

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Кафедра програмних систем і технологій

ДУХНОВСЬКА К.К., НІКОЛАЄНКО А.Ю.

SQL

Вид видання: *методичні вказівки*

Київ-2023

РЕЦЕНЗЕНТИ:

Доцент кафедри електронних пристроїв та систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря

Сікорського», кандидат технічних наук, Є. Ю. Коваленко

Кандидат технічних наук, доцент кафедри комп'ютеризованих систем управління Національного авіаційного університету Євген АРТАМОНОВ

Рекомендовано до публікації кафедрою програмних систем і технологій

Факультету інформаційних технологій

Київського національного університету

Імені Тараса Шевченка

протокол №13 від 14.03.2023

Духновська К.К., Ніколаєнко А.Ю.

SQL [Електронний ресурс]: методичні вказівки. Кафедра програмних систем і технологій/Духновська К.К. – К. : КНУ імені Тараса Шевченка, 2024. – 185 с.

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «SQL» для студентів денної форми навчання зі спеціальності 121 – «Інженерія програмного забезпечення» за освітньо-професійною програмою «Інженерія програмного забезпечення» освітнього рівня «Бакалавр»

Публікується в авторській редакції.

Електронна версія цього видання опублікована на сайті

кафедри програмних систем і технологій за адресою:

<https://pst.knu.ua/>

(дата публікації «01»листопада 2024 р.)

Зміст

Вступ.....	9
Лабораторна робота №1.....	12
Тема: SQL Server і командний рядок SQL.....	12
1.1. Інсталяція MS SQL Server 2022.	12
1.2. Робота з MS SQL Server в командному рядку.....	13
Приклад 1.1. Запуск sqlcmd.....	14
Приклад 1.2. Перевірка бази даних за замочуванням	15
Приклад 1.3. Список баз даних MS SQL Server.....	15
Приклад 1.4. Список баз даних MS SQL Server.....	15
Приклад 1.5. Перевірка середовища на чутливість до регістру	16
Приклад 1.6. Перевірка середовища на чутливість до регістру	17
Приклад 1.7. Перевірка випуску MS SQL Server.....	17
Приклад 1.8. Як перевірити автентифікацію SQL Server у sqlcmd.....	17
Приклад 1.9. Як перерахувати змінні середовища?	18
ЗАВДАННЯ	18
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	19
Лабораторна робота №2.....	20
Тема: Створення і видалення бази даних. Створення і видалення таблиці.	
SQL Server Management Studio	20
2.1 Створення бази даних за допомогою командного рядка	20
2.2 Видалення бази даних за допомогою командного рядка.....	25
2.3 Створення таблиці бази даних за допомогою командного рядка	25
2.4 Видалення таблиці бази даних за допомогою командного рядка.....	26
2.5 Створення і видалення бази даних в середовищі SQL Server Management Studio	27
2.6 Створення, заповнення і видалення таблиці бази даних в середовищі SQL Server Management Studio.....	31
ЗАВДАННЯ	36
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	39
Лабораторна робота №3.....	41

Тема: Робота з Azure Data Studio	41
3.1 Azure Data Studio	41
3.2 Додавання даних в таблицю.....	45
ЗАВДАННЯ	46
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	48
Лабораторна робота №4.....	49
Тема: Вибірки даних, редагування і видалення даних	49
4.1 Вибірки даних.....	49
Приклад 4.1.....	53
Приклад 4.2.....	56
Приклад 4.3.....	56
4.2 Сортування. ORDER BY	58
Приклад 4.4.....	58
Приклад 4.5.....	58
Приклад 4.6.....	59
4.3 TOP – вибірка з вказаної кількості перших рядків таблиці.....	60
Приклад 4.7.....	60
Приклад 4.8.....	61
4.4 OFFSET і FETCH – діапазон вибірки.....	61
Приклад 4.9.....	62
Приклад 4.10.....	62
4.5 WHERE – фільтрація	62
Приклад 4.11.....	63
4.6 Логічні оператори	63
Приклад 4.12.....	64
4.7 IS NULL.....	64
Приклад 4.13.....	64
4.8 Оператори фільтрації.....	65
Приклад 4.14.....	65
Приклад 4.15.....	66
4.9 Видалення даних. Команда DELETE	66
4.10 Оновлення даних. Команда UPDATE	67

Приклад 4.16.....	67
ЗАВДАННЯ	67
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	69
Лабораторна робота №5.....	71
Тема: Багатотабличні запити. Запити на з'єднання.....	71
5.1 З'єднання таблиць	71
5.2 Додавання даних до зв'язаних таблиць	75
Приклад 5.1.....	76
5.3 Декартів добуток таблиць	78
Приклад 5.2.....	78
Приклад 5.3.....	79
5.4 З'єднання таблиць	79
Внутрішні з'єднання.....	79
Приклад 5.4.....	79
Зовнішні з'єднання.	80
Приклад 5.5.....	80
Приклад 5.6.....	81
Перехресні з'єднання.....	81
Приклад 5.7.....	82
ЗАВДАННЯ	82
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	85
Лабораторна робота №6.....	86
Тема: Створення облікових записів, користувачів, ролей	86
6.1 Система безпеки SQL сервера	86
6.2 Створення облікового запису	87
Приклад 6.1.....	87
Приклад 6.2.....	88
6.3. Створення користувача	89
Приклад 6.3.....	89
Приклад 6.4.....	92
6.4 Створення ролей.....	92
Приклад 6.5.....	95

