/запиту (за необхідності).
5. Повторіть п.п.1-4. для створення ще 2-3 команд/запитів LINQ чи SQL.