

Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра ПІ

## **КОМПЛЕКС НАВЧАЛЬНО-МЕТОДИЧНОГО ЗАБЕЗПЕЧЕННЯ**

навчальної дисципліни

«\*Робота з даними на платформи .Net»

підготовки бакалавра  
напряму 6.050103 – Програмна інженерія,  
спеціальності 121 – «Інженерія програмного забезпечення»

Розробник: Широкопетлева М.С., ст. викл. каф. ПІ

Схвалено на засіданні кафедри ПІ  
Протокол від 17 жовтня 2017 р. № 4

Харків 2017 р.

## ЗМІСТ

1. Робоча програма	<a href="#"><u>3</u></a>
2. Методичні вказівки до практичних занять	<a href="#"><u>18</u></a>
3. Методичні вказівки до лабораторних робіт	<a href="#"><u>47</u></a>
4. Контрольні завдання	<a href="#"><u>87</u></a>

Харківський національний університет радіоелектроніки

Кафедра Програмної інженерії

ЗАТВЕРДЖУЮ  
Декан факультету КН  
А.Л.Єрохін  
(підпис, ініціали, прізвище)

«11» 09 \_\_\_\_\_ 2017 р.

## РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

### «\*Робота з даними на платформи .Net»

напрямок підготовки 6.050103 «Програмна інженерія»  
(шифр і назва напряму підготовки)

\_\_\_\_\_  
(шифр і назва спеціальності)

спеціалізація \_\_\_\_\_  
(назва спеціалізації)

факультет Комп'ютерних наук  
(назва інституту, факультету, відділення)

Харків – 2017

Робоча програма з дисципліни «\*Робота з даними на платформи .Net» для студентів за напрямом підготовки 6.050103 «Програмна інженерія»

Розробник: ст.викл. каф. ПІ, Широкопетлєва М.С.

Робочу програму схвалено на засіданні кафедри Програмної інженерії

Протокол від 31 серпня 2017 року № 1

Завідувач кафедри \_\_\_\_\_ З.В. Дудар  
(підпис) (ініціали, прізвище)

Схвалено методичною комісією факультету КН.

Протокол від 31 серпня 2017 року № 1  
Голова методичної комісії \_\_\_\_\_ О.Ф. Лановий  
(підпис) (ініціали, прізвище)  
“ \_\_\_\_\_ ” \_\_\_\_\_ 2017 р.

© ХНУРЕ, 2017

## ОПИС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Найменування показників	Галузь знань, напрям підготовки, освітньо-кваліфікаційний рівень	Характеристика навчальної дисципліни	
		денна форма навчання	заочна форма навчання
Кількість кредитів ЄКТС* – 5	Галузь знань 0501 «Інформатика та обчислювальна техніка»	За вибором Цикл професійної та практичної підготовки	
	Напрямок підготовки 6.050103 «Програмна інженерія» (шифр і назва)		
Модулів ** – 3	Спеціальність (шифр і назва)	Рік підготовки	
Змістових модулів – 3		3-й	4-й
Індивідуальних завдань*: КР		Семестр	
Курс.пр. -		6-й	7-й
Загальна кількість годин* – 150		Кількість годин	
Тижневих годин для денної форми навчання (1 семестр – 17 тижнів): аудиторних – 4,12 год. самостійної роботи студента – 1,52 год.	Освітньо-кваліфікаційний рівень: бакалавр (назва ОКР)	Аудиторні 1) лекції, год.	
		30	8
		2) практичні, год.	
		10	6
		3) лабораторні, год.	
		20	16
		4) консультації, год.	
		10 год.	10 год.
		Самостійна робота, год.	
		80	110
		в тому числі 1) РГЗ та КР, год.	
		4	10
		2) курсова робота, год.	
		-	-
		Вид контролю:	
		Залік	Залік

### Примітка.

Співвідношення кількості годин аудиторних занять до загальної кількості годин (%):

для денної форми навчання – 47

для заочної форми навчання – 27

\* Відомості з навчального плану.

\*\* Структурна одиниця дисципліни (складається із змістових модулів).

Рекомендована кількість модулів дорівнює кількості контрольних точок.

# 1 МЕТА ТА ЗАВДАННЯ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Метою курсу є вивчення понять та методів організації доступу до даних на платформі .Net, а саме: технології доступу ADO.Net, особливостей реалізації платформи Entity Framework та технології об'єктно-реляційного мапінгу, використання технологій Nhibernate, Linq to SQL та Linq to Entities для абстрагування від логічної моделі даних, а також організації запитів з використанням сервісів WCF; набуття навичок практичної реалізації компонентів доступу до даних з використанням Visual Studio 2017. Також необхідно навчити студентів методам програмування баз даних на базі MS SQL Server, елементам автоматизації адміністрування баз даних, визначенню стратегії резервного копіювання та відновлення, а також можливостям використання засобів та інструментаріїв продукта MS SQL Server, які відносяться до підтримки баз даних та передачі даних; продемонструвати особливості роботи з розподіленими даними, призначення реплікації, забезпечення безпеки даних на рівні СУБД та технології підвищення високої доступності, а також виконання моніторингу продуктивності та активності SQL Server.

**Завдання:** За результатами вивчення дисципліни студенти повинні:

**знати:**

- поняття та методи обробки транзакцій;
- архітектуру застосувань з використанням технології ADO.NET;
- технологію об'єктно-реляційного зв'язування (мапінгу);
- особливості застосування Entity Framework для побудови концептуальної, логічної та фізичної моделей;
- можливості формування та налаштування моделі даних у відповідності з бізнес вимогами та технічними задачами;
- особливості використання традиційних об'єктів середовища CLR (POCO) з Entity Framework, вимоги для класів POCO;
- основні принципи розробки застосувань з використанням технології LINQ;
- особливості реалізації запитів до даних з використанням Linq to SQL та Linq to Entities для модифікації даних;

- основні поняття сервіс-орієнтованої технології та проектування, розробки й використання сервісу WCF для читання, модифікації та обробки даних;
- ролі технологій EntityFramework, WCFDataServices та ADO.NET для створення та обслуговування застосунків.

#### **Вміти:**

- планувати, створювати та управляти базами даних з використанням MS SQL Server;
- реалізовувати запити з використанням технології LINQ;
- розробляти високопродуктивні, масштабуєми застосунки ADO.NET, які здатні звертатися до даних та оновлювати їх;
- створювати моделі даних з урахуванням вимог технології об'єктно-реляційного зв'язування;
- виконувати задачі по зміні даних в моделі EDM;
- проектувати та реалізовувати запити до моделі EntityDataModel(EDM), з використанням LINQ to Entities, Entity SQL та класів з простору імен EntityClient;
- використовувати класи POCO сумісно з EntityFramework
- використовувати LINQ to SQL для побудови шару доступу до даних;
- проектувати, реалізовувати та використовувати прості служби даних WCF, які забезпечують читання та оновлення даних.

#### **Володіти:**

- навичками розробки та супроводження інформаційних систем;
- навичками адміністрування інформаційних систем.

**Змістовий модуль 1. Основи .NET Framework.**

1. **Вступна лекція.** Основні поняття. Основи .NET Framework. Модель виконання коду в середовищі CLR Характеристика методів доступу.
2. **Складання.** Спільно використовувані складання і складання зі строгим ім'ям
3. **Основи .NET Framework.** Делегати. Обробка та виклик подій

**Змістовий модуль 2. Традиційні технології доступу до даних**

4. **Технологія ADO.Net.** Характеристика методів доступу. Серверні веб-елементи управління ASP.NET
5. **Платформа динамічних даних ASP.NET**
6. **Елементи управління джерелами даних.**
7. **Використання засобів LINQ.**

**Змістовий модуль 3. Технології об'єктно-реляційного мапінгу**

8. **Технологія Entity Framework (EF).** Побудова моделі „Сутність-дані” (EDM)
9. **EF. Запити сутностей даних.** Створення, оновлення і видалення сутностей даних.
10. **Використання традиційних об'єктів середовища CLR (POCO) з EF.**
11. **Технологія NHibernate.** Створення NHibernate mapping для завантаження та збереження бізнес-об'єктів

**Змістовий модуль 4. Хмарні технології зберігання даних**

12. **Метод PaaS платформи Windows Azure**
13. **Міграція застосувань ASP.NET на Windows Azure.**
14. **Контроль доступу до бази даних SQL Azure.**
15. **Заклучна лекція.** Перспективи розвитку технологій зберігання, обробки та доступу до даних.



### 3 СТРУКТУРА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Назви змістових модулів і тем	Кількість годин											
	денна форма						заочна форма					
	усього	у тому числі					усього	у тому числі				
		л	п	лаб.	конс.	с. р.		л	п	лаб.	конс.	с. р.
1	2	3	4	5	6	7	8	9	10	11	12	13
Змістовий модуль 1. Основи .NET Framework.												
1. Вступна лекція.	6	2				4	6					6
2. Складання	12	2		4		6	12	1		4		7
3. Делегати.	12	2	2		2	6	8	1				7
Разом за змістовим модулем 1	30	6	2	4	2	16	26	2	0	4	0	20
Змістовий модуль 2. Традиційні технології доступу до даних												
4. Технологія ADO.Net.	13	2		4	2	5	15	2		4	2	7
5. Платформа динамічних даних ASP.NET	6	2				4	6					6
6. Елементи управління джерелами даних.	9	2			2	5	7				2	5
7. Використання засобів LINQ.	10	2	2			6	9		2			7
Разом за змістовим модулем 2	38	8	2	4	4	20	37	2	2	4	4	25
Змістовий модуль 3. Технології об'єктно-реляційного мапінгу												
8. Технологія Entity Framework (EF)	8	2	2			4	11	2			2	7
9. EF. Запити сутностей даних.	12	2		4	2	4	13			4	2	7
10. Використання традиційних об'єктів середовища CLR (POCO) з EF	10	2	2			6	9		2			7
11. Технологія NHibernate.	12	2		4		6	9				2	7
Разом за змістовим модулем 3	42	8	4	8	2	20	42	2	2	4	6	28

Змістовий модуль 4. Хмарні технології зберігання даних												
12. Метод PaaS платформи Windows Azure	1 0	2	2			6	10	2	2			6
13. Міграція застосувань ASP.NET на Windows Azure.	1 2	2		4		6	11			4		7
14. Контроль доступу до бази даних SQL Azure.	8	2			2	4	7					7
15. Заключна лекція.	6	2				4	7					7
Разом за змістовим модулем	3 6	8	2	4	2	20	35	2	2	4	0	27
Контрольна робота	4					4	10					10
Усього годин	1 5 0	30	10	20	10	80	150	8	6	16	10	110

#### 4 ТЕМИ ПРАКТИЧНИХ ЗАНЯТЬ

№ з/п	Назва теми	Кількість годин	
		денна	заочна
1	Створення делегатів і подій у власних класах	2	
2	Використання Linq to Sql для побудови шару доступу до даних	2	2
3	Використання CLR SQL	2	
4	Використання об'єктно-реляційного зв'язування при створенні моделей даних	2	2
5	Організація доступу до даних в Windows Azure	2	2
	<b>Разом</b>	<b>10</b>	<b>6</b>

#### 5 ТЕМИ ЛАБОРАТОРНИХ РОБІТ

№ з/п	Назва теми	Кількість годин	
		денна	заочна
1	Розробка бібліотеки класів із суворим ім'ям і її розміщення в GAC	4	4
2	Використання ADO.Net для читання та модифікації даних	4	4

3	Створення та використання ADO.Net Entity Data Model	4	4
4	Конфігурація NHibernate для доступу до локальної БД	4	
5	Міграція застосувань ASP.NET на Windows Azure.	4	4
	<b>Разом</b>	<b>20</b>	<b>16</b>

## 6 САМОСТІЙНА РОБОТА

№ з/п	Назва теми	Кількість годин	
		денна	заочна
1	Вивчення теоретичного матеріалу з використанням конспектів і навчальної літератури	98	104
2	Підготовка до практичних робіт	10	4
3	Підготовка до лабораторних робіт	10	4
4	Виконання домашньої контрольної роботи	4	10
5	Домени застосувань	6	6
6	Валідація даних	6	6
7	Захист даних	6	6
8	Запити до даних з використанням сервісів даних WCF	10	10
	<b>Разом</b>	<b>150</b>	<b>150</b>

## 7 ІНДИВІДУАЛЬНІ ЗАВДАННЯ

### 7.1 Розрахунково-графічні завдання (РГЗ) та контрольні роботи (КР)

Домашня контрольна робота представляє проектування веб-додатка, що демонструє роботу з базою даних, реалізованою в СУБД MS SQL Server та передбачає демонстрацію закріплення всього матеріалу з даної тематики.

№	Назва теми	Кількість годин	
		денна	заочна
1	Домашня контрольна робота	4	10
	Загальна кількість	<b>4</b>	<b>10</b>

Вимоги до веб-застосування:

- база даних має містити не менш ніж 3 пов'язані між собою заповнені таблиці (для демонстрації роботи підпрограм), по яких створені необхідні індекси;
- використання підходу Database First (при використанні Code First – треба навести обґрунтування вибору такого підходу);
- використання ORM;
- наявність валідації даних.

Тематика контрольної роботи обирається студентом довільно, можливо на підставі раніше виконаної курсової роботи з дисциплін “Бази даних” або «Аналіз та рефакторинг програмного забезпечення».

## 7.2 Курсова робота

Не передбачена планом

## 8 МЕТОДИ НАВЧАННЯ

1. Вивчення дисципліни «\*Робота з даними на платформи .Net» здійснюється традиційними методами із застосуванням новітніх інформаційних технологій, пояснення та демонстрація на багатьох практичних прикладах.

2. Теоретичні знання, що викладаються під час лекцій, використовуються на практичних і лабораторних роботах, що проводяться у комп'ютерних аудиторіях, які обладнані сучасними комп'ютерними засобами.

3. Методи стимулювання й мотивації навчально-пізнавальної діяльності студентів у виконанні власних проектів з практичної реалізації завдань дисципліни.

4. Методи контролю (самоконтролю, корекції (самокорекції), за ефективністю навчально-пізнавальної діяльності студента ці методи спрямовані на самостійну, творчу пізнавальну діяльність студентів, особливо при створенні власних проектів.

5. Універсальні методи поєднують самостійну роботу студентів під час практичних занять з інструктуванням, допомогою викладача, у результаті чого студенти набувають навичок самостійності та самостійною роботою студентів поза аудиторного навантаження. Крім цього студент має у своєму розпорядженні приклади розв'язання задач з роз'ясненнями – усе

це поєднується в наочно-ілюстративно-практичний комплект матеріалів для навчання.

## 10 МЕТОДИ КОНТРОЛЮ ТА РЕЙТИНГОВА ОЦІНКА ЗА ДИСЦИПЛІНОЮ

Як форма підсумкового контролю для дисципліни «\*Робота з даними на платформи .Net» використовується залік.

Контроль знань, які здобувають студенти внаслідок проведення усіх форм навчання, здійснюється шляхом поточного контролю на заняттях та опитувань під час проведення практичних та лабораторних робіт.

Програмований контроль здійснюється за допомогою бланкових тестів, або в системі Open Test.

### 10.1 Рейтингова оцінка за дисципліною

Залік може бути виставлений за роботу протягом семестру як сума оцінок з усіх контрольних заходів. При цьому виді контролю підсумкова оцінка  $O_{\text{сем}}$  (у 100-бальній системі) обчислюється за формулою  $O_{\text{сем}} = \sum_{i=1}^n O_{\text{ПРi}} + \sum_{j=1}^m O_{\text{ЛРj}} + O_{\text{ДКР}}$ , де  $O_{\text{ПРi}}$  – оцінка за кожне  $i$ -те практичне заняття;  $O_{\text{ЛРj}}$  – оцінка за кожну  $j$ -ту лабораторну роботу;  $O_{\text{ДКР}}$  – оцінка за домашню контрольну роботу. Оцінки наведені в таблиці «Рейтингова оцінка за дисципліною». Для оцінювання роботи студента протягом семестру підсумкова рейтингова оцінка  $O_{\text{сем}}$  розраховується як сума оцінок за різні види занять та контрольні заходи.

Кожне практичне заняття в 5 балів (1 бал за присутність та 4 бали за роботу на занятті). Лабораторні роботи оцінюються в 10 балів (1 бал за присутність + 4 бали за відпрацювання + 5 балів за захист). Для отримання максимальної оцінки необхідно в строк виконати завдання, вказані в методичних вказівках до лабораторної роботи та виконати самостійне завдання, аналогічне представленому в ході роботи (отримане у викладача), оформити звіт та захистити роботу.

Рейтингова оцінка за семестр у 100-бальній шкалі.

Вид заняття / контрольний захід	Рейтингова оцінка	
	мін.	Макс.
ЛБ № 1	6	10
Пз № 1	3	5
ЛБ № 2	6	10
Пз № 2	3	5
Пз № 3	3	5
ЛБ № 3	6	10
Пз № 4	3	5
ЛБ № 4	6	10
<b>Контрольна точка I</b>	<b>36</b>	<b>60</b>
Пз № 5	3	5
ЛБ № 5	6	10
Домашня контрольна робота	15	25
<b>Контрольна точка II</b>	<b>24</b>	<b>40</b>
<b>Всього за семестр</b>	<b>60</b>	<b>100</b>

Домашня контрольна робота оцінюється в 25 балів та представляє веб-застосування що демонструє роботу з базою даних, реалізованою в СУБД MS SQL Server та передбачає демонстрацію закріплення всього матеріалу курсу. Захист роботи виконується індивідуально кожним студентом.

Оцінка під час захисту залежить від:

1. Вміння студентом створити проект, подібний до того, що використовувався в контрольній роботі.
2. Вміння студентом пояснити усі тексти програми.
3. Вміння студентом порівнювати різні варіанти програмної реалізації.
4. Вміння студентом обґрунтувати висновки по роботі.

Перші 2 пункти пов'язані з перевіркою самостійності виконання завдання. Якщо студент не показує потрібних вмінь, то оцінка не може бути позитивною

При невиконанні роботи в строк знімається по 1 балу за кожний прострочений тиждень без поважних причин.

Максимальна рейтингова оцінка протягом семестру – 100 балів.

Якщо студент, який протягом семестру набрав не менш ніж 60 балів, не згоден з рейтинговим оцінюванням, він може скласти залік усній формі протягом залікового тижня (останнього тижня семестру), показавши знання з теоретичного та практичного матеріалу, що викладався в дисципліні, при цьому усі бали, набрані протягом семестру не зараховуються, тобто оцінка за залік не додається до накопичених балів, а ставиться за показанні в співбесіді вміння та знання.