Приклад 6.6.	95
ЗАВДАННЯ	96
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	98
Лабораторна робота №7.....	98
Тема: Групування. Вбудовані функції	98
7.1 Вбудовані функції	99
7.2 Агрегатні функції	100
Приклад 7.1.	101
Приклад 7.2.	102
Приклад 7.3.	102
Приклад 7.4.	103
7.3 Символьні функції	103
Приклад 7.5.	104
7.4 Числові функції	105
7.5. Часові функції.....	105
Приклад 7.6.	106
Приклад 7.7.	106
Приклад 7.8.	106
Приклад 7.9.	107
Приклад 7.10.	107
Приклад 7.11.	107
Приклад 7.12.	107
Приклад 7.13.	108
7.6 Запити з угрупованням рядків	108
Приклад 7.14.	108
Приклад 7.15.	109
Приклад 7.16.	111
ЗАВДАННЯ	111
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	113
Лабораторна робота №8.....	114
Тема: Основи програмування за допомогою вбудованої мови SQL.	
Подання	114

8.1 Арифметичні і логічні операції	114
8.2 Ідентифікатори та змінні	115
Приклад 8.1.	116
Приклад 8.2.	116
Приклад 8.3.	116
Приклад 8.4.	117
Приклад 8.5.	117
Приклад 8.6.	117
8.3 Операторські дужки	118
8.4 Умовна конструкція IF	118
Приклад 8.7.	118
8.5 Цикл WHILE	119
Приклад 8.8.	119
8.6 Функції і збережені процедури	120
Приклад 8.9.	124
Приклад 8.10.	125
Приклад 8.11.	127
Приклад 8.12.	127
8.7 Подання	128
ЗАВДАННЯ	134
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	136
Лабораторна робота №9	138
Тема: Тригери. Транзакції	138
9.1 Створення, програмування і управління тригерами	138
Приклад 9.1.	139
9.2. Транзакції	141
Приклад 9.2.	143
Приклад 9.3.	144
Приклад 9.4.	145
Приклад 9.5.	146
ЗАВДАННЯ	147
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	148

Лабораторна робота №10.....	149
Тема: Розроблення програмних застосунків з БД.....	149
10.1 Введення в ADO.NET	149
10.2 C# і MS SQL Server	152
Приклад 10.1. Рядок підключення.....	152
Приклад 10.2. Створення підключення	156
Приклад 10.3. Отримання інформації про підключення.....	157
Приклад 10.4. Можливі помилки та їх вирішення.....	158
Приклад 10.5. Виконання команд і SqlCommand	159
Приклад 10.6. Додавання об'єктів.....	161
Приклад 10.7. Оновлення об'єктів	162
Приклад 10.8. Видалення об'єктів	163
Приклад 10.9. Читання результатів запиту і SqlDataReader	166
10.3 Визначення контексту даних і моделей.....	167
10.4 Фільтрація і сортування.....	171
10.5. Зміна об'єктів в LINQ to SQL	171
10.6 Додавання об'єктів в LINQ to SQL	173
10.7 Видалення об'єктів в LINQ to SQL	174
10.8 Платформа Entity Framework	175
10.9. Встановлення пакету EF6.....	176
10.10 Підходи Code First до існуючої бази даних.....	178
10.11 Додавання об'єктів через DbSet	179
10.12 Редагування об'єктів через DbSet	180
10.13 Видалення об'єктів через DbSet	181
ЗАВДАННЯ	182
ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ	182
ЛІТЕРАТУРА.....	183

Вступ

Навчальна дисципліна «SQL» входить до складу освітньо-професійної програми «Інженерія програмного забезпечення» для студентів освітнього рівня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення» галузі знань 12 «Інформаційні технології».

Метою дисципліни є формування знань, вмінь і навичок студентів щодо створення сучасного програмного забезпечення із використанням реляційних бази даних та SQL, засобів тестування, вимірювання ефективності запитів та їх оптимізації.

Завдання дисципліни полягає у формуванні у студента знань про реляційні системи управління базами даних. У результаті вивчення навчальної дисципліни студент повинен засвоїти різні типи запитів до реляційних даних, базові типи даних та функції, які визначені на них, принципи оптимізації запитів, вміти писати оператори для додавання та модифікації інформації в базі даних; на основі аналізу задачі, що розробляється, обирати найбільш раціональні оператори запитів, що забезпечують ефективну реалізацію задачі. Студент має володіти знаннями та вміннями виконувати операції фільтрації реляцій за умовою, з'єднання реляцій. Студент повинен уміти виконувати теоретико-множинні операції на реляціях, писати запити для групування даних, сортування результатів запитів за різними критеріями, вміти працювати з невизначеною інформацією, об'єднувати запити в транзакції. У результаті вивчення дисципліни студент повинен вміти працювати зі структурованою інформацією.

Дисципліна базується на знаннях, вміннях і практичних навичках, отриманих студентами при вивченні дисциплін «Математичні основи програмної інженерії», «Прикладна математика», «Комп'ютерні методи аналізу дискретних об'єктів», «Основи програмування», «Алгоритми та структури даних», «Вступ до об'єктно-орієнтованого програмування» (рис. В.1).