## 10.2 Якісні критерії оцінювання

### **Необхідний обсяг знань для одержання позитивної оцінки.**

1. Архітектуру застосувань з використанням технології ADO.NET;
2. Технологію об'єктно-реляційного зв'язування (мапінгу);
3. Основні методи та засоби розробки інформаційних систем.
4. Володіти інформацією про проектування та реалізацію шару доступу до даних у застосуваннях, розроблених на платформі .Net.

### **Необхідний обсяг умінь для одержання позитивної оцінки.**

1. Вміти обирати модель застосувань.
2. Вміти використовувати технологію об'єктно-реляційного мапінгу.
3. Вміти приймати рішення щодо ефективності використання різних технологій доступу до даних.

### **Критерії оцінювання роботи студента протягом семестру.**

*Задовільно D, E (60-74).* Мати мінімум знань та умінь. Виконати та захистити усі лабораторні роботи. Виконати усі пункти індивідуального завдання.

*Добре C (75-89).* Твердо знати мінімум знань. Уміти використовувати ці знання при вирішенні практичних завдань. Виконати та захистити усі лабораторні роботи в строк. Відпрацювати практичні заняття.

*Відмінно A, B (90-100).* Знати усі теми. Уміти оцінювати ефективність різних засобів. Уміти обрати з різних засобів найбільш ефективні. Виконати та захистити усі лабораторні роботи в строк з отриманням найвищої оцінки. Відпрацювати практичні заняття. Виконати контрольну роботу на найвищу оцінку.

### Шкала оцінювання: національна та ECTS

Сума балів за всі види навчальної діяльності	Оцінка ECTS	Оцінка за національною шкалою	
		для іспиту, курсового проекту (роботи), практики	для заліку
96–100	A	відмінно	зараховано
90–95	B		
75–89	C	добре	
66–74	D	задовільно	
60–65	E		
35–59	FX	незадовільно з можливістю повторного складання	не зараховано з можливістю повторного складання
0-34	F	незадовільно з обов’язковим повторним вивченням дисципліни	не зараховано з обов’язковим повторним вивченням дисципліни

## 11 МЕТОДИЧНЕ ЗАБЕЗПЕЧЕННЯ ТА РЕКОМЕНДОВАНА ЛІТЕРАТУРА

### 11.1 Базова література

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#.– М.: «Русская Редакция», Спб: «Питер», 2013.– 896 с.

2. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5– Изд-во: «Вильямс», 2013.–1310 с.

3. Нортроп, Т. Основы разработки приложений на платформе Microsoft .NET Framework. Учебный курс Microsoft [Текст]/ Т.Нортроп, Ш.Уилдермьюс, Б.Райан – М.: «Русская Редакция», Спб: «Питер», 2007.– 864 с.

4. Ицик Бен-Ган. Microsoft SQL Server 2012: T-SQL Fundamentals: Пер. с англ.. – М.:Эксмо. 2015 – 400с.

### 11.2 Допоміжна література



5. Руководство по ASP.NET MVC 5 [Электронный ресурс] / Сайт METANIT.COM / Режим доступа: <http://metanit.com/sharp/mvc5/1.1.php> – 30.08.2017 р. – Загол. з екр.

6. Фримен А. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов [Текст] / Фримен А. / М.: Изд: Вильямс – 2015. 736 стр., с ил.; ISBN 978-5-8459-2008-9

7. Ицик Бен-Ган. Microsoft SQL Server 2012. Создание запросов. Учебный курс Microsoft Fundamentals. Пер. с англ.. – М.: Русская Редакция. 2014 – 720с.

8. Бондарь А. Microsoft SQL Server 2014– СПб: БХВ-Петербург. 2015 – 592 с.

9. Database Documentation / Microsoft - Режим доступа : [www/ URL https://docs.microsoft.com/ru-ru/sql/t-sql/language-reference](http://www/URLhttps://docs.microsoft.com/ru-ru/sql/t-sql/language-reference) (дата звернення 30.08.2017)

10. Accessing Data in Visual Studio / Microsoft Developer Network 2014 Режим доступа : [www/ URL https://msdn.microsoft.com/en-us/library/wzabh8c4\(v=vs.100\).aspx](http://www/URLhttps://msdn.microsoft.com/en-us/library/wzabh8c4(v=vs.100).aspx) - 30.08.2017 – Загл. з екрану

11. Електронна документація по SQL Server [https://msdn.microsoft.com/ru-ru/library/ms130214\(v=sql.120\).aspx](https://msdn.microsoft.com/ru-ru/library/ms130214(v=sql.120).aspx)

### 11.3 Методичні вказівки до різних видів занять

12. Методичні вказівки до практичних занять з дисципліни «\*Робота з даними на платформі .NET» для студентів усіх форм навчання напрямку підготовки 6.050103 – Програмна інженерія, спеціальності 121- Інженерія програмного забезпечення [Електронне видання]/ Упоряд. М.С. Широкопетлева –Харків: ХНУРЕ, 2017. - 29 с.

13. Методичні вказівки до лабораторних робіт з дисципліни «\*Робота з даними на платформі .NET» для студентів усіх форм навчання напрямку 6.010503 – Програмна інженерія, спеціальності 121- Інженерія програмного забезпечення [Електронне видання]/ Упоряд. М.С. Широкопетлева –Харків: ХНУРЕ, 2017. - 40 с.

## 12 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

1. MS SQL Server 2014, SSMS.
2. Visual Studio 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ  
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

## **МЕТОДИЧНІ ВКАЗІВКИ**

до практичних занять з дисципліни  
„\*Робота з даними на платформі .NET”

для студентів усіх форми навчання  
напряму підготовки 6.050103 – «Програмна інженерія»  
спеціальності 121 – «Інженерія програмного забезпечення»

Електронне видання

**ЗАТВЕРДЖЕНО**  
кафедрою «Програмної інженерії».  
Протокол № 4 від 17.10.2017 р.

ХАРКІВ 2017

Методичні вказівки до практичних занять з дисципліни «\*Робота з даними на платформі .NET для студентів усіх форми навчання напряму підготовки 06.050103 – «Програмна інженерія», спеціальності 121 – «Інженерія програмного забезпечення» [Електронне видання] / Упоряд. М.С. Широкопетлева. – Харків: ХНУРЕ, 2017. – 29 с.

Упорядник: М.С.Широкопетлева

Рецензент

© ХНУРЕ, 2017 рік

## ЗМІСТ

Вступ	4
1 Створення делегатів і подій у власних класах	5
1.1 Мета заняття	5
1.2 Методичні вказівки з організації самостійної роботи студентів	5
1.3 Контрольні запитання і завдання	7
1.4 Приклади аудиторних і домашніх завдань	7
2 Використання Linq to Sql для побудови шару доступу до даних	10
2.1 Мета заняття	10
2.2 Методичні вказівки з організації самостійної роботи студентів	11
2.3 Контрольні запитання	11
2.4 Приклади аудиторних і домашніх завдань	11
3 Використання CLR SQL	16
3.1 Мета заняття	16
3.2 Методичні вказівки з організації самостійної роботи студентів	16
3.3 Контрольні запитання і завдання	17
3.4 Приклади аудиторних і домашніх завдань	17
4 Використання об'єктно-реляційного зв'язування при створенні моделей даних	19
4.1 Мета заняття	19
4.2 Методичні вказівки з організації самостійної роботи студентів	20
4.3 Контрольні запитання	20
4.4 Приклади аудиторних і домашніх завдань	20
5 Організація доступу до даних в Windows Azure	24
5.1 Мета заняття	24
5.2 Методичні вказівки з організації самостійної роботи студентів	24
5.3 Контрольні запитання	25
5.4 Приклади аудиторних і домашніх завдань	25

## ВСТУП

Дисципліна "Робота з даними на платформі .NET" має велику роль в підготовці фахівців в області проектування та розробки застосунків для обробки даних на платформі .Net. Ця дисципліна є однією з дисциплін, необхідних розробнику застосунків на платформі .Net.

У Microsoft .NET Framework з'явилося багато нових концепцій, технологій і термінів. Мета цього курсу- вивчення понять та методів організації доступу до даних на платформі .Net Framework. Також в цьому курсі висвітлені питання програмування хостосувачів з використанням баз даних, розроблених на підставі СУБД MS SQL Server.

Основне призначення даної дисципліни – засвоєння студентами технологій доступу до даних на основі платформи Microsoft .NET Framework та заснованих на цих технологій при проектуванні та розробці програмних застосунків.

Метою практичних занять студентів в ході вивчення курсу "Робота з даними на платформі .NET" є освоєння понять та принципів технологій доступу до даних, що використовуються при створенні захищених ефективних застосунків на платформі .Net

Для вивчення даної дисципліни потрібні знання в області програмування, об'єктно-орієнтованого програмування, баз даних, етапів проектування інформаційних систем.

# 1 СТВОРЕННЯ ДЕЛЕГАТИВ І ПОДІЙ У ВЛАСНИХ КЛАСАХ

## 1.1 Мета заняття

Ознайомитися з поняттями делегатів, набути практичні навички створення та роботи з делегатами

.

## 1.2 Методичні вказівки з організації самостійної роботи студентів

Необхідно ознайомитися з матеріалами лекцій і 1 [с. 434-463; 2, 386-413] .

Продумуючи архітектуру чергового класу може виникнути необхідність передачі в якості аргументу шматка виконуваного коду. Це дозволило б користувачам уникнути вервечки «if – iv» та «case – iv» і зробило б код більш елегантним.

Делегат являє собою об'єкт, який може посилатися на метод. Викликаючи делегат, викликаємо функцію на яку він вказує. Отже, коли створюється делегат, то в підсумку виходить об'єкт, що містить посилання на метод. Більш того, метод можна викликати за цим посиланням. Іншими словами, делегат дозволяє викликати метод, на який він посилається.

По суті, делегат - це безпечний щодо типів об'єкт, який вказує на інший метод (або, можливо, список методів) додатка, який може бути викликаний пізніше. Зокрема, об'єкт делегата підтримує три важливих фрагмента інформації:

- адреса методу, на якому він викликається;
- аргументи (якщо є) цього методу;
- повертається значення (якщо є) цього методу.

Як тільки делегат створений і забезпечений необхідною інформацією, він може динамічно викликати методи, на які вказує, під час виконання. Кожен делегат в .NET Framework (включаючи спеціальні делегати) автоматично забезпечується здатністю викликати свої методи синхронно або асинхронно. Цей факт значно спрощує завдання програмування, оскільки дозволяє викликати метод у вторинному потоці виконання без ручного створення і управління об'єктом Thread.

Тип делегата оголошується за допомогою ключового слова delegate. Нижче наведена загальна форма оголошення делегати:

`delegate повертаємий_тип ім'я (перелік_параметрів);`

де `повертаємий_тип` позначає тип значення, що повертається методами, які будуть викликатися делегатом;

`ім'я` - конкретне ім'я делегата;

`перелік_параметрів` - параметри, необхідні для методів, що викликаються делегатом. Як тільки буде створений екземпляр делегата, він може викликати і посилатися на ті методи, повертається тип і параметри яких відповідають зазначеним в оголошенні делегата.

Коли компілятор `C #` обробляє тип делегата, він автоматично генерує запечатаний (sealed) клас, успадкований від `System.MulticastDelegate`. Цей клас (у поєднанні з його базовим класом `System.Delegate`) надає необхідну інфраструктуру для делегата, щоб зберігати список методів, які підлягають виклику в більш пізній час.

Кожен делегат має три загальнодоступних методів. `Invoke ()` - використовується для синхронного виклику кожного з методів, підтримуваних об'єктом делегата; це означає, що викликаючий код повинен очікувати завершення виклику, перш ніж продовжити свою роботу.

Методи `BeginInvoke ()` і `EndInvoke ()` пропонують можливість виклику поточного методу асинхронним чином, в окремому потоці виконання.

В таблиці 1.1 описані основні члени, загальні для всіх типів делегатів:

Таблиця 1.1 - Методи і властивості делегатів `C #`

Член	Призначення
Method	Повертає об'єкт <code>System.Reflection.Method</code> , який представляє деталі статичного методу, підтримуваного делегатом
Target	Якщо метод є екземпляр ним, то <code>Target</code> повертає об'єкт, що представляє метод, підтримуваний делегатом. Якщо значення, повернене <code>Target</code> , дорівнює <code>null</code> , значить, метод є статичним
Combine()	Додає метод в список, підтримуваний делегатом, у <code>C #</code> можна використовувати перевантажену операцію <code>+</code>
GetInvocationList()	Повертає масив типів <code>System.Delegate</code> , кожен з яких представляє певний метод, доступний для виклику
Remove() RemoveAll()	Видаляють метод (або всі методи) зі списку дзвінків делегата.

Делегати мають наступні властивості:

- делегати схожі на покажчики функцій в C ++, але є типобезпечними;
- делегати дозволяють виробляти передачу методів подібно звичайним параметрам;
- делегати можна використовувати для визначення методів зворотного виклику;
- делегати можна пов'язувати один з одним; наприклад, при появі однієї події можна викликати кілька методів;
- точна відповідність методів типом делегата не потрібно.

### 1.3 Контрольні запитання і завдання

1. Що таке делегат ?
2. Яка з команд дозволяє створити делегат , який визначає метод , який приймає один параметр типу `IntPtr` і повертає значення `void` ?
3. Для яких типів підтримується коваріантність і контрваріантність при прив'язці методу до делегату ?
4. Мається делегат виду `delegate Object MyCallback ( FileStream s ) ;`
5. Як може виглядати прототип методу , пов'язаного з цим делегатом ?
6. Що таке ланцюжок делегатів ?
7. Яка з команд дозволить вилучити делегат `MyDelegate` з ланцюжка раніше створених делегатів ?
8. Яким чином можна визначити метод , на який посилається делегат ?
9. За допомогою якого методу класу `MultiCastDeleGate` можна викликати будь-який з делегатів ланцюжка ?
10. Створити ланцюжок делегатів. Додати до створеного ланцюжка новий делегат. Викликати один делегатів з ланцюжка (останній в ланцюгу, довільний). Видалити один з делегатів. Показати прийоми усунення недоліків ланцюжка з використанням методу `GetInvocationList`.
11. Продемонструвати використання узагальнених делегатів і власних (при неможливості використання узагальнених). Зробити висновок про необхідність створення власних делегатів.

### 1.4 Приклади аудиторних і домашніх завдань

#### 1.4.1 Продемонструвати роботу з делегатами:

Опишемо делегати та ланцюжок делегатів:



```
using System;
namespace TestDelegate
{
    delegate Int32 MyDelegate(Int32 a, Int32 b);
    delegate double OperationDelegate(double a, double b);
    delegate void ChainingDelegate(ref Int32[] arr);
}
```

В класі SimpleOperation необхідно створити наступні методи:

- void CountTwoInArr(int[] arr) – друкує кількість парних елементів масива;
- double Summa(double a, double b) – повертає суму двох чисел;
- double Mult(double a, double b) – повертає доданок двох чисел;
- double Divide(double a, double b) – повертає частку від ділення двох чисел;

#### 1.4.2 Створення події

В класі ArrOperation створимо подію

```
using System;
namespace TestDelegate
{
    public delegate void EventDelegate(int[] arr);

    class ArrOperation
    {
        public event EventDelegate MyEventExecute;
        public void ExecuteEvent(int[] arr)
        {
            if(MyEventExecute != null) MyEventExecute(arr);
        }
    }
}
```

Також в цьому класі створимо наступні методи:

- void fill(ref int[] mas) – для ініціалізації масива;
- void print(ref int[] mas) – друку елементів масива;
- void ArrTwo(ref int[] arr ) для визначення парних елементів масива;
- void ArrNoTwo (ref int[] arr ) для визначення непарних елементів масива;

### 1.4.3 Демонстрація виклику делегатів та подій

Для демонстрації роботи делегатів створимо метод Main:

```
static void Main(string[] args)
{
    Console.WriteLine("---Статичні методи-----");
    MyDelegate del1 = Summa;
    int answer = del1(10, 5);
    Console.WriteLine("Сума: " + answer);
    del1 = Mult;
    answer = del1(25, 5);
    Console.WriteLine("-----Екземплярні методи----");
    SimpleOperation simp = new SimpleOperation();
    opr = simp.Mult;
    result = opr(15.4, 4.7);
    Console.WriteLine("Доданок: " + result);
    opr = simp.Divide;
    result = opr(75.7, 50.2);
    Console.WriteLine("Частка " + result);
    Console.WriteLine("\n-Виклик ланцюжка делегатів");
    int[] arr = new int[10];
    ArrOperation arrOp = new ArrOperation();
    /* Створимо ланцюжок */
    ChainingDelegate chaining;
    ChainingDelegate cd1 = arrOp.fill;
    ChainingDelegate cd2 = arrOp.ArrTwo;
    ChainingDelegate cd3 = arrOp.ArrNoTwo;
    ChainingDelegate cd4 = arrOp.print;
    chaining = cd1;
    chaining += cd4;
    chaining += cd2;
    chaining += cd4;
    chaining += cd3;
    chaining += cd4;
    chaining(ref arr);
    chaining -= cd4;
    chaining(ref arr); // Виклик ланцюжка
    //Виклик довільного делегата з ланцюжка
    for (int i = 0; i < chaining.GetInvocationList().Length; i++)
    {
        switch (i)
        {
            case 2:
```

```

        cd2.Invoke(ref arr);
        break;
    }
}
Console.WriteLine("\n-----Демонстрація виклика подій --");
arrOp.fill(ref arr);
arrOp.print(ref arr);
// Додавання обробників до події
arrOp.MyEventExecute += new EventDelegate(simp.CountTwoInArr);
arrOp.MyEventExecute += new EventDelegate(CountNoTwoInArr);
arrOp.ExecuteEvent(arr);
Console.ReadKey();
}
}
}

```

#### 1.4.4 Завдання для самостійного виконання

У власній збірці створити делегат. Продемонструвати коваріативність та / або контрваріативність посилаєльних типів при прив'язці методу до делегату. Продемонструвати використання делегатів для виклику статичних і екземплярні методів. Розібрати класи з використанням утиліти ILDasm.exe. Продемонструвати тотожність виклику делегатів з використанням методу Invoke() та без нього.

У ході виконання роботи використовувати спрощення при роботі з делегатами.

## 2 ВИКОРИСТАННЯ LINQ TO SQL ДЛЯ ПОБУДОВИ ШАРУ ДОСТУПУ ДО ДАНИХ

### 2.1 Мета заняття

Ознайомитися з особливостями технології LINQ TO SQL, набути практичних навичок з використання LINQ TO SQL для побудови шару доступу до даних.