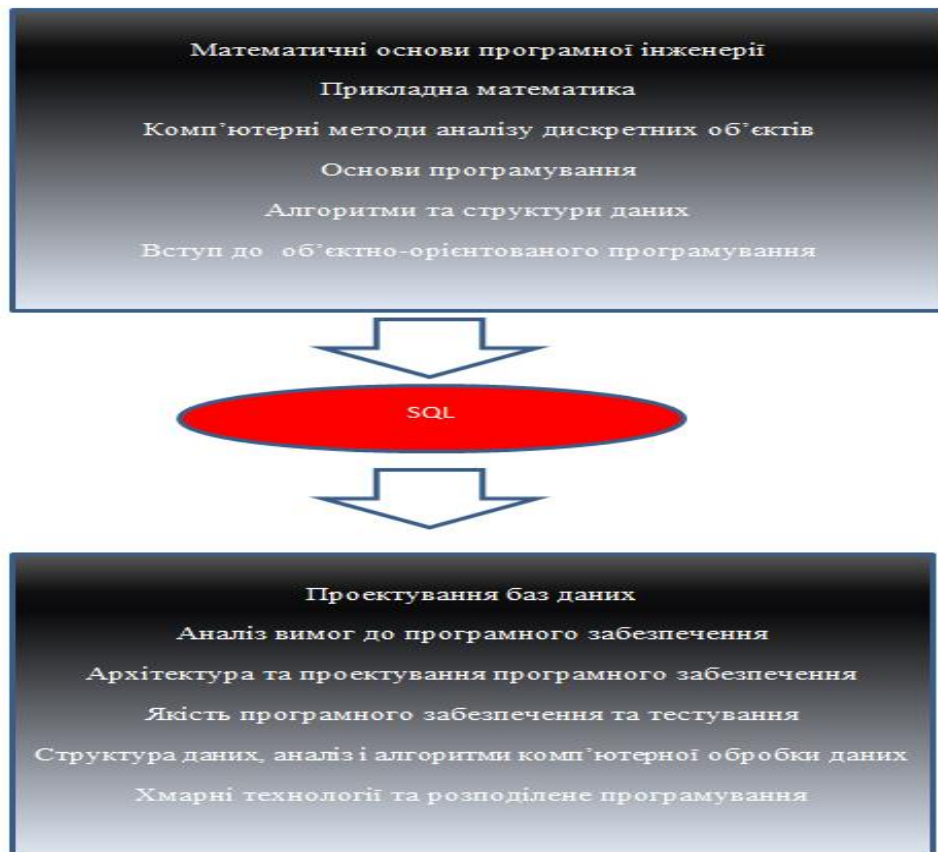


Рис. В.1 Місце дисципліни у навчальному процесі

Перша лабораторна робота присвячена розгляду систем управління базами даних та інструментів для комфортної роботи з ними. Також в ній розглянуто створення елементарних записів у консолі SQL. У другій лабораторній роботі наведено огляд створення і видалення реляційної бази даних і таблиць за допомогою запитів та інструментів SQL Server Management Studio. Третя лабораторна робота присвячена знайомству з Azure Data Studio. У четвертій лабораторній роботі розглянуто елементарні однотабличні запити. П'ята лабораторна робота присвячена багатотабличним запитам, запитам на з'єднання та їх оптимізації. Шоста лабораторна робота знайомить з механізмами безпеки в SQL Server: обліковими записами, користувачами бази даних і їх ролями. У сьомій лабораторній роботі розглянуто групування даних і вбудовані функції. Предметом восьмої лабораторної роботи є основи програмування за допомогою вбудованої мови SQL і подання. У дев'ятій лабораторній роботі наведено огляд роботи з тригерами і транзакціями. У заключній десятій лабораторній роботі розглянуто вбудовування бази даних в

програмний застосунок за допомогою технологій LINQ, ADO.NET, Entity Framework.

Після проходження дисципліни студент має опанувати не тільки мову запитів SQL, а й допоміжні інструменти для роботи з SQL Server: SQL консоль, SQL Server Management Studio, Azure Data Studio.

Лабораторна робота №1

Тема: SQL Server і командний рядок SQL

План

1. Інсталяція MS SQL Server 2022;
2. Робота з консоллю;
3. Завдання;
4. Питання для самоперевірки.

1.1. Інсталяція MS SQL Server 2022.

Існують різні випуски MS SQL Server, серед них безкоштовні – це Express і Developer. Для простих застосунків може вистачити випуску Express. У нього є перевага – його можна використовувати як серверний застосунок в реальних задачах, однак він має урізаний функціонал в порівнянні з повною версією.

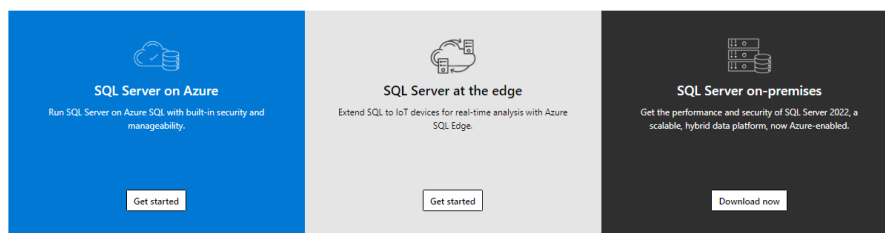
MS SQL Server Developer Edition – це повнофункціональний випуск, тобто він містить весь функціонал, націлений тільки для потреб розроблення, як і повна версія MS SQL Server Enterprise. У той же час ця версія не може бути використана для розгортання серверного застосунку в реальних проєктах.

Версія Developer Edition є оптимальним варіантом для вивчення всього спектру MS SQL Server, тому саме цю версію буде розглянуто далі.

Для завантаження MS SQL Server 2022 Developer Edition перейдіть за посиланням <https://www.microsoft.com/en-us/sql-server/sql-server-downloads> (рис. 1.1).

Завантажувати інсталятор MS SQL Server потрібно той, який буде працювати саме з вашою операційною системою. Ще можна встановити образ контейнера SQL Server 2022 за допомогою Docker – платформи для управління контейнерами (рис. 1.2).

Try SQL Server on-premises or in the cloud



Or, download a free specialized edition



Рис. 1.1 Сторінка для встановлення MS SQL Server

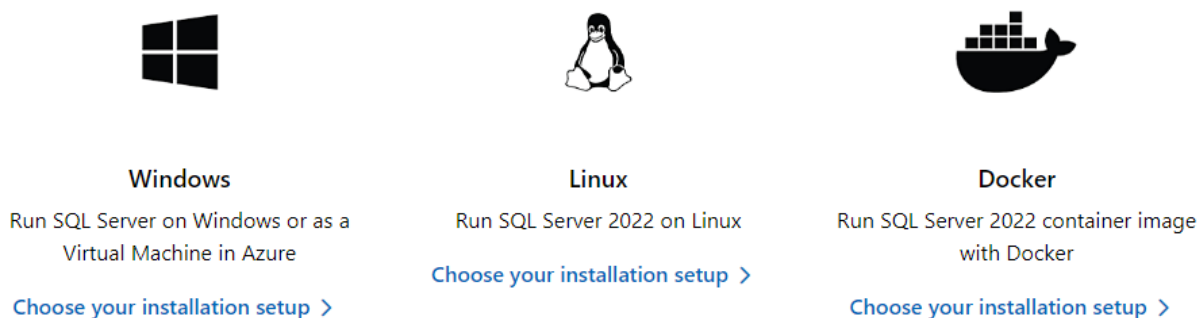


Рис. 1.2 Вибір інсталятора MS SQL Server залежно від операційної системи

Також на цьому сайті є інсталятори інструментів для більш комфортної роботи з MS SQL Server. Нам знадобляться SQL Server Management Studio та [Azure Data Studio](#) (рис. 1.3).

Після встановлення всіх цих програм ви отримаєте на своїй машині систему управління базами даних ([СУБД](#)) MS SQL Server та інструменти, які спростують роботу з ними.

1.2. Робота з MS SQL Server в командному рядку

Сервер СУБД не має інтерфейсу користувача. Для виконання операцій з базою даних йому потрібно надсилати команди або за допомогою командного рядка, або за допомогою будь-якої прикладної програми.

Працювати в консолі (командному рядку) дає змогу службова програма **sqlcmd**. Її область застосування: SQL Server (усі підтримувані версії), база

даних SQL Azure, керований екземпляр SQL Azure, Azure Synapse Analytics, Analytics Platform System (PDW).

SQL Server tools and connectors

Tools

[Download Azure Data Studio](#)

[Download SQL Server Management Studio \(SSMS\)](#)

[Download SQL Server Data Tools \(SSDT\)](#)

[Download Data Migration Assistant](#)

[Download SQL Server Migration Assistant for Oracle](#)

Connectors

[Microsoft ADO.NET for SQL Server](#)

[Microsoft JDBC Driver for SQL Server](#)

[Microsoft ODBC Driver for SQL Server](#)

[Node.js Driver for SQL Server](#)

[Python Driver for SQL Server](#)

[Ruby Driver for SQL Server](#)

Рис. 1.3 Інструменти і конектори для MS SQL Server

Sqlcmd дає змогу виконувати запити, вирази T-SQL і сценарії SQL Server за допомогою командного рядка.

Приклад 1.1. Запуск sqlcmd

Відкрийте консоль (командний рядок, cmd). Консоль операційної системи Windows можна викликати декількома шляхами, наприклад, у меню пуск знайти програму Командний рядок, або «Виконати» (комбінацією клавіш Win + X) і набрати cmd.

Формат запуску службової програми sqlcmd такий:

```
sqlcmd [ /E ] [ /S servername ]
```

Наприклад, щоб запустити sqlcmd і з'єднатися із сервером на ім'я DESKTOP-BJ9ERKQ\SQLEXPRESS, користувач Users має набрати такий код:

```
C:\Users\1>sqlcmd -S DESKTOP-BJ9ERKQ\SQLEXPRESS -E
1>
```

Цифра 1> означає, що є підключення і програма sqlcmd готова приймати Ваші команди для виконання.

Якщо ввімкнена автентифікація SQL Server, то потрібно буде вказати ім'я користувача та пароль користувача:

```
sqlcmd -S DESKTOP-BJ9ERKQ\SQLEXPRESS -U ім'я_користувача
```

а коли і пароль задаєте, то:

sqlcmd -S DESKTOP-BJ9ERKQ\SQLEXPRESS -U ім'я_користувача -P
далі повинні ввести пароль.

Приклад 1.2. Перевірка бази даних за замочуванням

Для того, щоб подивитися, яка база даних зазначена за замочуванням, наберіть такий код:

```
1> select DB_NAME()  
2> GO  
  
-----  
-----  
master  
  
(1 rows affected)  
1>
```

База даних master – це системна база даних MS SQL Server, яка зберігає всю інформацію сервера, включаючи облікові записи сервера, відомості про всі бази даних та їх користувачів.