## 2.2 Методичні вказівки з організації самостійної роботи студентів

Ознайомитися з матеріалами лекцій. Розгляньте різні області застосування LINQ TO SQL. Рекомендується практичне завдання виконувати з використанням комп'ютерної техніки.

### 2.3 Контрольні запитання

1. Охарактеризуйте достоїнства та недоліки LINQ to SQL
2. Як використовують технологію LINQ to SQL для модифікування даних?
3. Наведіть основні класи LINQ to SQL.
4. Який тип відображення даних використовується у технології LINQ to SQL?

### 2.4 Приклади аудиторних і домашніх завдань

Для роботи з LINQ to SQL в проект потрібно додати посилання на дві збірки: System.Data.Linq і Microsoft.SqlServer.Types . Якщо з першою бібліотекою проблем немає ( її можна знайти на вкладці « . NET » форми « Add Reference » - додавання посилання на використовувану в проекті бібліотеку ) , то друга потрібно буде пошукати в директорії "C: \ Program Files \ Microsoft SQL Server \ 100 \ SDK \ Assemblies \ ». Для того , щоб остання збірка надалі відображалася у вкладці « . NET » форми додавання зборок , потрібно її зареєструвати один раз за допомогою утиліти gacutil .

2.4.1 Перший крок у використанні LINQ to SQL - це створення класів - відображень для таблиць бази даних.

На одну таблицю - один клас.

```
using System ;
using System.Data.Linq.Mapping ;
using Microsoft.SqlServer.Types ;
namespace MyNamespace
{
    [ Table ( ) ]
    public sealed class Boundaries_Country
    {
```

```

[ Column ( AutoSync = AutoSync.OnInsert , DbType = "
uniqueidentifier " , IsPrimaryKey = true , IsDbGenerated = true ,
UpdateCheck = UpdateCheck.Never )]
public Guid FeatureID ;
[ Column ( DbType = " varchar ( 100 )", CanBeNull = false )]
public string CountryName ;
[ Column ( DbType = " geography " , CanBeNull = false )]
public SqlGeography CountryBoundary ;
}
}

```

Над оголошенням класу і над полями розставлені атрибути . Наприклад , у рядку 7 , атрибут Table вказує , що цей клас асоційований з таблицею в базі даних. Якщо ім'я класу збігається з ім'ям таблиці , то атрибут можна записувати так , як в прикладі, а якщо ні , то потрібно буде вказати додаткову властивість Name: [ Table (Name = « Boundaries\_Country » )] .

Ще один клас потрібен для створення контексту бази даних. Ми можемо використовувати вже наявний DataContext , але краще зробити свого спадкоємця для суворої типізації .

```

using System.Data.Linq ;
namespace MyNamespace
{
public class ExampleDatabase : DataContext
{
public Table <Boundaries_Country> BoundariesCountry ;
public ExampleDatabase ( string connectionString )
: Base ( connectionString )
{
}
}
}

```

**2.4.2 Приклад, виберемо з бази даних інформацію , для яких назва країни починалося з літери « С».**

```

static void Main ( string [] args )
{
ExampleDatabase db = new ExampleDatabase ( @ "... " ) ;

var q = from item in db.BoundariesCountry
where item.CountryName.StartsWith ( " C " )

```

```

select item ;

foreach ( var item in q )
Console.WriteLine ( item.CountryName ) ;
}

```

Якщо в режимі налагодження зупинити виконання програми на рядку 9 і побачити вміст змінної `q` , то ми побачимо сформований LINQ to SQL запит.

### 2.4.3 Збережені процедури

Для вибірки геометричних фігур за критерієм потрапляння в заданий прямокутник для однієї конкретної таблиці, досить створити таку просту процедуру, що зберігається .

```

CREATE PROCEDURE [ dbo ] . [ Sp_bbx_Boundaries_Country ]
@ boundingBox varbinary ( max )
AS
BEGIN
SET NOCOUNT ON ;
SELECT *
FROM dbo.Boundaries_Country
WHERE GEOGRAPHY :: STGeomFromWKB ( @ boundingBox , 4326 ) .
STIntersects ( CountryBoundary ) = 1 ;
RETURN ;
END

```

Вхідний параметр - це прямокутник ( заданий полігоном ) в WKB - форматі. У рядку 8 він перетвориться статичним методом `STGeomFromWKB` в об'єкт типу даних `geography` і вже на ньому викликається функція `STIntersects` , що здійснює перевірку на потрапляння конкретної кордону в прямокутник.

У програмі , в класі , реалізующем `DataContext` (у нас цей клас називається `ExampleDatabase` ) , опишемо обгортку для виклику цієї процедури.

```

[ Function ( ) ]
public ISingleResult <Boundaries_Country>
sp_bbx_Boundaries_Country (
[ Parameter ( DbType = " varbinary ( max ) " ) ] byte []
boundingBox )
{
IExecuteResult execResult = this.ExecuteMethodCall ( this , ( (
MethodInfo )
( MethodInfo.GetCurrentMethod ( ) ) ) , boundingBox ) ;
ISingleResult <Boundaries_Country> result =

```

```
( ( ISingleResult <Boundaries_Country> ) execResult.ReturnValue )
;
return result ;
}
```

Тут , також як і для таблиць , описуються атрибути для функції і параметрів .

У рядку 4 викликається збережена процедура і результат зберігається в `execResult` , потім , в рядку 5 , він перетвориться до необхідному типу даних і повертається в основну програму.

Користуємося наступним чином :

```
var q = from item in db.sp_bbx_Boundaries_Country (
sqlEnvelope.STAsBinary ( ) . Buffer )
where item.CountryName.StartsWith ( " C " )
select item ;
foreach ( var item in q )
Console.WriteLine ( item.CountryName ) ;
```

#### 2.3.4 Виведення результату на консоль.

Зауваження , якщо в робочому проекті , декілька таблиць з географічними даними , то можливі наступні варіанти .

1) Для кожної таблиці створити свою збережену процедуру, а в програмі для кожної процедури, що зберігається свою функцію - обгортку . Можна спростити життя користувачу арі , якщо написати «центральный» generic - метод в якому , по актуальному типу , використовуваному при виклику generic - методу , будуть викликатися приватні метод - обгортки збережених процедур і виконуватися необхідні приведення типів .

2) Написати одну збережену процедуру з використанням динамічного sql . У програмі потрібно буде зробити один generic - метод , з якого також , як і в попередньому варіанті будуть викликати спеціалізовані (по типу даних) методи - обгортки навколо однієї і тієї ж процедури.

#### 2.3.5 Table - valued функції

Table - valued функція , витягає з таблиці бази даних записи за критерієм потрапляння в заданий регіон , може бути створена таким чином.

```
CREATE FUNCTION [ dbo ] . [ F_bbx_Boundaries_Country ]
(
@ boundingBox varbinary ( max )
```

```

)
RETURNS TABLE
AS
RETURN
(
SELECT *
FROM dbo.Boundaries_Country
WHERE GEOGRAPHY :: STGeomFromWKB (
@ boundingBox , 4326 ) . STIntersects ( CountryBoundary ) = 1
)

```

Тобто за змістом повністю аналогічно збереженій процедурі , описаній раніше.

Для функції також потрібно буде створити свою обгортку .

```

[ Function ( IsComposable = true )]
public IQueryable <Boundaries_Country> f_bbx_Boundaries_Country (
[ Parameter ( DbType = " varbinary ( max ) " )] byte []
boundingBox )
{
return this.CreateMethodCallQuery <Boundaries_Country> (this,
( ( MethodInfo ) ( MethodInfo.GetCurrentMethod ( ))) , boundingBox
) ;
}

```

В атрибуті методу вказуємо властивість IsComposable , яке говорить , що зараз ми будемо запускати функцію на SQL Server , а не збережену процедуру . Для виклику функції використовується метод CreateMethodCallQuery .



## 3 ВИКОРИСТАННЯ CLR SQL

### 3.1 Мета заняття

Ознайомитися з особливостями SQLCLR, набути практичних навичок з використання SQLCLR.

### 3.2 Методичні вказівки з організації самостійної роботи студентів

Ознайомитися з матеріалами лекцій. Розгляньте різні області застосування SQLCLR. Рекомендується практичне завдання виконувати з використанням комп'ютерної техніки.

За допомогою SQLCLR можна створювати такі об'єкти:

- збережені процедури ;
- функції визначені користувачем , які повертають одиничне значення ;
- функції визначені користувачем , які повертають таблицю і можуть бути викликані з FROM , JOIN , APPLY
- тригери ( DML , DDL і тригери за логіном )
- агрегати певні користувачем
- типи певні користувачем.

CLR Функції визначені користувачем , які повертають одиничне значення відрізняються від збережених процедур лише тим , що тип який повертає функція, описаний в коді НЕ void , а деякий Sql тип ( наприклад SqlBoolean ) .

```
[ SqlFunction ( IsDeterministic = true , IsPrecise = true ) ]
public static SqlBoolean IsRegExMatch ( SqlString input ,
SqlString pattern )
{
    using ( SqlConnection conn = new SqlConnection ( " context
connection = true " ))
    {
        if ( input.IsNull || pattern.IsNull )
            return SqlBoolean.Null ;
        return ( SqlBoolean ) Regex.IsMatch ( input.Value ,
pattern.Value ) ;
    }
}
```

### 3.3 Контрольні запитання і завдання

1. Що таке SQLCLR?
2. У чому полягає перевага SQLCLR над T- SQL?
3. Опишіть основні кроки для використання CLR всередині БД
4. Напишіть SQLCLR функцію , яка б повертала кількість CreditCard , у яких не було б у номері цифри 14 з таблиці Sales .
5. Напишіть CLR функцію , яка б повертала б введену пропозицію у вигляді таблиці, де кожне слово - елемент в таблиці .
6. Напишіть свій CLR агрегат , який би складав цілі числа.

### 3.4 Приклади аудиторних і домашніх завдань

Розглянемо простий приклад: напишемо користувача функцію розрізання рядків по роздільнику, використовуючи SQL синтаксис та SQL CLR на базі C # і порівняємо результати.

Сценарій реалізації :

Для використання CLR всередині бази даних необхідно провести наведені нижче дії :

Створіть сутність SQL сервера, яка б дозволила запускатися CLR коду.

Напишіть код (на C # або Visual Basic .NET) , який виконує дії з об'єктами бази даних.

Скомпілюйте написаний код в збірку за допомогою CLR компілятора.

Завантажте збірку в SQL сервер.

Створіть об'єкт в базі даних і точку входу в збірку , використовуючи Data Definition Language ( DDL ) .

Проробимо всі ці кроки над конкретним прикладом. Запустимо SQL Server Management Studio (SSMS), використовуємо базу даних AdventureWorks. Створимо просту процедуру, яка б обирала дані заданого користувача

```
CREATE PROCEDURE Sales.spCustomerGet
@ CustomerID INT
AS
BEGIN
SELECT * FROM Sales.Customer
WHERE CustomerID = @ CustomerID
END
```

Однак з точки продуктивності краще використовувати SQLCLR , для створення такої процедури. Для початку потрібно дати доступ SQLCLR коду для запуску на SQL сервері. Для цього запустіть T- SQL код:

```
EXEC sp_configure 'clr_enabled' , 1 ;
```

Перший параметр ' clr\_enabled ' - параметр SQL Server, який треба змінити.

Другий параметр 1 - значення , яке необхідно задати першому параметру . В прикладі 1 виступає в ролі ідентифікатора true.

```
RECONFIGURE ;
```

Перший рядок дає доступ до CLR на сервері. Другий рядок застосовує зміни на сервері.

Далі створимо файл CLRStoredProc.cs, в якому і буде зберігатися код збереженої процедури , написаний мовою C #. Нижче приведений код файлу CLRStoredProc.cs :

```
using System ;
using System.Data.SqlTypes ;
using System.Data.SqlClient ;
using Microsoft.SqlServer.Server ;

namespace CLRProc
{
    class ExampleProcedure
    {
        public static void CustoeerGetProcedure (SqlInt32
customerId)
        {
            using (SqlConnection conn = new SqlConnection("context
connection = true"))
            {
                SqlCommand cmd = conn.CreateCommand();
                cmd.CommandText = @ "SELECT * FROM Sales.Customer
WHERE CustomerID = @CustomerID " ;
                cmd.Parameters.AddWithValue ("@ CustomerID",
customerId) ;

                conn.Open ();
                SqlContext.Pipe.ExecuteAndSend (cmd) ;
            }
        }
    }
}
```

Тепер необхідно скомпілювати код в CLR збірку. Для цього використаємо ідкриємо компілятор csc.exe, який можна знайти в директорії C:\Windows\Microsoft.NET\Framework\.... Для того , щоб скомпілювати код , запустіть команду з директорії , де зберігається ваш файл CLRStoredProc.cs :

```
csc.exe / target : library / out : CLRStoredProc.dll  
CLRStoredProc.cs
```

Буде створена бібліотека CLRStoredProc.dll , яку слід помістити на сервер в базу даних. Це можна зробити за допомогою T- SQL команди , яку можна запустити з SSMS:

```
USE AdventureWorks ;  
CREATE ASSEMBLY ExampleProcedure  
FROM 'C : \ CLRStoredProc.dll ' ;
```

Наприкінці необхідно буде створити збережену процедуру яка б запускала написаний нами код на C # з допомогою T- SQL команди

```
CREATE PROCEDURE Sales.spCustomerGetClr  
@ CustomerID INT  
AS  
EXTERNAL NAME ExampleProcedure . " ExampleProcedure " .  
CustomerGetProcedure ;  
GO
```

Процедура створена. Для виклику використовуйте T- SQL команду в SSMS :

```
EXEC Sales.spCustomerGetClr @ CustomerID = 1 ;
```

## 4 ВИКОРИСТАННЯ ОБ'ЄКТНО-РЕЛЯЦІЙНОГО ЗВ'ЯЗУВАННЯ ПРИ СТВОРЕННІ МОДЕЛЕЙ ДАНИХ

### 4.1 Мета заняття

Ознайомитися з особливостями використання об'єктно-реляційного зв'язування при створенні моделей даних. Набути практичних навичок зі створення застосувань з використанням підходів DataBaseFirst, Model First та Code First.

## 4.2 Методичні вказівки з організації самостійної роботи студентів

Ознайомитися з матеріалами лекцій. Розгляньте різні області застосування об'єктно-реляційного зв'язування. Рекомендується практичне завдання виконувати з використанням комп'ютерної техніки.

### 4.3 Контрольні запитання

1. Охарактеризуйте технологію об'єктно-реляційного зв'язування.
2. В яких випадках доцільно використовувати зв'язування типу „один-до-багатьох”
3. Яким чином можна виконати зв'язування програмованих об'єктів БД?
4. Які недіоліки існують у технології об'єктно-реляційного зв'язування? Яким чином їх можна подолати?
5. Наведіть приклади переваг використання об'єктно-реляційного зв'язування перед технологією Linq.

### 4.4 Приклади аудиторних і домашніх завдань

#### 4.4.1 Використання підходу DataBase First.

Database First був першим підходом, який з'явився в Entity Framework. Даний підхід багато в чому схожий на Model First і підходить для тих випадків, коли розробник вже має готову базу даних. Щоб Entity Framework міг отримати доступ до бази даних, в системі повинен бути встановлений відповідний провайдер. Так, Visual Studio вже підтримує відповідну інфраструктуру для СУБД MS SQL Server.

Створимо проект за шаблоном ConsoleApplication.

Далі слід сгенерувати модель (див. рис. 4.1).

Після цього нам відкриється вікно майстра створення моделі (рис.4.2). Треба обрати «Generate from database».

Далі відкриється вікно наступного кроку по створенню моделі, на якому треба буде встановити підключення до бази даних. У нашому випадку база даних містить тільки одну таблицю. Автоматично створюється connection string і записується в App.Config під ім'ям, яке вказуєш. Далі треба обрати таблиці (рис.4.3), на підставі яких створюється модель.

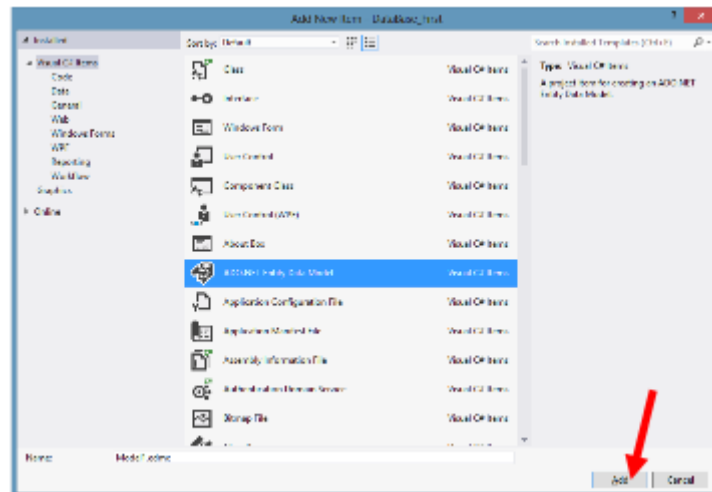


Рисунок 4.1 – Додавання нової моделі

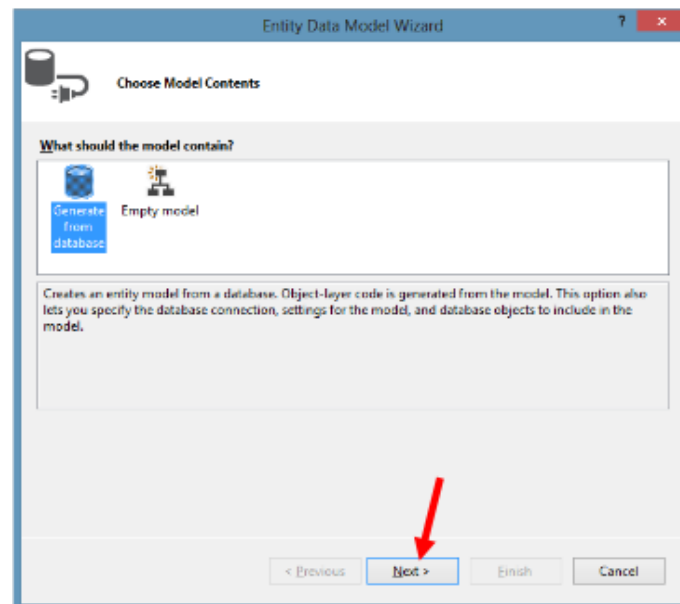


Рисунок 4.2 – Вибір опції

В результаті буде створена модель, яка відкриється у вікні дизайнера (рис.4.4). Щоб отримати доступ до моделі, потрібно створити екземпляр класу, який автоматично був згенерований з ім'ям, яке було зазначено на попередньому кроці. Після виділення сутності в Solution Explorer можна побачити властивості для цієї сутності.