Приклад 1.3. Список баз даних MS SQL Server

Результатом виконання такої команди:

```
1> select name from sys.databases  
2> go  
name  
  
-----  
-----
```

буде перераховано всі бази даних в екземплярі SQL:

```
-----  
master  
  
tempdb  
  
model  
  
msdb
```

Приклад 1.4. Список баз даних MS SQL Server

Для отримання списку баз даних також можна використовувати збережену процедуру sp_databases:

```
1> Sp_databases  
2> Go
```


Приклад 1.6. Перевірка середовища на чутливість до регістру

Цю інформацію можливо перевірити за допомогою процедури sp_helpsort:

```
(1 rows affected)
1>
2> sp_helpsort
3> go
Server default collation

-----
Ukrainian, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
```

Приклад 1.7. Перевірка випуску MS SQL Server

Випуск SQL Server можливо перевірити, використовуючи такі скрипти T-SQL:

```
1> SELECT SERVERPROPERTY('EDITION')
2> GO
```

Результат у мене:

```
-----
Express Edition (64-bit)
```

Приклад 1.8. Як перевірити автентифікацію SQL Server у sqlcmd

До SQL Server було два варіанти автентифікації на SQL Server:

- Автентифікація Windows, де можна використовувати обліковий запис Active Directory або локальний обліковий запис Windows.
- Автентифікація Windows і SQL, де ви також можете пройти автентифікацію за допомогою облікового запису, створеного в SQL Server.

Щоб перевірити автентифікацію, можливо використовувати такі скрипти:

```
(1 rows affected)
1> SELECT SERVERPROPERTY('IsIntegratedSecurityOnly')
2> GO
```

Результат виконання цього скрипта може бути таким:

```
-----
1
```

Якщо результат 0, це означає, що обидві автентифікації ввімкнено. Якщо значення 1, ввімкнено лише автентифікацію Windows.

Приклад 1.9. Як перерахувати змінні середовища?

Щоб отримати список усіх змінних середовища, виконайте таку команду в sqlcmd:

```
1> :ListVar
```

На екрані відобразиться схожий результат:

```
SQLCMDCOLSEP = " "
SQLCMDCOLWIDTH = "0"
SQLCMDDBNAME = ""
SQLCMDEDITOR = "edit.com"
SQLCMDERRORLEVEL = "0"
SQLCMDHEADERS = "0"
SQLCMDINI = ""
SQLCMDLOGINTIMEOUT = "8"
SQLCMDMAXFIXEDTYPEWIDTH = "0"
SQLCMDMAXVARTYPEWIDTH = "256"
SQLCMDPACKETSIZE = "4096"
SQLCMDSERVER = "DESKTOP-BJ9ERKQ\SQLEXPRESS"
SQLCMDSTATTIMEOUT = "0"
SQLCMDUSER = "1"
SQLCMDWORKSTATION = "DESKTOP-BJ9ERKQ"
1>
```

ЗАВДАННЯ

1. Встановити MS SQL Server та інструменти для роботи з ним.
2. Виконати всі приклади із лабораторної роботи.
3. Вивчити всі змінні середовища MS SQL Server (приклад 1.9), надати їх опис, пояснити чому саме такі значення у них.
4. Зробити звіт.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Дайте визначення поняття «база даних» і «таблиця».
2. Що таке система керування базою даних? Наведіть приклади таких систем.
3. Що зберігається в базі даних master?
4. Що таке SQL?
5. Де і коли SQL використовується?

Лабораторна робота №2

Тема: Створення і видалення бази даних. Створення і видалення таблиці. SQL Server Management Studio

План

1. Створення бази даних за допомогою командного рядка;
2. Видалення бази даних за допомогою командного рядка;
3. Створення таблиці бази даних за допомогою командного рядка;
4. Видалення таблиці бази даних за допомогою командного рядка;
5. Створення і видалення бази даних в середовищі SQL Server Management Studio;
6. Створення, заповнення і видалення таблиці бази даних в середовищі SQL Server Management Studio.
7. Завдання;
8. Питання для самоперевірки.

2.1 Створення бази даних за допомогою командного рядка

Самий простий спосіб створити базу даних набрати таку команду в sqlcmd:

```
1> CREATE DATABASE MyFirstDB  
2> GO
```

Таким чином створиться база даних з ім'ям MyFirstDB. Перевірити, чи вона дійсно створилась, можна таким чином:

```

1> select name from sys.databases
2> GO
name
-----
master
tempdb
model
msdb
CheckList
FirstDB
Education
Adress
Sales
MyFirstDB

```

Як видно з останнього рядка із скріна – база даних MyFirstDB на сервері є. У цьому випадку всі параметри створеної бази даних ідуть за замовчуванням. А самі файли з базою даних знаходяться в теці, в яку встановлений MS SQL Server, наприклад:

C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA.

Для створення бази даних (БД) Sales, яка буде зберігатися у файлі salesdat.mdf в теці D:\DATA\ (перед цим потрібно створити цю теку), наберіть такий код в будь-якому текстовому редакторі:

```

USE master;
GO
CREATE DATABASE Sales
ON
( NAME = Sale,
  FILENAME = 'D:\DATA\salesdat.mdf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 5 )
LOG ON
( NAME = Sale_log,
  FILENAME = 'D:\DATA\saleslog.ldf',
  SIZE = 5MB,
  MAXSIZE = 25MB,
  FILEGROWTH = 5MB );
GO

```

Шлях до теки можете вибрати інший, але врахуйте це, коли будете використовувати код, наданий у прикладі.

Скопіюйте цей код в командний рядок sqlcmd:

```

1> USE master;
2> GO
Changed database context to 'master'.
1> CREATE DATABASE Sales
2> ON
3> ( NAME = Sale,
4>   FILENAME = 'D:\DATA\salesdat.mdf',
5>   SIZE = 10,
6>   MAXSIZE = 50,
7>   FILEGROWTH = 5 )
8> LOG ON
9> ( NAME = Sale_log,
10>  FILENAME = 'D:\DATA\saleslog.ldf',
11>  SIZE = 5MB,
12>  MAXSIZE = 25MB,
13>  FILEGROWTH = 5MB );
14> GO

```

У результаті створення БД у теці DATA, з'являться два файли – salesdat.mdf і saleslog.ldf (рис. 2.1). Тобто на одну базу даних припадає як мінімум два файли: *.mdf і *.ldf. В першому зберігаються самі дані, таблиці, індекси тощо, а в другому знаходяться транзакції та інформація, яка необхідна для відновлення бази даних.