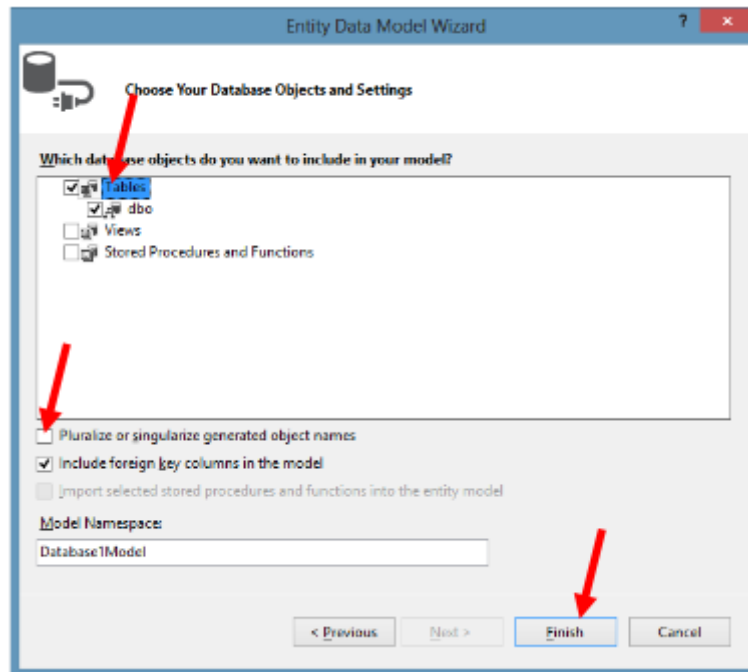


Рисунок 4.3 – Вибір таблиць

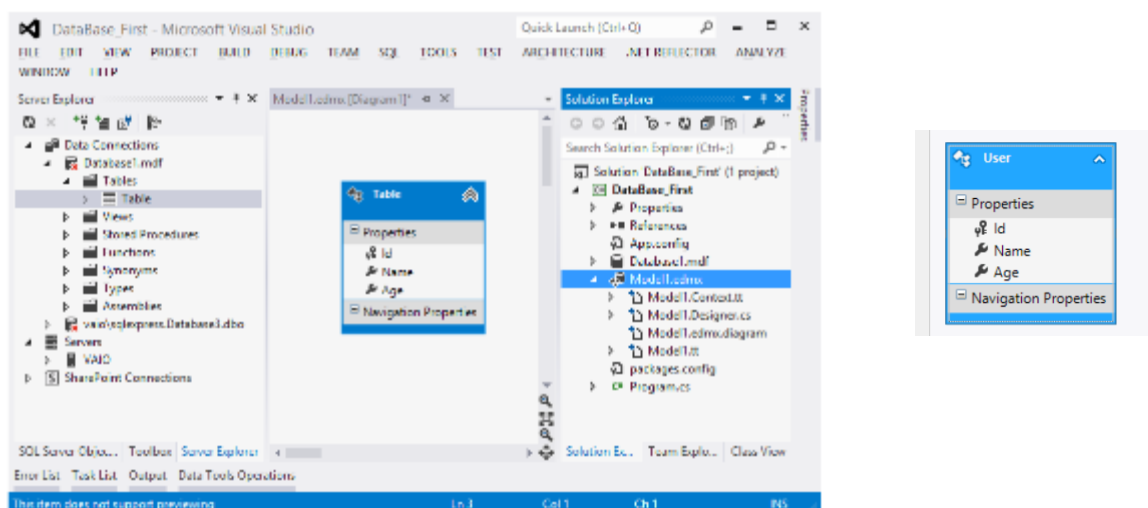


Рисунок 4.4 – Модель у вікні дизайнера

В наступному прикладі коду отримаємо доступ до всіх елементів таблиці Table та виведемо в консоль ім'я.

```
class Program
{
    static void Main()
    {
        Database1Entities db = new Database1Entities();
        var query = from t in db.Table select t;
        foreach (var item in query)
```

```
{
Console.WriteLine(item.Name);
}
}
}
```

В результаті отримаємо перелік усіх значень поля Name.

#### 4.4.2 Використання підходу Model First.

Model First представляє ще один підхід до роботи з Entity Framework. Суть даного підходу полягає в тому, що спочатку робиться модель, а потім по ній створюється база даних.

Отже, створимо новий проект по типу Console Application. І потім додамо в проект новий елемент. Натиснемо правою кнопкою миші на проект у вікні Solution Explorer і в списку, що з'явився виберемо Add -> New Item. І потім у вікні додавання нового елемента виберемо ADO.NET Entity Data Model/

Оскільки модель буде описувати людини, то назвемо її User. Натиснемо ОК і нам відкриється майстер створення моделі.

Вікно пропонує чотири варіанти створення моделі, з яких нам треба вибрати Empty EF Designer Model. Натиснемо кнопку Finish, після чого відкриється порожнє вікно створення моделі.

Перетягнемо на це поле з панелі Toolbox (Панель Інструментів) в лівій частині елемент Entity. Тепер на поле створення моделі є невелика схема майбутньої моделі, в якій зараз за замовчуванням вказано лише одне поле - Id. По-перше, перейменуємо сутність. За умовчанням вона називається Entity1. Виділимо схему і перейдемо до вікна властивостей.

Відредагуємо значення властивості Name на User (ім'я сутності). Також змінимо значення властивості Entity Set Name на Users (назва набору об'єктів User).

Далі створимо дві властивості для імені і віку. Отже, виділимо схему суті і натиснемо на праву кнопку миші. У випадаючому списку виберемо Add New -> Scalar Property. Після цього буде додано нову властивість (Scalar Property має на увазі властивості на основі найпростіших типів int, float, string, тощо). Додамо два властивості - Name і Age. За замовчуванням всі властивості, що додаються, мають тип string. Однак можемо змінити тип у вікні властивостей.

Таким чином, отримаємо схему, аналогічну схемі, наведеній на рис.4.4.

Після створення діаграми моделі перебудуємо проект за допомогою опції Rebuild.



Тепер за моделлю можна згенерувати код і базу даних. Спочатку сгенерируем код моделі. Для цього натиснемо на діаграму моделі правою кнопкою миші і виберемо пункт Add Code Generation Item. Далі буде запропоновано вибрати версію EF. Після цього в структурі проекту можна побачити вузол User.tt, який в якості підвузла міститиме клас моделі у файлі User.cs.

Далі згенеруємо базу даних за отриманою моделлю. Натиснемо на діаграму моделі правою кнопкою миші і в випадаючому списку виберемо Generate Database from Model. З використанням майстра створення підключення налаштуємо підключення до БД. Як ім'я бази даних введемо usersdb, а в якості сервера: (localdb)\v11.0. Натиснемо OK і потім Visual Studio встановить в якості підключення моделі тільки що створену базу даних.

Після цього буде згенеровано скрипт бази даних. Натиснемо Finish і автоматично відкриється в Visual Studio файл скрипта User.edmx.sql. По завершенні треба буде запустити цей скрипт (кнопка Execute).

При відкритті вікна Database Explorer (його можна відкрити, вибравши в меню View-> Other Windows), можна побачити базу даних. А розкривши вузол, також побачимо, що вона містить всю ту схему, яку визначили в моделі:

## 5 ОРГАНІЗАЦІЯ ДОСТУПУ ДО ДАНИХ В WINDOWS AZURE

### 5.1 Мета заняття

Ознайомитися з особливостями доступу до даних в Windows Azure, набути практичних навичок зі створення застосувань з використанням хмарних сервісів.

### 5.2 Методичні вказівки з організації самостійної роботи студентів

Ознайомитися з матеріалами лекцій. Розгляньте різні області застосування сховищ даних у Windows Azure. Рекомендується практичне завдання виконувати з використанням комп'ютерної техніки.

### 5.3 Контрольні запитання

1. Що являють собою підписки і яким чином вони пов'язані з безкоштовним обліковим записом Azure?
2. У чому полягає політика життєвого циклу для хмарних служб Azure?
3. Яким чином здійснюється підключення до сховища даних Azure?
4. Створити проект з використанням поставників Azure ASP.NET із застосуваннями MVC
5. Створити проект з використанням поставників Azure ASP.NET із застосуваннями з веб-формами.

### 5.4 Приклади аудиторних і домашніх завдань

#### Використання постачальників Azure ASP.NET з додатками MVC

У цій вправі розглядається зміна MVC ASP.NET додатку для використання постачальників ASP.NET із прикладів Windows Azure.

Додамо на сайт систему аутентифікації за допомогою постачальника членства. Потім застосовується постачальник ролей для класифікації користувачів і налаштовуються продукти, які буде пропонувати додаток. Нарешті, відбудеться налаштування постачальника стану сеансу для зберігання вмісту кошика.

#### 5.4.1 Налаштування доступу до додатка з перевіркою достовірності

Необхідно налаштувати додаток таким чином, щоб він вимагав доступ з перевіркою достовірності до всіх сторінок з кошиком покупок.

Якщо необхідно, запустіть Visual Studio в режимі підвищених адміністративних привілеїв, клацнувши правою кнопкою миші по значку Microsoft Visual Studio 2010 і вибравши пункт Run as administrator.

Продовжити роботу з рішенням, отриманим в ході виконання лабораторної роботи №5.

Відкрийте файл HomeController.cs в папці Controllers і відзначте клас атрибутом Authorize (Авторизувати). Ця установка змусить додаток вимагати доступ з перевіркою достовірності для всіх можливих дій на даному контролері.

С #

```

[HandleError]
[Authorize]
public class HomeController: Controller
{
    ...
}

```

Збережіть файл HomeController.cs.

#### 5.4.2 Налаштування підтримки членства з допомогою Azure TableStorageMembershipProvider.

Потрібно додати і налаштувати постачальників Azure ASP.NET для підтримки стану сеансу, управління ролями і членством.

Додайте до рішення проект постачальників Windows Azure ASP.NET. У браузері рішень Solution Explorer клацніть правою кнопкою миші рішення Begin, виберіть опцію Add, а потім команду Existing Project. Оберіть файл проекту та відкрийте його.

У проекті веб-ролі додайте посилання на AspProviders. У браузері рішень Solution Explorer клацніть правою кнопкою миші по вузлу проекту CloudShop і виберіть Add Reference. У діалоговому вікні Add Reference перейдіть на вкладку Projects, виберіть проект AspProviders і натисніть OK.

Оновіть конфігурацію служби таким чином, щоб вона включала рядок з'єднання з Обліковим записом сховища, де будуть міститися дані. Розгорніть вузол Roles в проекті CloudShopService і двічі клацніть вузол CloudShop, щоб відкрити вікно властивостей цієї ролі.

У вікні властивостей CloudShop [Role] (AzureStore [Роль]) виберіть вкладку Settings і натисніть Add Setting. Встановіть ім'я нового параметра на DataConnectionString і змініть тип Type на Connection String. Потім в стовпці значень клацніть кнопку з «...».

У діалоговому вікні Storage Account Connection String виберіть пункт Use the storage emulator і натисніть OK.

Натисніть CTRL + S, щоб зберегти зміни в конфігурації ролей.

Відкрийте файл Web.config, розташований в кореневій папці проекту CloudShop.

#### 5.4.3 Налаштуйте зберігання інформації облікового запису, який вимагають постачальники ASP.NET в файлі конфігурації програми.

Для цього знайдіть елемент `<appSettings>`, який повинен бути порожнім, і замініть його наступним блоком конфігурації. При відсутності `appSettings` вставте блок в якості безпосереднього дочірнього елемента `<configuration>`.

Крім файлу конфігурації служби для настройки постачальників Azure можна використовувати файл `Web.config` самого додатка. Це дозволяє розміщувати додаток поза структурою Azure, але при цьому, як і раніше використовувати постачальників і сховище Azure ASP.NET. Однак при запуску програми в середовищі Windows Azure параметри конфігурації, встановлені у файлі конфігурації служби для постачальників ASP.NET, мають перевагу перед параметрами, встановленими в файлі `Web.config`. Використання параметрів Windows Azure дозволяє уникнути необхідності повторного розгортання програми при зміні параметрів постачальника.

XML

...

```
<add key="DataConnectionString" value="UseDevelopmentStorage=true"/>
```

У проєкті `AspProviders` налаштуйте додаток на використання постачальника членства. Для цього замініть існуючий розділ `<membership>` в елементі `<system.web>` на наступну конфігурацію.

XML

...

```
<membership defaultProvider="TableStorageMembershipProvider" userIsOnlineTimeWindow="20" hashAlgorithmType="HMACSHA256">
```

```
<providers>
```

```
<clear/>
```

```
<add name="TableStorageMembershipProvider"
```

```
type="Microsoft.Samples.ServiceHosting.AspProviders.TableStorageMembershipProvider"
```

```
description="Membership provider using table storage"
```

```
applicationName="AzureStore"
```

```
enablePasswordRetrieval="false"
```

```
enablePasswordReset="true"
```

```
requiresQuestionAndAnswer="false"
```

```
minRequiredPasswordLength="1"
```

```
minRequiredNonalphanumericCharacters="0"
```

```
requiresUniqueEmail="true"
```

```
passwordFormat="Hashed"/>
```

```
providers>
```

```
membership>
```

...

Зберемо та запустимо додаток (F5). При першому запуску програми може знадобитися процедура ініціалізації, яка використовує емулятор зберігання.

При виконанні застосування перенаправляється на сторінку входу, оскільки згідно з новими параметрами авторизації доступ до домашнього контролера Home controller вимагає перевірки автентичності. Для продовження необхідно створити обліковий запис, оскільки база даних членства спочатку порожня. Щоб перейти до форми реєстрації нового користувача, клацніть Register на сторінці входу.

Щоб створити обліковий запис, заповніть реєстраційну форму та натисніть Register. Вхід в систему буде виконаний автоматично після створення облікового запису, на екрані відобразиться сторінка продуктів. Зверніть увагу: ім'я користувача буде відображатися у верхньому правому куті вікна.

Щоб зупинити запущену програму, закрийте вікно оглядача.

Електронне навчальне видання

Методичні вказівки  
до практичних занять

з дисципліни  
«\*Робота з даними на платформі .NET»

для студентів усіх форми навчання  
напряму 6.050103 «Програмна інженерія»  
спеціальності 121 – «Інженерія програмного забезпечення»

Упорядник ШИРОКОПЕТЛЄВА Марія Сергіївна

Відповідальний випусковий З.В.Дудар

Авторська редакція

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни  
«\*Робота з даними на платформі .NET»

для студентів усіх форм навчання  
напряму підготовки 6.050103 – Програмна інженерія.  
спеціальності 121 – «Інженерія програмного забезпечення»

Електронне видання

Затверджено  
кафедрою ПП.  
Протокол № 4 від 17.10.2017 р.

Харків 2017

Методичні вказівки до лабораторних робіт з дисципліни «\*Робота з даними на платформі .NET» для студентів усіх форм навчання напряму підготовки 06.050103 – Програмна інженерія, спеціальності 121 – «Інженерія програмного забезпечення» [Електронне видання] / Упоряд.: М.С.Широкопетлева. – Харків: ХНУРЕ, 2017. – 40 с.

Упорядники: М.С.Широкопетлева.

Рецензент:



## ЗМІСТ

Вступ.....	4
1 Розробка бібліотеки класів із суворим ім'ям і її розміщення в GAC .....	5
1.1 Мета роботи.....	5
1.2 Методичні вказівки з організації самостійної роботи студентів.....	5
1.3 Порядок виконання роботи і методичні вказівки з її виконання .....	8
1.4 Зміст звіту .....	10
1.5 Контрольні запитання і завдання .....	10
2 Використання ADO.Net для читання та модифікації даних .....	11
2.1 Мета роботи .....	11
2.2 Методичні вказівки з організації самостійної роботи студентів .....	11
2.3 Порядок виконання роботи та методичні вказівки з її виконання .....	12
2.4 Зміст звіту .....	19
2.5 Контрольні запитання та завдання .....	20
3 Створення та використання ADO.NET Entity Data Model .....	20
3.1 Мета роботи.....	20
3.2 Методичні вказівки з організації самостійної роботи студентів.....	20
3.3 Порядок виконання роботи та методичні вказівки з її виконання .....	22
3.4 Зміст звіту .....	28
3.5 Контрольні запитання та завдання .....	28
4 Конфігурація NHibernate для доступу до локальної БД.....	29
4.1 Мета роботи.....	29
4.2 Методичні вказівки з організації самостійної роботи студентів.....	29
4.3 Порядок виконання роботи та методичні вказівки з її виконання .....	29
4.4 Зміст звіту .....	35
4.5 Контрольні запитання та завдання .....	35
5 Міграція застосувань ASP.NET на Windows Azure. ....	36
5.1 Мета роботи.....	36
5.2 Методичні вказівки з організації самостійної роботи студентів.....	36
5.3 Порядок виконання роботи та методичні вказівки з її виконання .....	36
5.4 Зміст звіту .....	38
5.5 Контрольні запитання та завдання .....	38
Перелік посилань.....	39

## ВСТУП

У сучасних умовах однією з найпоширеніших задач для прикладного програміста є розробка та супроводження програмних застосунків, створених на платформі .Net. Дисципліна "\*Робота з даними на платформі .NET" має за мету опанування студентами компонентної моделі програмування на основі платформи Microsoft .NET Framework та заснованих на цій моделі методів конструювання програм.

Дані методичні вказівки орієнтовані на опанування студентами-програмістами тонкощів роботи з MS Visual Studio 2012/2013/2015. Лабораторні роботи розраховані на те, що студенти добре володіють мовою C# та мають досвід використання MS Visual Studio 2012 / 2013/2015.

Для виконання лабораторних робіт необхідні такі програмні засоби: ОС Windows 7/8/10, платформи .Net Framework 4.0/4.5 та вище, середовища візуального проектування MS Visual Studio 2012/2013/2015/2017. Методичні вказівки розроблено з використанням ОС Windows 7, платформи .Net Framework 4.5, середовища візуального проектування MS Visual Studio 2012/2013/2015. У апробації лабораторних робіт приймали участь студенти потоку ПІ-13.

Всі терміни, що використовується у методичних вказівках, наводяться у контексті програмної платформи .Net.

# 1 РОЗРОБКА БІБЛІОТЕКИ КЛАСІВ ІЗ СУВОРИМ ІМ'ЯМ І ІІ РОЗМІЩЕННЯ В GAC

## 1.1 Мета роботи

Ознайомитися з поняттями збірки , збірки з суворим ім'ям, набути практичні навички створення зборок і їх розміщення в Global Assembly Cache (GAC, глобальний кеш збірок).

## 1.2 Методичні вказівки з організації самостійної роботи студентів

Необхідно ознайомитися з матеріалами лекцій і 1, с. 94-121; 2, с. 54-60, 74-75, 494-537.

1.2.1 Файл Program.exe - це збірка (assembly), тобто набір з одного або декількох файлів з визначеннями типів і файлами ресурсів, а не просто PE-файл з метаданими. Один з файлів збірки вибирають для зберігання її декларації.

Збірка - це одиниця повторного використання, управління версіями і безпеки типів, яка дозволяє розподіляти типи та ресурси окремих файлів , щоб її користувачі могли вирішити , які файли упаковувати і розгортати разом.

Декларація (manifest) - це набір таблиць метаданих, які в основному містять імена файлів, складових збірки. Завантаживши файл з декларацією, Common Language Runtime (CLR) може визначити, які файли збірки містять типи і ресурси, на які посилається додаток. Будь-якому споживачеві збірки треба знати лише ім'я файлу, що містить декларацію, після чого він зможе, не порушуючи роботи програми, абстрагуватися від особливостей розподілу вмісту збірки по файлах, яке з часом може змінюватися.

Характеристики збірок:

- в збірці визначені повторно використовувані типи;
- збірка позначена номером версії;
- зі збіркою може бути пов'язана інформація безпеки.

При компіляції збірки з використанням командного рядку можна використовувати різні параметри. Наприклад, параметри, які використовуються при включенні в збірку файлів ресурсів:

- параметр /embed [ resource ] дозволяє додати в збірку файли ресурсів (файли у форматі , відмінному від PE) використанні утиліти Assembly Linker).

Параметр приймає будь-який файл і включає його вміст в результуючий PE-файл;

- параметр `/link [ resource ]` приймає файл з ресурсами ( оновлює таблиці декларації `ManifestResourceDef` і `FileDef` відомостями про ресурс і про те, в якому файлі збірки він знаходиться). Сам файл з ресурсами зберігається окремо;

- параметр `/win32res` дозволяє включити стандартні ресурси Win32;

- параметр `/win32icon` дозволяє включити в збірку стандартний ресурс значка Win32.

Кожна збірка має бути пронумерована, номер версії включають для повноти характеристики збірки та відстежування її модифікацій. Існують такі номери версій:

- `AssemblyFileVersion` - цей номер версії зберігається в ресурсі версії Win32 і призначений лише для інформації, CLR його повністю ігнорує (автоматично не оновлюється);

- `AssemblyInformationalVersion` - цей номер версії також зберігається в ресурсі версії Win32 і призначений лише для інформації, служить для вказівки версії продукту, в який входить збірка.

- `AssemblyVersion` - цей номер версії зберігається в декларації, в таблиці метаданих `AssemblyDef`. CLR використовує цей номер версії для прив'язки до збірок, які мають строгі імена.

При закритому розгортанні збірок Visual Studio упакує всі збірки, необхідні програмному застосуванню в один файл `.appx`, який або відправляється в Windows Store, або завантажується.

Збірки з закритим розгортанням (*privately deployed assemblies*) - збірки, що розгортаються в тому ж каталозі, що і додаток. Ці збірки не використовуються спільно іншими додатками (тільки якщо інші програми не розгортають в тому ж каталозі).

Метадані вказують, яку збірку, на яку вони посилаються, потрібно завантажити. При цьому не використовуються параметри реєстру.

### 1.2.2 Збірки з суворим ім'ям

Наведемо атрибути, що унікально ідентифікують збірку:

- ім'я файлу (без розширення);
- номер версії (*version*);
- ідентифікатор регіонального стандарту (*culture*);

– відкритий ключ (оскільки відкриті ключі являють собою дуже великі числа, замість останнього атрибуту використовується невеликий хеш відкритого ключа - маркер відкритого ключа (public key token)).

Наведемо приклади різних файлів збірки:

"MyExamples, Version = 1.0.8123.0, Culture = neutral, PublicKeyToken = b32a1c567890e012".

" MyExamples, Version = 1.0.8123.0, Culture ="en US", PublicKeyToken = b32a1c567890e012".

"MyExamples, Version = 2.0.1234.0, Culture = neutra , PublicKeyToken = b32a1c567890e012"

"MyExamples, Version = 1.0.8123.0, Culture = neutral, PublicKeyToken = b03f5f7f99d50a3a"

Використання стандартних криптографічних технологій, заснованих на парі із закритого і відкритого ключів дозволяє створити збірку з суворим ім'ям (див. [55, с. 5-9]).

Перший етап створення збірки - отримання ключа за допомогою утиліти Strong Name , SN.exe (у складі . NET Framework SDK і Microsoft Visual Studio). Параметри командного рядка SN.exe чутливі до регістру.

Компіляція збірки з використанням закритого ключа виконується командою:

```
csc / keyfile : <ім'я файлу >
```

Наприклад:

```
csc / keyfile : MyAssembly.keys app.cs
```

Виявивши в початковому тексті цей параметр, компілятор відкриває заданий файл (MyAssembly.keys), підписує збірку закритим ключем і вбудовує відкритий ключ в декларацію збірки. Підписується лише файл збірки, що містить декларацію.

### 1.2.3 Глобальний кеш збірок (GAC)

Глобальний кеш збірок - місце, де розташовуються спільно використовувані збірки:

C:\Windows\Assembly

При використанні GAC існують стандартні проблеми:

- колізія імен (можливий варіант що збірка використовує типи з тими ж іменами , що використовуються в розділяється збірці);
- питання сумісності.

У період розробки і тестування зборок зі строгими іменами для встановлення їх у GAC найчастіше застосовують інструмент GACUtil.exe з використанням таких параметрів:

- /i - установка збірки в GAC;
- /u - видалення збірки з GAC;
- /l – перегляд усіх збірок, розміщених у GAC (або отримання інформації щодо конкретної збірки, для чого вказується ім'я збірки без розширення).

#### 1.2.5 Пошук потрібної збірки

Якщо задано ім'я файлу без вказівки шляху, csc.exe намагається знайти потрібну збірку в каталогах, порядок перегляду яких наступний:

- робочий каталог;
- каталог, де знаходиться CLR. Цей каталог також містить dll-бібліотеки CLR;
- каталоги, задані параметром командного рядка /lib при виклику csc.exe;
- каталоги, зазначені в змінній оточення LIB.

### 1.3 Порядок виконання роботи і методичні вказівки з її виконання

#### 1.3.1 Створимо код для збірки

Створимо програмний додаток «Калькулятор», який дозволяє виконувати додавання та віднімання цілих чисел. Додаток містить клас Calculator.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace APP_LAB1
{
    public static class Calculator
    {
        public static int Add(int a, int b)
        {
            return a + b;
        }
        public static int Subtract(int a, int b)
        {
            return a - b;
        }
    }
}
```

```
}
```

Клас Program містить виклик створених методів та друк результатів.

```
using System;
namespace APP_LAB1
{
    public class Program
    {
        static void Main(string[] args)
        {
            int a = 10;
            int b = 5;
            Console.WriteLine(".NET Laba 1");
            /*Виведення інформацію про збірку: ім'я, версію*/
            Console.WriteLine("version: {0}",
                System.Reflection.Assembly.GetExecutingAssembly().GetName().Version);
            Console.WriteLine("{0} + {1} = {2}", a, b,
                Calculator.Add(a, b));
            Console.WriteLine("{0} - {1} = {2}", a, b,
                Calculator.Subtract(a, b));
            Console.ReadLine();
        }
    }
}
```

1.3.2 Створимо пару ключів з використанням утиліти sn.exe. Команда генерації пари ключів:

```
SN- k MyKey.keys
```

Далі можна відкомпілювати збірку з вказівкою сгенерованої пари ключів.

При необхідності використання лише відкритого ключа, можна створити файл, що містить тільки відкритий ключ:

```
SN- p MyKey.keys MyKey.PublicKey
```

Переглянемо створений відкритий ключ:

```
SN- tp MyKey.PublicKey
```

Закритий ключ не відображається.

1.3.3 Підпишемо збірку і розмістимо її в GAC, для чого використаємо утиліту GACUtil.exe. Знайдемо розміщення утиліти gacutil.exe на локальній машині. Зазвичай, вона знаходиться в підкаталогу нижчого рівня каталогу C:\Program Files\Microsoft SDKs\Window\версія Window... . Виконаємо команду:

Gacutil.exe /i APP\_LAB1.exe

Після додавання в каталозі C:\Windows\Microsoft.NET\assembly\GAC\_MSIL буде знаходитися додана збірка - для неї буде створений окремий каталог, як і для інших збірок, який носитиме коротке ім'я збірки.

Далі можна використовувати збірку з GAC. Для цього створимо який-небудь проект та у вікні Solution Explorer натиснимо на вузол References (Посилання). У меню, що з'явилося, треба обрати Add Reference. У вікні додавання посилання на збірку натиснимо на кнопку Browse та знайдемо в GAC збірку. Після цього вона буде додана в проект, в якому можна використовувати увесь закладений функціонал.

#### 1.4 Зміст звіту

Звіт оформлюється у текстовому редакторі. Він має містити: титульний аркуш, назву лабораторної роботи, мету, постановку задачі, скрипти створення збірок, згідно останній цифрі номера в журналі (див. завдання 9, п.1.5) та результати виконання створених програм; висновки по роботі.

Звіт має бути збережений у файлі з іменем виду «Прізвище\_PNet\_1», де «Прізвище» співпадає з прізвищем студента, який виконав роботу. Файл звіту разом з текстами створених програм має бути запакований до ZIP-архіву, що само розпаковується з ім'ям виду «Прізвище\_PNet\_1» та зданий викладачу або розміщений у відповідному розділі сайту організації індивідуальної роботи студентів.

#### 1.5 Контрольні запитання і завдання

- 1 . Що таке метадані?
2. Яким чином можна включити в збірку файл ресурсів?
3. Охарактеризуйте многофайлові збірки.
4. Який з номерів версії зберігається в маніфесті ?
5. Що таке регіональний стандарт ?
6. Яким чином можна згенерувати пару ключів?
7. Що потрібно для розміщення збірки в глобальний кеш збірок?
8. Який з каталогів перевіряється компілятором першим при пошуку потрібної збірки ?
9. Створити збірку згідно власного варіанту:
  - 0) Знаходження сум та доданків діагоналей квадратної матриці.



- 1) Знаходження найбільшого та найменшого елементів матриці та їх індексів. Упорядкування матриці.
- 2) Визначення балансу дужок у файлі.
- 3) Реалізація структури зберігання інформації щодо музичних творів та їх авторів. Визначення авторів, які мають найбільшу та найменшу кількість творів.
- 4) Реалізація алгоритму знаходження найкоротшого шляху між двома точками.
- 5) Реалізація алгоритму визначення дальності польоту тіла, кинутого під кутом до горизонту.
- 6) Реалізація структури зберігання інформації щодо школярів та їх семестрових оцінок. Визначення найкращих учнів у кожному класі.
- 7) Реалізація структури зберігання інформації щодо школярів та їх підручниках. Визначення учнів, які вчасно не повернули підручники.
- 8) Реалізація структури зберігання інформації щодо клієнтів спортивного клубу та їх відвідуваннях. Визначення найактивніших клієнтів.
- 9) Реалізація структури зберігання інформації щодо клієнтів спортивного клубу та їх тренерів. Визначення найпопулярніших тренерів (тренерів, які мають найбільшу кількість клієнтів).
10. Підписати створену збірку, розмістити її в GAC .
11. Перевірити наявність створеної збірки в GAC, змінити версію збірки, повторно створити збірку та розмістити її в GAC. Перевірити наявність двох збірок з однаковими іменами у GAC та підключити одну зі створених збірок до іншого проекту.
12. Яким чином можна розмістити збірку з відкладеним підписом у GAC?

## 2 ВИКОРИСТАННЯ ADO.NET ДЛЯ ЧИТАННЯ ТА МОДИФІКАЦІЇ ДАНИХ

### 2.1 Мета роботи

Набуття практичних навичок і закріплення теоретичних відомостей з використання технології ADO.Net.

### 2.2 Методичні вказівки з організації самостійної роботи студентів

Ознайомитися з матеріалами лекцій. Звернути увагу на особливості доступу до даних на підставі технології ADO.NET

ADO.NET надає собою технологію роботи з даними, яка заснована на платформі .NETFramework. Ця технологія являє нам набір класів, через які ми можемо відправляти запити до баз даних, встановлювати підключення, отримувати відповідь від бази даних і виробляти ряд інших операцій.

Основу інтерфейсу взаємодії з базами даних в ADO.NET представляє обмежене коло об'єктів: Connection, Command, DataReader, DataSet і DataAdapter. За допомогою об'єкта Connection відбувається установка підключення до джерела даних. Об'єкт Command дозволяє виконувати операції з даними з БД. Об'єкт DataReader зчитує отримані в результаті запиту дані. Об'єкт DataSet призначений для зберігання даних з БД і дозволяє працювати з ними незалежно від БД. І об'єкт DataAdapter є посередником між DataSet і джерелом даних. Головним чином, через ці об'єкти і буде йти робота з базою даних.

Однак щоб використовувати один і той же набір об'єктів для різних джерел даних, необхідний відповідний провайдер даних. Власне через провайдер даних в ADO.NET і здійснюється взаємодія з базою даних. Причому для кожного джерела даних в ADO.NET може бути свій провайдер, який власне і визначає конкретну реалізацію вищевказаних класів.

Схематично архітектуру ADO.NET можна представити як наведено на рис. 2.1.

В даному випадку ми будемо розглядати основні концепції ADO.NET переважно на прикладі MS SQL Servera. Тому спочатку нам треба встановити SQL Server Express. Всі необхідні матеріали для установки можна знайти за адресою <https://www.microsoft.com/en-US/download/details.aspx?id=42299>. Разом з сервером також встановлюється спеціальна програма SQL Server Management Studio, яка використовується для управління базами даних на сервері.



Рисунок 2.1 – Схематичне представлення технології ADO.NET

## 2.3 Порядок виконання роботи та методичні вказівки з її виконання

Для початку створимо просту базу даних на MS SQL Server. Для цього відкріємо SQL Server Management Studio та натиснемо на вузол Databases правою кнопкою миші. Після цього в контекстному меню виберемо New Database.

Після цього нам відкривається вікно для створення бази даних.

У ньому в поле Database Name нам треба ввести назву бази даних. Введемо animaldb. Більше тут вводити нічого не потрібно, і тому натиснемо на ОК.

Після цього в вузлі Databases з'являється новий елемент, який представляє тільки що створену базу даних animaldb. Розкриємо його і натиснемо правою кнопкою миші на його подвузел Tables.

Потім нам відкривається дизайнер таблиці.

У ньому треба вказати три стовпці: Id, Name і Age, які представлятимуть відповідно унікальний ідентифікатор тваринного, його ім'я і вік. У першого і третього стовпчика треба вказати тип int (тобто цілочисельний), а у стовпці Name - тип nvarchar (строковий).

Крім того, у вікні властивостей таблиці в поле Name треба ввести ім'я таблиці - Animals, а в поле Identity ввести Id, тобто тим самим вказуючи, що стовпець Id буде ідентифікатором.

І в кінці нам треба встановити курсор на стовпець Id і в панелі інструментів програми натиснути на золотий ключик. Після цього напроти поля Id також повинен з'явитися золотий ключик. Цей ключик буде вказувати, що стовпець Id виконуватиме роль первинного ключа.

Після цього натиснемо на збереження і потім на клавішу F5 (оновлення), і в вузлі нашої бази даних з'явиться нова таблиця, яка буде називатися dbo.Animals.

Після визначення джерела даних ми можемо до нього підключатися. Для цього створимо проект простого консольного застосування.

Насамперед треба визначити рядок підключення, що надає інформацію про базу даних та сервері, до яких належить встановити підключення:

```
class Program
{
    static void Main(string[] args)
    {
        string connectionString = @"Data
Source=.\SQLEXPRESS;Initial Catalog=animalsdb;Integrated Security=True";
    }
}
```

Data Source: вказує на назву сервера. За замовчуванням це ". \ SQLEXPRESS". Оскільки в рядку використовується слеш, то на початку рядка ставиться символ @. Якщо ім'я сервера бази даних відрізняється, то відповідно його і треба використовувати.

Initial Catalog: вказує на назву бази даних на сервері.

Integrated Security: встановлює перевірку справжності.

Жорстке кодування рядка підключення (тобто її визначення в коді програми), як правило, рідко використовується. Набагато більш гнучкий шлях представляє визначення її в спеціальних конфігураційних файлах програми. У проектах десктопних додатків це файл App.config, а в веб-додатках це в основному файл Web.config. Хоча додаток також може використовувати інші способи визначення конфігурації.

Додамо в appConfig.xml

```

<?xml version="1.0" encoding="utf-8" ?>
    <configuration>
        <startup>

            <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
        </startup>
        <connectionStrings>

            <add name="DefaultConnection" connectionString="Data
Source=.\SQLEXPRESS;Initial Catalog=animalsdb;Integrated Security=True"

            providerName="System.Data.SqlClient" />
        </connectionStrings>
    </configuration>

```

Отримаємо рядок підключення в застосуванні:

```

using System;
using System.Configuration;

namespace AdoNetConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            //string connectionString = @"Data
Source=.\SQLEXPRESS;Initial Catalog=animalsdb;Integrated Security=True";
            // получаем строку подключения
            string connectionString =
ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionStr
ing;

            Console.WriteLine(connectionString);

            Console.Read();
        }
    }
}

```

Для підключення до бази даних, необхідно створити та використовувати об'єкт SqlConnection:

```

using System;
using System.Data.SqlClient;

```

```

namespace AdoNetConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = @"Data
Source=.\SQLEXPRESS;Initial Catalog=animalsdb; Integrated Security=True";

            // Створення підключення
            SqlConnection connection = new SqlConnection(connectionString);
            try
            {
                // Відкриття підключення
                connection.Open();
                Console.WriteLine("підключення відкрито");
            }
            catch (SqlException ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                // закриття підключення
                connection.Close();
                Console.WriteLine("підключення закрито...");
            }

            Console.Read();
        }
    }
}

```

В якості альтернативи можна використовувати:

```

static void Main(string[] args)
{
    string connectionString = @"Data Source=.\SQLEXPRESS;Initial
Catalog=animalsdb;Integrated Security=True";
    using (SqlConnection connection =
new SqlConnection(connectionString))
    {
        connection.Open();
        Console.WriteLine("Подключениеоткрыто");
    }
}

```

```

    }
    Console.WriteLine("Подключениезакрито...");
    Console.Read();
}

```

Для виконання команди потребується sql-вираз та об'єкт підключення:

```

stringconnectionString = @"Data Source=.\SQLEXPRESS;Initial
Catalog=animalsdb;Integrated Security=True";
using (SqlConnection connection = newSqlConnection(connectionString))
{
    connection.Open();
    SqlCommandcommand = newSqlCommand();
    command.CommandText = "SELECT * FROM Animals";
    command.Connection = connection;
}

```

За допомогою властивості CommandText встановлюється SQL-вираз, який буде виконуватися. В даному випадку це запит на отримання всіх об'єктів з таблиці Animals. А за допомогою властивості Connection можна встановити об'єкт підключення SqlConnection.

Виконаємо команду по додаванню одного об'єкта в таблицю Animals бази даних animalsdb, яка раніше була створена:

```

classProgram
{
    staticvoid Main(string[] args)
    {
        stringconnectionString = @"Data Source=.\SQLEXPRESS;Initial
Catalog=animalsdb;Integrated Security=True";
        stringsqlExpression = "INSERT INTO Animals (Name, Age) VALUES
('Tom', 18)";

        using (SqlConnectionconnection =
newSqlConnection(connectionString))
        {
            connection.Open();
            SqlCommandcommand = newSqlCommand(sqlExpression,
connection);
            int number = command.ExecuteNonQuery();
            Console.WriteLine("Добавленообъектов: {0}", number);
        }
        Console.Read();
    }
}

```

Збережені процедури є ще однією формою виконання запитів до бази даних. Але в порівнянні з раніше розглянутими запитам, які надсилаються з програми бази даних,

збережені процедури визначаються на сервері і надають велику продуктивність і є більш безпечними.

Але щоб використовувати збережені процедури, нам треба їх спочатку створити. Для цього перейдемо в SQL Server Management Studio до нашої бази даних usersdb, розкриємо її вузол і далі виберемо Programmability-> Stored Procedures. Натиснемо на цей вузол правою кнопкою миші і в контекстному меню виберемо пункт Stored Procedure ...:

У центральній частині програми відкриває код процедури, який генерується за замовчуванням. Замінімо цей код наступним:

```
CREATE PROCEDURE [dbo].[sp_InsertAnimal]
    @name nvarchar(50),
    @age int
AS
    INSERT INTO Animals (Name, Age)
    VALUES (@name, @age)

    SELECT SCOPE_IDENTITY()
GO
```

І потім натиснемо на кнопку Execute. Після цього в базу даних додається збережена процедура.

Подібним чином додамо ще одну процедуру, яка буде повертати об'єкти:

```
CREATE PROCEDURE [dbo].[sp_GetAnimals]
AS
    SELECT* FROM Animals
GO
```

Тепер перейдемо до коду C # і визначимо наступну програму

```
classProgram
{
    staticstringconnectionString = @"Data
Source=.\SQLEXPRESS;Initial Catalog=animalsdb;Integrated
Security=True";
    staticvoidMain(string[] args)
    {
        Console.Write("Введіть ім'я домашнього любимця:");
        stringname = Console.ReadLine();

        Console.Write("Введіть вік домашнього любимця:");
        intage = Int32.Parse(Console.ReadLine());

        AddAnimal(name, age);
        Console.WriteLine();
        GetAnimals();

        Console.Read();
    }
}
```

```

    }
    // добавление животного
    private static void AddAnimal(string name, int age)
    {
        // название процедуры
        string sqlExpression = "sp_InsertAnimal";

        using (SqlConnection connection =
new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression,
connection);
            // указываем, что команда представляет хранимую
процедуру
            command.CommandType =
System.Data.CommandType.StoredProcedure;
            // параметр для ввода имени
            SqlParameter nameParam = new SqlParameter
            {
                ParameterName = "@name",
                Value = name
            };
            // добавляем параметр
            command.Parameters.Add(nameParam);
            // параметр для ввода возраста
            SqlParameter ageParam = new SqlParameter
            {
                ParameterName = "@age",
                Value = age
            };
            command.Parameters.Add(ageParam);

            var result = command.ExecuteScalar();
            // если нам не надо возвращать id
            //var result = command.ExecuteNonQuery();

            Console.WriteLine("Id добавленного объекта: {0}",
result);
        }
    }

    // вывод всех пользователей

```



```

private static void GetAnimals()
{
    // названіє процедури
    string sqlExpression = "sp_GetAnimals";

    using (SqlConnection connection =
new SqlConnection(connectionString))
    {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression,
connection);
        // указуємо, що команда представляє збережену
процедуру
        command.CommandType =
System.Data.CommandType.StoredProcedure;
        var reader = command.ExecuteReader();

        if (reader.HasRows)
        {
            Console.WriteLine("{0}\t{1}\t{2}",
reader.GetName(0), reader.GetName(1), reader.GetName(2));

            while (reader.Read())
            {
                int id = reader.GetInt32(0);
                string name = reader.GetString(1);
                int age = reader.GetInt32(2);
                Console.WriteLine("{0} \t{1} \t{2}", id, name,
age);
            }
        }
        reader.Close();
    }
}

```

## 2.4 Зміст звіту

Звіт має містити: мету роботи; порядок виконання роботи у вигляді екранних форм, тексти програм; формулювання та результати завдань для самостійного виконання.

## 2.5 Контрольні запитання та завдання

1. Для чого використовується технологія ADO.Net?
2. Охарактеризуйте достоїнства та недоліки технології ADO.Net.
3. Яким чином можна підключитися до БД за квикористанням технології ADO.Net?
4. Створіть веб-застосування на підставі технології ADO.Net, яке використовує створену у лабораторній роботі№1 базу даних.
5. Організуйте перегляд даних. Додайте фільтр, пошук інформації за обраними критеріями.
6. Виконайте модифікацію даних.

## 3 СТВОРЕННЯ ТА ВИКОРИСТАННЯ ADO.NET ENTITY DATA MODEL

### 3.1 Мета роботи

Набуття практичних навичок і закріплення теоретичних відомостей з використання технології ADO.Net.

### 3.2 Методичні вказівки до виконання роботи

Ознайомитися з матеріалами лекцій. Звернути увагу на особливості використання технології Entity Framework

Платформа Entity Framework являє собою набір технологій ADO.NET , що забезпечують розробку додатків , пов'язаних з обробкою даних. Архітекторам і розробникам додатків , орієнтованих на обробку даних , доводиться враховувати необхідність досягнення двох зовсім різних цілей. Вони повинні моделювати сутності, зв'язки і логіку розв'язуваних бізнес -задач , а також працювати з ядрами СУБД , використовуваними для збереження і отримання даних. Дані можуть розподілятися за кількома системами зберігання даних , в кожній з яких застосовуються свої протоколи , але навіть в додатках , що працюють з однією системою зберігання даних , необхідно підтримувати баланс між вимогами системи зберігання даних і вимогами написання ефективного і зручного для обслуговування коду програми .

У Entity Framework розробники отримують можливість працювати з даними, представленими у формі відносяться до конкретних доменам об'єктів і властивостей, таких як клієнти та їх адреси, не будучи вимушеними звертатися до базових таблиць і стовпців бази даних, де зберігаються ці дані. Entity Framework дає розробникам можливість працювати з даними на більш високому рівні абстракції, створювати і супроводжувати додатки, орієнтовані на роботу з даними, одночасно з цим скорочуючи обсяг коду, в порівнянні з традиційними додатками. Оскільки Entity Framework є компонентом .NET Framework, додатки Entity Framework можуть працювати на будь-якому комп'ютері, де встановлена платформа .NET Framework, починаючи з версії 3.5 з пакетом оновлень 1 (SP1).

При використанні об'єктно-орієнтованого програмування для взаємодії з системами зберігання даних виникають складнощі. Безумовно, організація класів часто нагадує організацію таблиць реляційної бази даних, але така відповідність неідеально. Кілька нормалізованих таблиць часто відповідають єдиному класу, а зв'язки між класами представлені не так, як зв'язку між таблицями. Наприклад, для представлення клієнтові замовлення на продаж в класі Order може використовуватися властивість, що містить посилання на екземпляр класу Customer, але рядок таблиці Order бази даних містить стовпець зовнішнього ключа (або набір стовпців) із значенням, яке відповідає первинному ключу в таблиці Customer. Клас Customer може включати властивість з ім'ям Orders, що містить колекцію екземплярів класу Order, але таблиця Customer бази даних не містить порівнянного шпальти. У цьому випадку Entity Framework надає розробникам гнучкість у поданні зв'язків або більш повні зв'язку моделі (такі, як в базі даних).

Засоби модель EDM (сутнісна модель даних) можуть сформувати клас, похідний від DbContext, який представляє контейнер сутностей в концептуальній моделі. Контекст об'єкта надає кошти для відстеження змін та управління ідентифікаторами, паралелізмом і зв'язками. Цей клас представляє також доступ до методу SaveChanges, який записує результати вставки, оновлення та видалення даних у джерело даних. Подібно запитам, ці зміни проводяться або командами, автоматично сформованими системою, або збереженими процедурами, зазначеними розробником.

### 3.3 Порядок виконання роботи та методичні вказівки з її виконання

#### Створення і використання ADO.NET Entity Data Model

Створимо звичайний Windows Form проект.

У вкладці Server Explorer нам потрібно встановити з базою даних (див. рис.3.1). Для цього потрібно натиснути на піктограму Connect to Database.

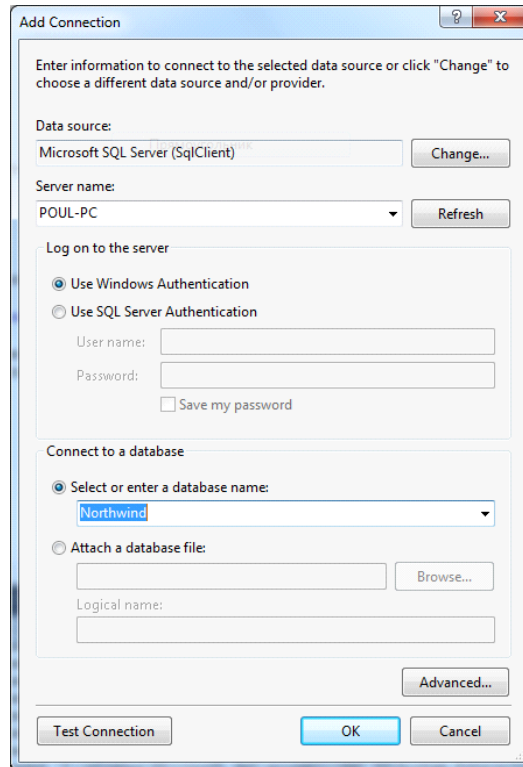


Рисунок 3.1 – Налаштування підключення до бази даних

Після чого відкриється вікно в якому вам потрібно вказати сервер де розташована потрібна БД, він може бути розташований на вашому комп'ютері, або віддалено в мережі. А також потрібно вибрати потрібну базу даних зі списку, які відображаються у списку бази даних - це ті бази які вже приєднані до сервера і функціонують на даний момент. Альтернативний варіант, можна вибрати базу даних з файлу, тоді вона автоматично приєднається до вибраного вами серверу, коли ви клікніть ОК.

Після чого в Server Explorer з'явитися підключення до обраної бази (рис.3.2).

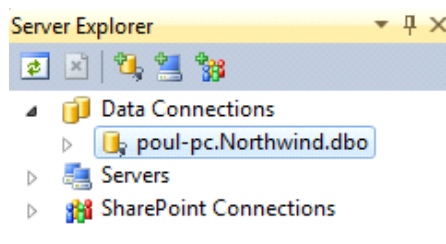


Рисунок 3.2 – Готове підключення

Після того, як встановлено підключення до бази даних можна переходити до створення Entity Model. На вкладці Solution Explorer клацаємо правою клавішею мишки по вашому проекту, і вибираємо в контекстному меню Add -> New item. У вікні, вибираємо в Installed Templates пункт Data і в центральній частині вікна вибираємо ADO.NET Entity Data Model, це буде файл з розширенням edmx (на зразок LINQ to SQL має розширення dbml), даємо моделі ім'я і натискаємо ADD.

Далі нам буде запропоновано згенерувати нашу модель по існуючій базі даних або створити порожню. Вибираємо Generate from database і натискаємо Next. В наступному, в випадаючому списку окне вибираємо підключення, яке ми створили, з'явиться текст, який буде вставлено в строку підключення і ім'я моделі.

Натискаємо Next і створюється готова модель по вибраній базі даних і підключаються всі необхідні References для роботи з даними і моделлю Entity. А також створиться файл edmx, який складається з двох частин - графічної (згенерований xml-файл), і файлу дизайнера, де знаходиться весь код нашої моделі (див. рис.3.3).

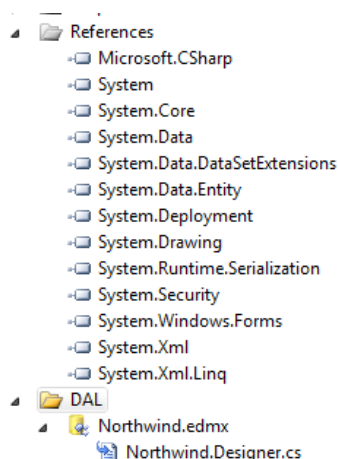


Рисунок 3.3 – Вкладка Solution Explorer – після додавання edmx-файла

Після того як ми зв'язали Entity Model з базою даних потрібно клікнути по edmx файлу і в головному вікні відобразитися графічне представлення нашої

моделі (див. рис.3.4): всі сутності будуть представлені прямокутниками, в яких відображатися всі властивості (Properties), цієї сутності (класу), і посилання (Navigation properties) на інші сутності, які зв'язуватися з допомогою зовнішніх ключів (foreign key) в базі даних. Всі сутності мають зв'язку між собою, на кінцях яких можна побачити тип зв'язку: один-до-одного, один-до-багатьох, багато-до-багатьох.

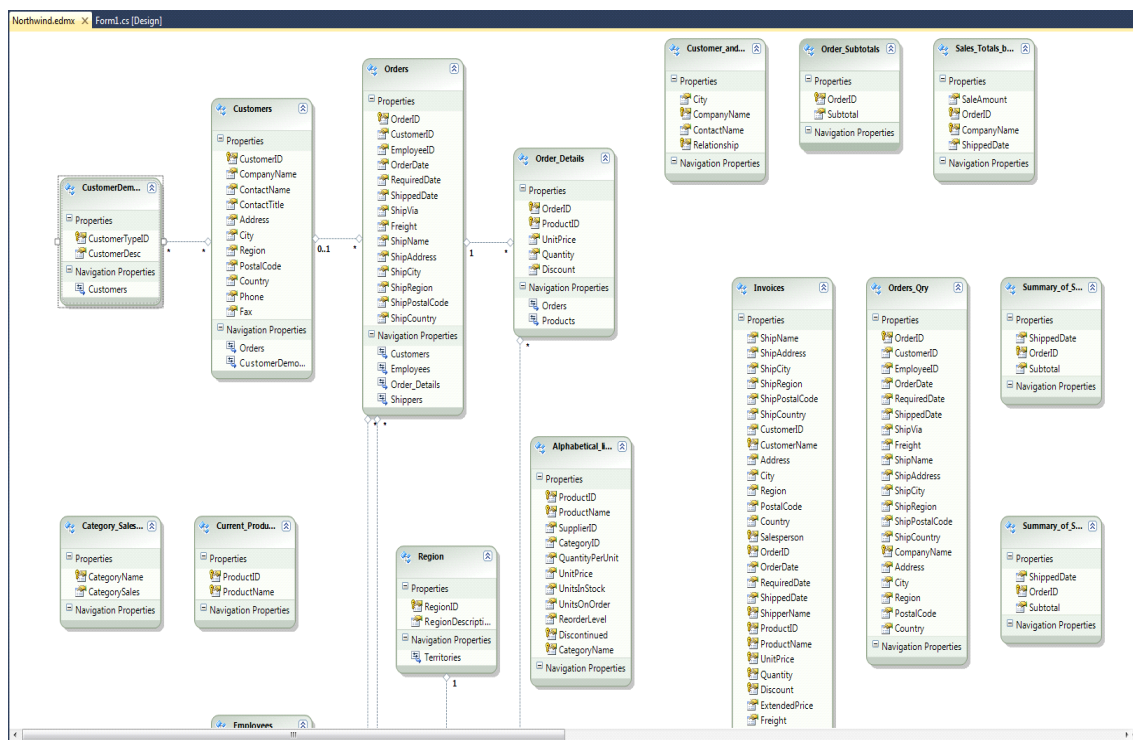


Рисунок 3.4 – Графічне відображення моделі

Прямо на представленні можна перейменовувати самі сутності, а також властивості (див. рис.3.5). Ще можна видаляти властивості за своїм розсудом, а також додавати нові властивості, посилання на інші сутності і складні властивості.

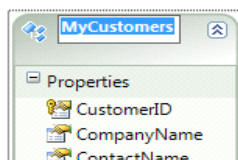


Рисунок 3.5 – Переіменування сутностей

Внизу з'явиться вкладка Mapping Details в режимі Map Entity to Table / Views (див.рис.3.6 та 3.7). При натисканні на будь-яку з сутностей ви там з'явитися відображення (мапінг) властивостей моделі і полів таблиці. Ми

можемо вручну виставляти відповідності, перейменовувати їх, присвоювати іншим властивостям чи декільком відразу. Так само можна накладати умови на деякі поля, де ми вказуємо, чому має рівняти наше поле. До однієї сутності можна прімаппіть відразу кілька таблиць і давати виставити свої властивості в моделі для іншої таблиці.

Region : nvarchar	↔	Region : String
PostalCode : nvarchar	↔	Country : String
Country : nvarchar	↔	CustomerID : String
Phone : nvarchar	↔	Fax : String
Fax : nvarchar	↔	Phone : String
		PostalCode : String
		Region : String

Add a Table or View>

Рисунок 3.6– Встановлення відповідностей між стовпчиками і властивостями

Mapping Details - Order_Details		
Column	Operator	Value / Property
<b>Tables</b>		
Maps to Order Details		
When OrderID	=	""
And ProductID	Is	Not Null
<Add a Condition>		
<b>Column Mappings</b>		
OrderID : int	↔	OrderID : Int32
ProductID : int	↔	ProductID : Int32
UnitPrice : money	↔	UnitPrice : Decimal
Quantity : smallint	↔	Quantity : Int16
Discount : real	↔	Discount : Single
<Add a Table or View>		

Рисунок 3.7 – Вкладка Mapping Details в режимі Map Entity to Table/Views

У лівому верхньому куті можна перемкнутися в режим Map Entity to Functions. В ньому можна відразу задати операціями вставки, оновлення та видалення відповідні збережені процедури які вже є в нашій моделі і були взяти з бази даних (див.рис.3.8).

<b>Functions</b>	
Insert Using CustOrdersDetail	
Parameters	
OrderID : int	OrderID : Int32
Result Column Bindings	
<Add Result Binding>	
<Select Update Function>	
<Select Delete Function>	

Рисунок 3.8– Вкладка Mapping Details в режимі Map Entity to Functions

Можна відразу налаштувати входять параметри і повертається результат для властивостей нашої сутності .

У правій частині екрана з'явиться нова вкладка Model Browser . У ній повністю відображень всі сутності і дані нашої моделі . Вона складається з двох частин - власне моделі і відображення нашої бази даних.

У моделі , в папці Entity Types знаходяться впорядковані сутності та їх властивості , які ми вже раніше бачили в графічному режимі. При правому кліку мишки , ми з контекстного меню можемо виробляти ті ж дії , що і в Mapping Details : видаляти , додавати , перейменовувати властивості , додавати збережені процедури і т. д. і т. п.

Так само вибравши в контекстному меню Update Model from Data на папці Entity Types можна оновити модель , якщо база даних була змінена. Всі автоматично переписує вашу модель відповідно з базою даних , - це важлива перевага над LINQ to SQL , де при зміні бази даних , доводиться заново пересоздавати весь маппінг і додавати модель , що дуже незручно в процесі розробки. За допомогою Generate Database from Model можна на основі моделі сформувати sql - скрипти для створення бази даних , які можна запустити в SQL проекті в Visual Studio або в MS SQL Management Studio . У папку Complex Types можна додавати складні типи , які складаються з декількох сутностей. А в папці Associations зберігаються всі зв'язки ( зовнішні ключі ) між сутностями.

У відображенні бази даних , іншої частини файлу edmx впорядковані всі елементи бази даних - таблиці та подання, їх поля, збережені процедури і зв'язки між таблицями (див.рис.3.9).

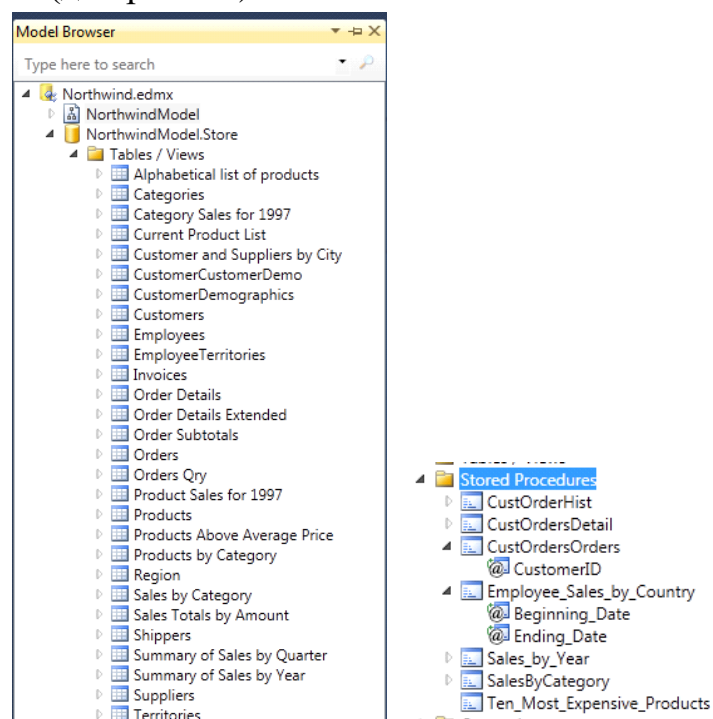


Рисунок 3.9 – Таблиці та збережені процедури в Model Browser



Після того створена модель можна програмно використовувати її у своєму проєкті. Візьмемо найпростіший випадок. Нехай необхідно відобразити всі дані з таблиці Customers . Для цього потрібно використовувати нашу модель цієї таблиці - MyCustomers .

Створіть просту форму і на неї перетягніть з вкладки Toolbox елемент DataGridView . Після чого необхідно клацнути два рази за формою і ми перейдемо до написання коду. При клацанні автоматично створиться метод `***_Load` , який виконатися при завантаженні нашої форми. У ньому необхідно відкрити контекст нашої моделі для того щоб виконати запит до бази і повернути потрібні нам дані