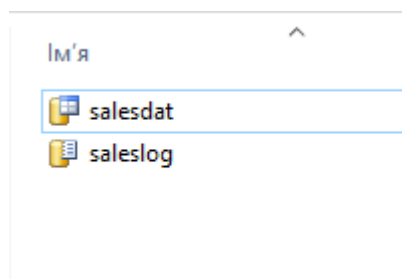


Рис. 2.1 Створена база даних Sales – файл з даними та файл з журналом транзакцій

Схематичне зображення скрипту створення БД подано на рис. 2.2.

```

CREATE DATABASE NameDB|
ON
(
*****
*****
)
LOG ON
(
*****
***** );

```

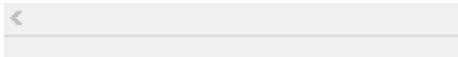


Рис. 2.2 Схематичний скрипт створення БД з ім'ям NameDB

CREATE DATABASE Sales – це команда для створення БД з ім'ям Sales. Ця команда містить параметри NAME, FILENAME, SIZE, MAXSIZE та FILEGROWTH, які описують створювані файли. Серед цих параметрів лише параметр FILENAME є обов'язковим.

БД може мати фізичне і логічне ім'я.

NAME = Sale – це параметр для задавання логічного імені, за яким можна звернутися до БД в скриптах. У наведеному вище прикладі логічне ім'я буде Sale.

FILENAME = 'D:\DATA\salesdat.mdf' – це параметр для задавання фізичного імені та повного шляху до файлу операційної системи, де зберігатиметься створювана БД. Зверніть увагу, що його значення є рядком, тобто має бути вказаним в одинарних лапках. Цей параметр вказує ім'я та шлях до файлу. Шлях обов'язково має вказувати на теку на обчислювальному пристрої, на якому інстальовано SQL Server. Не можна використовувати мережеві диски інших пристроїв.

SIZE = 10 – це параметр для задавання початкового розміру кожного створюваного файлу.

Усі створювані в SQL Server бази даних використовують деякий шаблон – база даних model, яка завжди має бути в системі з SQL Server. Вміст бази даних model (включаючи параметри) під час виконання команди CREATE DATABASE повністю копіюється у першу частину створюваної бази даних, а решта нової бази заповнюється порожніми сторінками. Тому розмір, вказаний

для основного файлу бази даних, повинен бути не меншим за розмір основного файлу бази даних model. У разі зміни бази даних model усі створені після цього бази даних успадкують ці зміни. Вихідний розмір за замовчуванням файлу журналу транзакцій та файлу даних для бази даних model становить 8 МБ.

У наведеному вище прикладі для файлу даних salesdat.mdf задано розмір 10 МБ, а для файлу транзакцій saleslog.ldf – 5 МБ. Відкрийте за допомогою файлового провідника теку D:\DATA\ та подивіться на розмір цих файлів (рис. 2.3).

Ім'я	Дата змінення	Тип	Розмір
 salesdat	01.02.2023 13:25	SQL Server Databa...	10 240 KB
 saleslog	01.02.2023 13:25	SQL Server Databa...	5 120 KB

Рис. 2.3 Тека зі створеними файлами заданого розміру

В процесі заповнення БД ці файли будуть збільшуватись. Розмір файлів можна контролювати, встановивши максимальний розмір і крок збільшення файлів.

MAXSIZE – максимальний розмір файлу, ціле число. Якщо не вказано максимальний розмір, то розмір файлу програмно не обмежується, тобто файл може збільшуватись доти, доки є вільне місце у дисковому просторі.

FILEGROWTH – одиниця збільшення файлу, вказується в Мб (за змовчуванням) або у відсотках, значення 0 забороняє збільшення файлу. Це значення використовується, якщо потрібно додати дані у базу, але «зарезервоване» під нові дані місце скінчилось (база заповнена). У цьому випадку її розмір збільшується на вказане значення (але не більше за MAXSIZE). Тобто у базу спочатку додаються порожні сторінки згідно шаблону, які вже потім заповнюються новими даними.

Значення параметрів SIZE, MAXSIZE та FILEGROWTH мають бути обов'язково цілими числами. За замовчуванням ці значення вказуються в мегабайтах, але це можна змінити за допомогою суфіксів: кілобайт (KB), мегабайт (MB), гігабайт (GB) і терабайт (TB). Значення FILEGROWTH може бути також задано у процентах, для цього використовується суфікс %.

Якщо FILEGROWTH вказано у %, то розмір приросту дорівнює зазначеному відсотку від розміру файлу на момент збільшення. У цьому випадку розмір округлюється до найближчих 64 КБ, а мінімальне значення становить 64 КБ.

2.2 Видалення бази даних за допомогою командного рядка

Для видалення БД (у нашому випадку MyFirstDB) потрібно в консолі набрати:

```
> drop MyFirstDB
```

У цей момент ніхто, навіть ви, не повинен працювати у БД, яку видаляєте. Для того щоб відв'язатись від БД, яку ви хочете видалити достатньо набрати:

```
> use master
```

2.3 Створення таблиці бази даних за допомогою командного рядка

Ви можете не тільки виконувати окремі команди за допомогою командного рядка, а й запускати сценарії в командному режимі.