```
NorthwindEntities context = new NorthwindEntities ( )
```

Далі ми витягаємо LINQ -запитом з контексту потрібну сутність , в даний момент потрібно сутність Customers , і робимо на ній простий select для того щоб витягти всі дані з таблиці , і записуємо в колекцію реалізовує інтерфейс IQueryable . І отримуємо так званий «відкладений » запит. Ось так виглядає код:

```
IQueryable <MyCustomers> myCustomers = from c in context.Customers  
select c ;
```

Потім відкладений запит потрібно прив'язати до таблиці , яка розташована на формі , до елемента DataGridView . У цього елемента є властивість DataSource якому необхідно присвоїти значення колекції , попередньо перевіривши її в простий список :

```
dataGridView1.DataSource = myCustomers.ToList <MyCustomers> ();
```

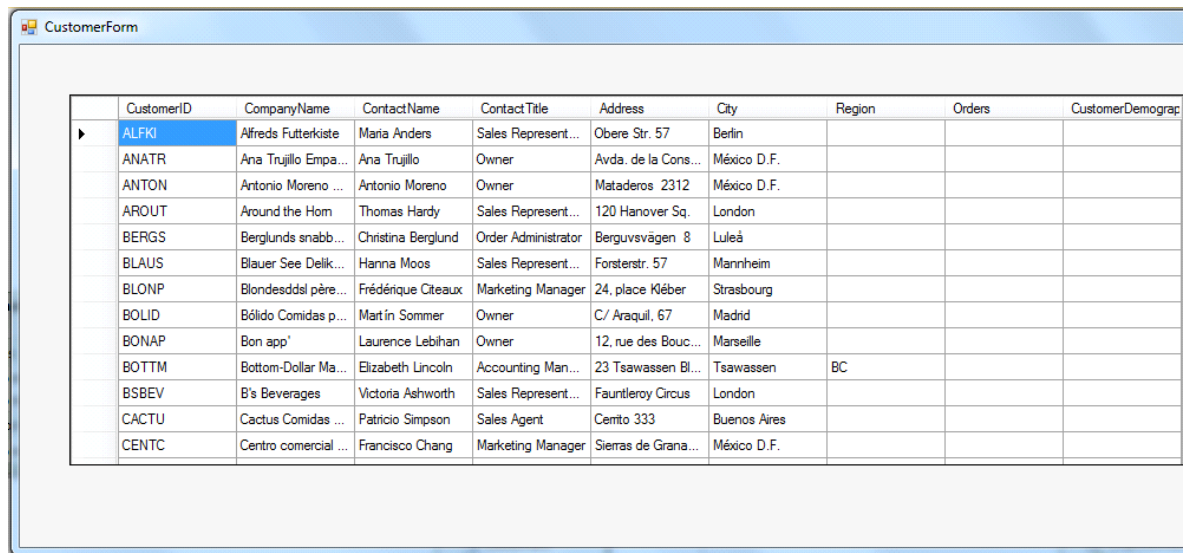
В цілому весь метод виглядає так:

```
private void CustomerForm_Load ( object sender , EventArgs e )  
{  
    using ( NorthwindEntities context = new NorthwindEntities ( ))  
    {  
        IQueryable <MyCustomers> myCustomers = from c in  
context.Customers select c ;  
        dataGridView1.DataSource = myCustomers.ToList <MyCustomers> ();  
    }  
}
```

Найважливіше пам'ятати те , що весь відкладений запит буде повертати значення , лише у відкритому контексті , тобто в межах директиви using . Якщо ми призначимо властивості DataSource наш відкладений запит поза директиви using, вміст ресурсу буде порожньо. Настійно не рекомендується

використовувати статичний контекст для виконання запитів. У реальних, високонавантажених, масштабних проектах він буде споживати велику кількість програмних і апаратних ресурсів. Відкривати контекст рекомендується тільки тоді, коли буде виконувати певну кількість запитів до бази з подальшим їх використанням.

Результат виконання функції описаної вище гаведено на рис.3.10.



	CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	Orders	CustomerDemograp
▶	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Represent...	Obere Str. 57	Berlin			
	ANATR	Ana Trujillo Empa...	Ana Trujillo	Owner	Avda. de la Cons...	México D.F.			
	ANTON	Antonio Moreno ...	Antonio Moreno	Owner	Mataderos 2312	México D.F.			
	AROUT	Around the Hom	Thomas Hardy	Sales Represent...	120 Hanover Sq.	London			
	BERGS	Berglunds snabb...	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå			
	BLAUS	Blauer See Delk...	Hanna Moos	Sales Represent...	Forsterstr. 57	Mannheim			
	BLONP	Blondesddsl père...	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg			
	BOLID	Bóldo Comidas p...	Martín Sommer	Owner	C/ Araquil, 67	Madrid			
	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouc...	Marseille			
	BOTTM	Bottom-Dollar Ma...	Elizabeth Lincoln	Accounting Man...	23 Tsawassen Bl...	Tsawassen	BC		
	BSBEV	B's Beverages	Victoria Ashworth	Sales Represent...	Fauntleroy Circus	London			
	CACTU	Cactus Comidas ...	Patricio Simpson	Sales Agent	Centto 333	Buenos Aires			
	CENTC	Centro comercial ...	Francisco Chang	Marketing Manager	Sierras de Grana...	México D.F.			

Рисунок 3.10– Відображення таблиці з даними на формі

### 3.4 Зміст звіту

Звіт має містити: мету роботи; порядок виконання роботи у вигляді екранних форм, тексти програм; формулювання та результати завдань для самостійного виконання.

### 3.5 Контрольні запитання та завдання

1. Які основні переваги Entity Framework перед іншими платформами (LINQ to SQL, ADO.NET)?
2. Розкажіть про архітектурі платформи Entity Framework?
3. Охарактеризуйте достоїнства та недоліки LINQ to Entity.
4. Яким класом контекст об'єкта представлений в Entity to SQL?
5. Які дії можна робити в Entity Model з сутностями моделі?
6. Що таке відкладене та негайне виконання запиту?
7. Що таке скомпільований запит?

8. Візьміть тестову або яку-небудь іншу готову базу даних і на її основі створюєте edmx -модель , з довільного числа таблиць , зв'язків і збережених процедур.

9. Виконайте будь-які дії з видалення , перейменування створенню нових сутностей і властивостей у них . Складіть асоціації на основні дії в таблицях (Update , Insert , Delete) за допомогою збережених процедур.

10. Виконайте відкладені запити до моделі , і виконайте деякі маніпуляції з отриманими даними. Візуально відобразіть отримані результати.

## 4 КОНФІГУРАЦІЯ NHIBERNATE ДЛЯ ДОСТУПУ ДО ЛОКАЛЬНОЇ БД

### 4.1 Мета роботи

Набуття практичних навичок і закріплення теоретичних відомостей з використання технології NHibernate.Net.

### 4.2 Методичні вказівки до виконання роботи

Ознайомитися з матеріалами лекцій. Звернути увагу на особливості реалізації доступу до даних на підставі технології NHibernate.Net. Окрему увагу слід приділити установці Nhibernate

Якщо ви завантажили архів з NHibernate , то все що вам потрібно це розпакувати його куди-небудь. Можна створити папку SharedLibs с : \ Code \ SharedLibs \ NHibernate і розпакувати архів в цю папку. Але можете робити звичайно як вам зручніше. Це ваша SharedLibs папка з якої ви згодом додасте посилання на збірки NHibernate і NUnit. Додайте посилання на NHibernate в обидва ваших демо проекту .

### 4.3 Порядок виконання роботи та методичні вказівки з її виконання

Необхідно створити застосування, що використовує NHibernate для доступу до даних. Перед тим як ми почнемо створювати наш додаток і бізнес об'єкти , ми повинні створити порожній проект. Запустіть Visual Studio і створіть новий Class Library проект. А тепер давайте подивимося на що то більш цікаве - це Бізнес Об'єкти .

Визначимо простий домен. На даний момент він включає тільки один клас Product. У Product є три властивості Name , Category і Discontinued (див.рис. 4.1).

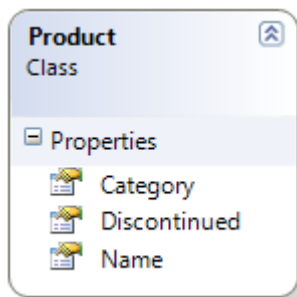


Рисунок 4.1 - Клас Product

Додайте папку Domain в проект FirstSample з вашого рішення. Додайте новий клас Product.cs в цю папку. Код дуже простий і використовує автоматичні властивості.

```
namespace FirstSolution.Domain
{
    public class Product
    {
        public string Name { get ; set ;}
        public string Category { get ; set ;}
        public bool Discontinued { get ; set ;}
    }
}
```

Зараз нам треба створити можливість, щоб наші створені класи зберігалися в БД. Для вирішення цього завдання ми і вибрали NHibernate . Рядок у створеному класі буде відповідати рядку в таблиці в БД. Для цього ми визначимо mapping між створеним нами класом і соответствующей таблицею в базі даних. Цей mapping може бути зроблений за допомогою mapping файлу ( xml - документ) або можна зробити за теж саме за допомогою атрибутів . Я почну з mapping файлу.

#### 4.3.2 Створення NHibernate mapping для завантаження і збереження бізнес об'єктів

Створіть папку Mappings в проекті FirstSample . Додайте новий xml - документ в цю папку і назвіть його Product.hbm.xml . Не забудьте " hbm " частина імені файлу. Ця угода імен використовується в NHibernate для автоматичного визначення файлу як mapping файлу. Задайте " Embedded Resource " як Build Action для xml файлу.

Знайдіть файл `nhibernate - mapping.xsd` в папці `NHibernate'a` і скопіюйте його в `SharedLibs` папку. Тепер ми можемо використовувати цю `xml` схему для більш зручного створення наших `mapping` файлів. `VS` буде нам надавати кошти `intellisense` і `validation` при редагуванні `xml mapping` документа . У `VS` додайте схему в `Product.hbm.xml` файл

І так давайте почнемо . Кожен `mapping` файл містить кореневої `node` `<hibernate-mapping>`

```
<? xml version = " 1.0" encoding = " utf -8" ? >
< hibernate - mapping xmlns = " urn : nhibernate - mapping -
2.2"
                                assembly = " FirstSolution "
                                namespace = " FirstSolution.Domain " >

    <! - More mapping info here ->

</ hibernate - mapping >
```

У `mapping` файлі коли ми посилаємося на клас -домен ви завжди повинні задати коректне повне ім'я класу (наприклад `FirstSample.Domain.Product`, `FirstSample`).

Зараз для початку нам треба задати `primary key` для класу `product` . Технічно ми могли б узяти властивість `Name` і могли б задати його як унікальне. Але в основному так не робиться і ми зробимо по іншому. Для цього ми додамо в наш клас нова властивість і назовемо його `Id`. Будемо використовувати `Guid` тип , але також можуть бути використані замість нього і `int` і `long` .

```
using System ;

namespace FirstSolution.Domain
{
    public class Product
    {
        public Guid Id { get ; set ; }
        public string Name { get ; set ; }
        public string Category { get ; set ; }
        public bool Discontinued { get ; set ; }
    }
}
```

## Завершуємо створення mapping файлу

```
<? xml version = " 1.0" encoding = " utf -8" ? >
< hibernate - mapping xmlns = " urn : nhibernate - mapping -
2.2"

        assembly = " FirstSolution "
        namespace = " FirstSolution.Domain " >

<class name="Product">
    <id name="Id">
        <generator class="guid" />
    </ id >
    <property name="Name" />
    <property name="Category" />
    <property name="Discontinued" />
</ class >

</ hibernate - mapping >
```

NHibernate не надходить як ми , він робить багато чого за замовчуванням. Так якщо ви не задали ім'я колонки в " property " , то ім'я буде взято з властивостей . Або NHibernate може автоматично взяти назву таблиці із заданого класу. Як наслідок - мій xml mapping файл не захищений зайвою інформацією . Будь ласка подивіться документацію для більш детального пояснення mapping . Ви можете знайти її тут.

Solution explorer повинен виглядати як продемонстровано на рис.4.3.

### 4.3.3 Конфігурація NHibernate для доступу до вашої локальної Базі Даних

Зараз необхідно вирішити яку БД будемо використовувати і треба визначити деталі підключення в connection string.

Додайте новий xml файл в FirstSolution проект and назвіть його hibernate.cfg.xml . Встановіть властивість " Copy to Output " в " Copy always " . Так як ми в нашому прикладі будемо використовувати SQL Server Compact Edition то вам необхідно додати наступну інформацію в xml файл.

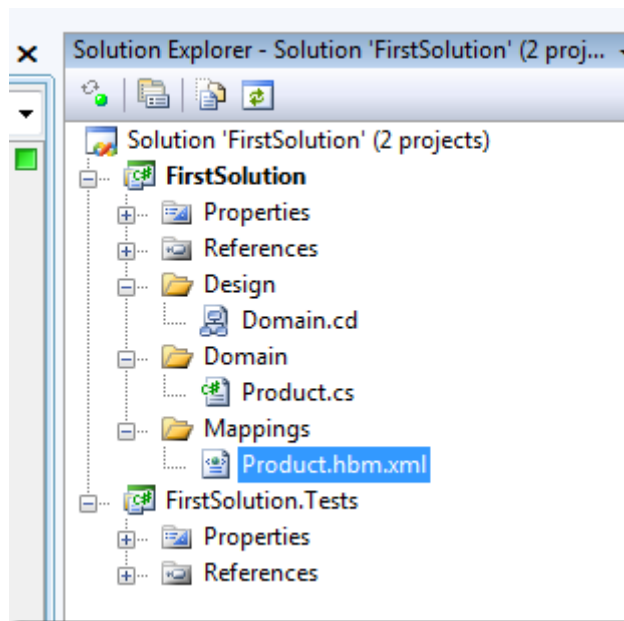


Рисунок 4.3- Вигляд Solution explorer

```
<? xml version = " 1.0" encoding = " utf -8" ? >
<hibernate-configuration xmlns="urn:nhibernate-configuration-
2.2">
    <session-factory>
        <property name="connection.provider">
NHibernate.Connection.DriverConnectionProvider </ property >
        <property name="dialect">
NHibernate.Dialect.MsSqlCeDialect </ property >
        <property name="connection.driver_class">
NHibernate.Driver.SqlServerCeDriver </ property >
        <property name="connection.connection_string"> Data
Source = FirstSample.sdf </ property >

        <property name="show_sql"> true </ property >
    </ session - factory >
</ hibernate - configuration >
```

За допомогою цього файлу конфігурації ми говоримо NHibernate'у що ми будемо використовувати MS SQL Server Compact Edition як нашу основну БД і ім'я БД буде FirstSample.sdf ( що еквівалентно connection string ) . Ми також визначаємо що нам треба бачити SQL який NHibernate генерує і посилає в БД ( це рекомендовано зробити , так це нам допоможе надалі в дебагом в процесі розробки).

Додайте порожню БД в ваш FirstSample проект і назвіть її FirstSample.sdf (виберіть Local Database в шаблонах )

Натисніть Додати і проігноруйте Майстер створення Dataset , просто натисніть Cancel.

#### 4.3.4 Запити до Бази Даних

Необхідно реалізувати три методи для запитів до об'єктів бази даних. Давайте почнемо з найпростішого GetById . Для початку напишемо тест

```
[ Test ]
public void Can_get_existing_product_by_id ( )
{
    IProductRepository repository = new
ProductRepository ( );
    var fromDb = repository.GetById ( _products [ 1 ]
. Id ) ;

    Assert.IsNotNull ( fromDb ) ;
    Assert.AreNotSame ( _products [ 1 ] , fromDb ) ;
    Assert.AreEqual ( _products [ 1 ] . Name ,
fromDb.Name ) ;
}
і тепер додамо код в репозитарії щоб закінчити з цим тестом
public Product GetById ( Guid productId )
{
    using ( ISession session =
NHibernateHelper.OpenSession ( ))
        return session.Get <Product> ( productId ) ;
}
```

Для наступних двох методів ми використовуємо новий метод об'єкта session . Давайте почнемо з методу GetByName. Як завжди ми спочатку пишемо тест

```
[ Test ]
public void Can_get_existing_product_by_name ( )
{
    IProductRepository repository = new
ProductRepository ( );
    var fromDb = repository.GetByName ( _products [ 1
] . Name ) ;

    Assert.IsNotNull ( fromDb ) ;
    Assert.AreNotSame ( _products [ 1 ] , fromDb ) ;
}
```



```
Assert.AreEqual ( _products [ 1 ] . Id ,  
fromDb.Id ) ;  
}
```

Реалізувати даний метод можна двома шляхами. Перше це використовувати HQL (Hibernate Query Language ) і друге це HCQ ( Hibernate Criteria Query) . Давайте почнемо з HQL . HQL це об'єктно -орієнтована мова запитів схожий (але не ідентичний) на SQL.

Таким чином показано як реалізувати простий базовий домен, mapping файл до БД і як налаштувати NHibernate з можливістю збереження об'єктів домену в БД.

#### 4.4 Зміст звіту

Звіт має містити: мету роботи; порядок виконання роботи у вигляді екранних форм, тексти програм; формулювання та результати завдань для самостійного виконання.

#### 4.5 Контрольні запитання та завдання

1. Для чого використовується технологія NHibernate?
2. Охарактеризуйте достоїнства та недоліки технології NHibernate.
3. Яким чином можна можна використовувати NHibernate mapping для завантаження і збереження бізнес об'єктів?
4. Створіть веб-застосування на підставі технології NHibernate, яке використовує створену у лабораторній роботі№1 базу даних.
5. Організуйте перегляд даних. Додайте фільтр, пошук інформації за обраними критеріями.
6. Виконайте модифікацію даних.
7. Протестуйте створене застосування.

## 5 МІГРАЦІЯ ЗАСТОСУВАНЬ ASP.NET НА WINDOWS AZURE.

### 5.1 Мета роботи

Набуття практичних навичок і закріплення теоретичних відомостей з використання технології NHibernate.Net.

### 5.2 Методичні вказівки до виконання роботи

Ознайомитися з матеріалами лекцій. Звернути увагу на різні області застосування сховищ даних у Windows Azure.

Технологія ASP.NET забезпечує підтримку безлічі запропонованих постачальниками додатків варіантів реалізації управління членством, ролями, профілями і сеансами. Більшість постачальників поставляється в версії, засновані на базі даних SQL, або використовуює уявлення даних, керованих постачальниками, в оперативній пам'яті. Приклади Windows Azure включають реалізації постачальників, що використовують масштабовані і надійні служби сховищ таблиць і BLOB-об'єктів. Крім того, постачальники стикаються з проблемою веб-додатків, розміщених на різних комп'ютерах в структурі Windows Azure. Служби сховища таблиць і BLOB-об'єктів доступні відразу ж при розгортанні веб-додатки в центрах обробки даних Windows Azure, завдяки чому доступ до них з програми спрощується.

Azure Store - стандартний зразок ASP.NET додатки, що імітує просте комерційне додаток. Воно являє собою список, з якого користувачі можуть вибирати певні продукти і додавати їх в свою корзину покупок.

### 5.3 Порядок виконання роботи та методичні вказівки з її виконання

Можливо, перед початком виконання вправи вам буде потрібно побудувати і запустити рішення, щоб ознайомитися з його роботою. У початковому стані додаток працює поза комп'ютерного емулятора.

При виконанні цього завдання, щоб підготувати додаток до запуску в Windows Azure, необхідно створити в Visual Studio проект Windows Azure.

Запустіть Visual Studio в режимі підвищених адміністративних привілеїв і вибравши пункт Run as administrator.

Якщо з'явиться діалогове вікно User Account Control, натисніть Yes.

Відкрийте наявний проект.

Потім створіть новий проект хмарної служби та додайте його до вирішення. В меню File виберіть пункт Add, потім New. У діалоговому вікні Add New Project розгорніть пункт Visual C # в списку Add New Project і виберіть Cloud. Виберіть шаблон Windows Azure Project, встановіть ім'я проекту - CloudShopService, збережіть передбачуване розташування папки з рішенням і натисніть кнопку OK

У діалоговому вікні New Windows Azure Project виберіть OK, не додаючи нових ролей до вирішення, оскільки існуючий додаток буде використовуватися як веб-роль.

Зв'яжіть проект ASP.NET з хмарним проектом. У браузері рішень Solution Explorer клацніть правою кнопкою миші по вузлу Roles проекту CloudShopService, виберіть Add, а потім Web Role Project in solution.

У діалоговому вікні Associate with Role Project, виберіть проект CloudShop і натисніть OK. При зв'язуванні з новою роллю Visual Studio оновлює файли ServiceDefinition.csdef і ServiceConfiguration.cscfg. Якщо який-небудь з цих файлів в цей час відкрито, необхідно зберегти зроблені зміни.

Додайте посилання на збірки, необхідні для підтримки середовища Azure. У браузері рішень Solution Explorer клацніть правою кнопкою на проекті CloudShop, виберіть Add Reference, клацніть вкладку .NET, виберіть компоненти Microsoft.WindowsAzure.Configuration, Microsoft.WindowsAzure.Diagnostics, Microsoft.WindowsAzure.ServiceRuntime і Microsoft.WindowsAzure.Storage і натисніть OK.

Для підключення діагностичного протоколювання додатку налаштуйте TraceListener. Щоб налаштувати, відкрийте файл Web.config проекту CloudShop і вставте розділ system.diagnostics як зазначено нижче.

```
...
<System.diagnostics>
<Trace autoflush = "false" indentsize = "4">
<Listeners>
<Add name = "AzureDiagnostics" type =
"Microsoft.WindowsAzure.Diagnostics.DiagnosticMonitorTraceListener
, Microsoft.WindowsAzure.Diagnostics, Version = 1.8.0.0, Culture =
neutral, PublicKeyToken = 31bf3856ad364e35" />
listeners>
trace>
system.diagnostics>
```

Такі установки розділу `system.diagnostics` забезпечують настройку прослуховувача трасування для використання з Windows Azure, що дозволяє додатку здійснювати трасування виконання коду за допомогою класів і методів, доступних в класі `System.Diagnostics.Trace`. Як правило, цей етап можна пропустити при роботі з ролями, розробленими в Visual Studio, оскільки всі необхідні установки вже включені в їх шаблони.

Оголосіть наступні простору імен в файлі `Global.asax` веб-полі.

```
using Microsoft.WindowsAzure;  
using Microsoft.WindowsAzure.ServiceRuntime;
```

При виконанні MVC проекту ASP.NET необхідно упевнитися, що збірка `System.Web.Mvc` включена в пакет служби, розгортання якої здійснюється в Windows Azure. Для цього розгорніть вузол `References` в браузері рішень `Solution Explorer` проекту `CloudShop`, клацніть правою кнопкою миші збірку `System.Web.Mvc` і виберіть пункт `Properties`. Змініть значення параметра `Copy Local` на `True`.

Зазвичай в будь-якому складанні, яке не встановлено за замовчуванням на віртуальні машини Windows Azure, необхідно встановити значення параметра `Copy Local = True`, щоб забезпечити його розгортання з додатком.

## 5.4 Зміст звіту

Звіт має містити: мету роботи; порядок виконання роботи у вигляді екранних форм, тексти програм; формулювання та результати завдань для самостійного виконання.

## 5.5 Контрольні запитання та завдання

1. Що являють собою підписки і яким чином вони пов'язані з безкоштовним обліковим записом Azure?
2. У чому полягає політика життєвого циклу для хмарних служб Azure?
3. Яким чином здійснюється підключення до сховища даних Azure?
4. Створити проект з використанням поставників Azure ASP.NET із застосуваннями MVC
5. Створити проект з використанням поставників Azure ASP.NET із застосуваннями з веб-формами.

## ПЕРЕЛІК ПОСИЛАНЬ

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#.— М.: «Русская Редакция», Спб: «Питер», 2013.— 896 с.
2. Троелсен, Э. Язык программирования C# 2010 и платформа .NET 4.0— Изд-во: «Вильямс», 2011.—1392 с.
3. Нортроп, Т. Основы разработки приложений на платформе Microsoft .NET Framework. Учебный курс Microsoft [Текст]/ Т.Нортроп, Ш.Уилдермьюс, Б.Райан — М.: «Русская Редакция», Спб: «Питер», 2007.— 864 с.
4. Чедвик Д. ASP.NET MVC 4. Разработка реальных веб-приложений с помощью ASP.NET MVC [Текст]/ Д. Чедвик - М.: Вильямс. ,2013. — 432 с.
5. Руководство по ASP.NET MVC 5 [Электронный ресурс] / Сайт METANIT.COM / Режим доступа: <http://metanit.com/sharp/mvc5/1.1.php> — 10.10.2015 г. — Загол. 3 экр.
6. Фримен А. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов [Текст] / Фримен А. /М.: Изд: Вильямс — 2015. 736 стр., с ил.; ISBN 978-5-8459-2008-9

Електронне навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ  
до лабораторних робіт з дисципліни  
"\*Робота з даними на платформі .NET "

для студентів усіх форм навчання  
напряму 6.050103 Програмна інженерія  
спеціальності 121 – «Інженерія програмного забезпечення»

Упорядники: ШИРОКОПЕТЛЄВА Марія Сергіївна

Відповідальний випусковий З.В.Дудар

Авторська редакція

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

КОНТРОЛЬНІ ЗАВДАННЯ

з дисципліни

"\*Робота з даними на платформі .NET "

для студентів усіх форм навчання  
напряму 6.050103 - Програмна інженерія,  
спеціальності 121 – Інженерія програмного забезпечення  
освітня програма «Програмна інженерія»

Електронне видання

Затверджено  
кафедрою ПІ  
Протокол № 4  
від 17.10.2017р.

Харків 2017

Контрольні завдання з дисципліни «\*Робота з даними на платформі .NET» для студентів усіх форм навчання напряму 6.050103 - Програмна інженерія, спеціальності 121 – Інженерія програмного забезпечення [Електронне видання] / Упоряд.: М.С.Широкопетлева. – Харків: ХНУРЕ, 2017. – 9 с.

Упорядник: М.С.Широкопетлева

©М.С.Широкопетлева 2017 рік



## ЗМІСТ

1 ЗАВДАННЯ НА КОНТРОЛЬНУ РОБОТУ .....	4
2 ПРИКЛАДИ ЗАВДАНЬ ДЛЯ ПРОМІЖНОГО КОНТРОЛЯ ЗНАНЬ .....	5
3 КОНТРОЛЬНІ ПИТАННЯ .....	6

## 1 ЗАВДАННЯ НА КОНТРОЛЬНУ РОБОТУ

Домашня контрольна робота представляє проектування веб-додатка, що демонструє роботу з базою даних, реалізованою в СУБД MS SQL Server та передбачає демонстрацію закріплення всього матеріалу з даної тематики.

Вимоги до веб-застосування:

- база даних має містити не менш ніж 3 пов'язані між собою заповнені таблиці (для демонстрації роботи підпрограм), по яких створені необхідні індекси;
- використання підходу Database First (при використанні Code First – треба навести обґрунтування вибору такого підходу);
- використання ORM;
- наявність валідації даних.

Тематика контрольної роботи обирається студентом довільно, можливо на підставі раніше виконаної курсової роботи з дисциплін “Бази даних” або «Аналіз та рефакторинг програмного забезпечення».

## 2 ПРИКЛАДИ ЗАВДАНЬ ДЛЯ ПРОМІЖНОГО КОНТРОЛЯ ЗНАНЬ

1. Візьміть тестову або яку-небудь іншу готову базу даних і на її основі створюєте edmx -модель , з довільного числа таблиць , зв'язків і збережених процедур.

Виконайте будь-які дії з видалення , перейменування створенню нових сутностей і властивостей у них. Складіть асоціації на основні дії в таблицях (Update , Insert , Delete) за допомогою збережених процедур.

Виконайте відкладені запити до моделі, і виконайте деякі маніпуляції з отриманими даними. Візуально відобразіть отримані результати.

Опишіть послідовність налагодження і тестування створеної веб-сторінки.

Створити проект з використанням поставників Azure ASP.NET із застосуваннями MVC

Створити проект з використанням поставників Azure ASP.NET із застосуваннями з веб-формами.

### 3 КОНТРОЛЬНІ ПИТАННЯ

1. Що містить маніфест метаданих?
2. Опишіть порядок створення пари ключів?
3. Які вимоги висуваються до збірок для розміщення їх в GAC?
4. Що таке регіональний стандарт?
5. Що таке колізія імен (для GAC)?
6. Який інструмент використовують в період розробки та тестування збірок із суворим ім'ям для установки їх в GAC?
7. Що таке делегат?
8. Охарактеризуйте ланцюжок делегатів.
9. За допомогою якого метода класу `MultiCastDeleagate` можна викликати будь-який з делегатів ланцюжка?
10. В яких випадках не можна використовувати узагальнені делегати?
11. Які основні переваги Entity Framework перед іншими платформами (LINQ to SQL, ADO.NET)?
12. Розкажіть про архітектурі платформи Entity Framework?
13. Що таке LINQ to Entity?
14. Що таке контекст об'єкта? Яким класом він представлений в Entity to SQL?
15. Які дії можна робити в Entity Model з сутностями моделі?
16. Що таке відкладене виконання запиту?
17. Що таке негайне виконання запиту?
18. Що таке скомпільований запит?
19. Які існують недоліки в LINQ to Entity?
20. Простір назв `System.Linq`.
21. Технічні засоби реалізації технології Linq.
22. Синтаксис Linq -запитів.
23. Умови відбору, оператор зв'язування, проекції, створення об'єктів анонімних класів у проекторі.
24. Використання результатів Linq -запитів.
25. Класи для доступу до даних.
26. Постачальники даних.
27. Класи з'єднання.
28. Створення рядків і об'єктів з'єднання.

29. Класи команд.
30. Модель взаємодії з постійним доступом.
31. Виконання команд.
32. Модель з рассоединением.
33. Клас DataSet.
34. Класи таблиць і зв'язків.
35. Конструктори запитів.
36. Основи технології об'єктно- реляційного зв'язування.
37. Створення моделі класів для технології Linq to SQL.
38. Клас контексту даних.
39. Створення бази даних за моделлю класів.
40. Створення моделі класів по базі даних.
41. Утиліта svcutil.
42. Концепція моделювання даних.
43. Відділення концептуальної структури даних від логічної структури в сховищі даних.
44. Створення , оновлення та видалення сутностей даних.
45. Способи модифікації даних на платформі Entity Framework.
46. Механізм відстеження змін.
47. Обробка багатокористувацьких сценаріїв за допомогою сервісів об'єкт
48. Модель паралельного доступу до даних.
49. Використання транзакцій для забезпечення цілісності даних.
50. Налаштування сутностей і створення користувацьких класів сутностей.
51. Налаштування та розширення сутностей за допомогою власної бізнес -логіки.
52. Використання традиційних об'єктів середовища CLR ( POCO ) з Entity Framework.
53. Способи визначення настроюваних об'єктів класів у Entity Framework.
54. За замовчуванням класи сутностей створюються в моделі EDM.
55. Створення багаторівневих рішень за допомогою Entity Framework.
56. Архітектурні проблеми при створенні багаторівневих додатків і їх рішення за допомогою Entity Framework.
57. Обробка оновлень в багаторівневому вирішенні за допомогою Entity Framework.

- 58. Способи обробки змін даних в багаторівневому вирішенні і керування винятками.
- 59. Створення рішень з непостійним підключенням до джерел даних.
- 60. Доступ до автономних або довільно підключаємих джерел даних в клієнтських додатках.
- 61. Кешування даних в локальних файлах XML за допомогою LINQ to XML і реалізація довільно підключаємих програм за допомогою Sync Framework.
- 62. Особливості доступу до даних у Windows Azure.

Електронне навчальне видання

## КОНТРОЛЬНІ ЗАВДАННЯ

з дисципліни

«\*Робота з даними на платформі .NET»

для студентів усіх форм навчання  
напряму 6.050103 – Програмна інженерія,  
спеціальності 121 – Інженерія програмного забезпечення  
освітня програма «Програмна інженерія»

Упорядники: ШИРОКОПЕТЛЄВА Марія Сергіївна

Відповідальний випусковий З.В.Дудар

Авторська редакція