Сценарії командного рядка – це файли, у яких в текстовому вигляді збережено набори команд. Тобто це такі ж команди, як і для виконання через командний рядок, але збережені у файлі. У разі запуску таких файлів усі команди будуть виконуватись послідовно, тож не забувайте писати кожен окрему команду у файлі з нового рядка. Сценарії можна створювати у будь-якому текстовому редакторі, наприклад, у «Блокноті», тільки під час збереження треба буде вказати розширення – *.sql.

Відкрийте текстовий редактор і наберіть такий скрипт для створення таблиці з ім'ям Goods в БД Sales:

```
CREATE TABLE Goods /* створення таблиці Goods */
(
  ID_Product Integer Not Null Primary Key, /* створення стовпця ID_Product
  Name VarChar(100), /* створення стовпця Name типу VarChar,
  Price Decimal(15, 2), /* створення стовпця Price типу Decimal, елементи
```

```
Count Integer, /* створення стовпця Count типу Integer */
Date Date, /* створення стовпця Date типу Date */
Note VarChar(200) /* створення стовпця Note типу VarChar,
)
```

Збережіть цей файл з назвою example.sql у теку з БД Sales (у цьому прикладі шлях до бази даних D:\DATA\). Виконайте цей сценарій з командного рядка:

```
C:\Users\1>sqlcmd -S DESKTOP-BJ9ERKQ\SQLEXPRESS -E -i D:\DATA\example.sql
Changed database context to 'Sales'.
```

У результаті буде створена таблиця Goods у БД Sales.

2.4 Видалення таблиці бази даних за допомогою командного рядка

Для перегляду інформації щодо всіх таблиць, які входять у склад БД Sales, створіть скрип такого запиту:

```
use master
select table_name from sales.information_schema.tables
Gd
```

Збережіть скрипт у файлі example1.sql та виконайте його у командному рядку:

```
C:\Users\1>sqlcmd -E -i example1.sql -o D:\output.txt -S DESKTOP-BJ9ERKQ\SQLEXPRESS
C:\Users\1>
```

Параметр -i використовується для визначення вхідних даних. Ви вказуєте файл сценарію із запитами.

Параметр -o використовується для запису результатів виконання сценарію у файл.

У результаті виконання цього скрипту у файл output.txt записується інформація про всі таблиці БД Sales (рис. 2.4).

```
Файл  Редагування  Формат  Вигляд  Довідка
Changed database context to 'Sales'.
table_name
-----
Goods

(1 rows affected)
```

Рис. 2.4 Вміст файлу output.txt – інформація про всі таблиці БД Sales

Для видалення таблиці Goods змініть текст у файлі example1.sql таким чином:

```
use sales
drop Table Goods
select table_name from sales.information_schema.tables
gc
```

Знову виконайте цей сценарій у командному рядку і перегляньте змінений файл output.txt (рис. 2.5).

```
Файл  Редагування  Формат  Вигляд  Довідка
Changed database context to 'Sales'.
table_name
-----

(0 rows affected)
```

Рис. 2.5 Вміст файлу output.txt після видалення таблиці

Як видно з файлу output.txt таблиця Goods більше не відображується серед інформації про всі таблиці БД Sales, тобто її було видалено.

2.5 Створення і видалення бази даних в середовищі SQL Server Management Studio

Для створення баз даних, крім застосування скриптів на мові SQL, поширеним є використання спеціальних програм, які мають зручні інструменти для створення та адміністрування БД. Однією з найбільш поширених таких програм є SQL Server Management Studio (SSMS), яка містить багато графічних інструментів і багатофункціональних редакторів скриптів для доступу до SQL Server.

Відкрийте SSMS, натисніть правою кнопкою миші на вузол Databases і в контекстному меню виберіть пункт New Database (рис. 2.6).

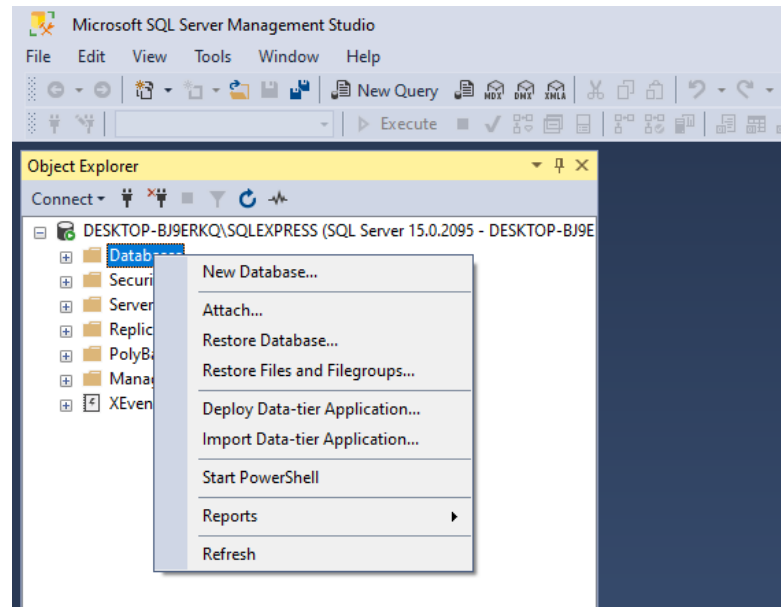


Рис. 2.6 Створення нової бази даних в SQL Server Management Studio: пункт New Database

Відкриється вікно для створення бази даних (рис. 2.7). В поле Database name необхідно ввести назву нової БД. У розглянутому прикладі нова БД буде мати назву FIT. Поле Owner задає власника бази даних. За замовчуванням воно має значення <default>, тобто власником буде той, хто створює цю базу даних.

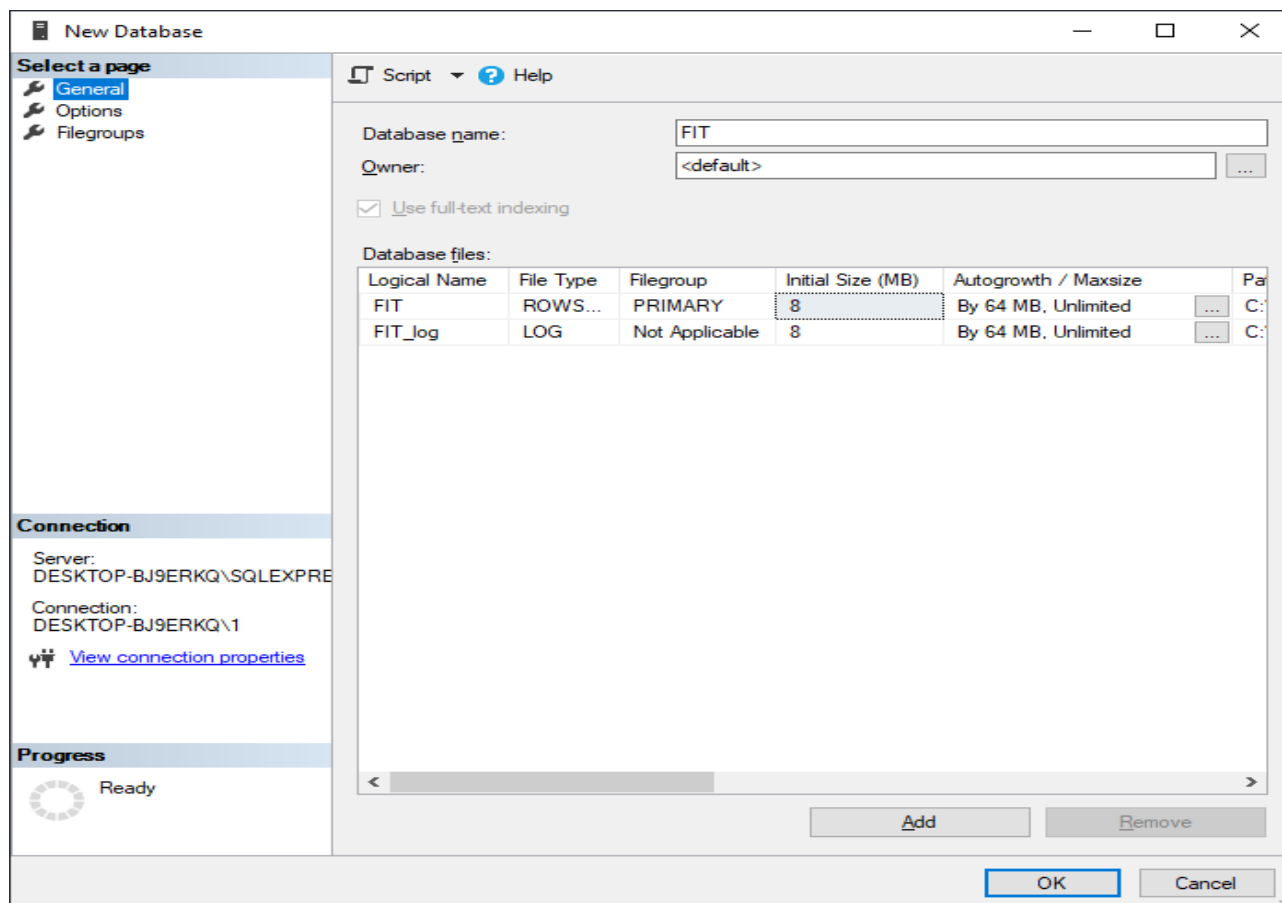


Рис. 2.7 Вікно для створення бази даних

Далі йде таблиця для установки загальних налаштувань бази даних. Вона містить два рядки: перший – для установки налаштувань головного файлу, де будуть зберігатися дані, і другий – для конфігурації файлу логування. Зокрема, можливо встановити такі налаштування:

Logical Name – логічне ім'я, яке присвоюється файлу бази даних.

File Type – є кілька типів файлів, але, як правило, основна робота ведеться з файлами даних (ROWS Data) і файлом LOG.

Filegroup – визначає групу файлів, яка може зберігати безліч файлів і може використовуватися для розбиття бази даних на частини для розміщення в різних місцях.

Initial Size (MB) – встановлює початковий розмір файлів, які створюються. Фактичний розмір може відрізнятись від цього значення.

Autogrowth/Maxsize – це значення для кроку збільшення і максимального розміру файлу (аналогічно FILEGROWTH і MAXSIZE, які описані у пункті 2.1).

Path – шлях до теки, де буде зберігатися створювана база даних.

File Name – безпосереднє ім'я фізичної файлу. Якщо воно не вказано, то застосовується логічне ім'я.

Введіть назву бази даних (FIT) і натисніть на кнопку ОК. Створена база даних FIT з'явиться серед БД сервера.

Якщо згодом ця БД стане непотрібна, то її можна видалити, натиснувши на неї правою кнопкою миші і вибравши в контекстному меню пункт Delete (рис. 2.8).

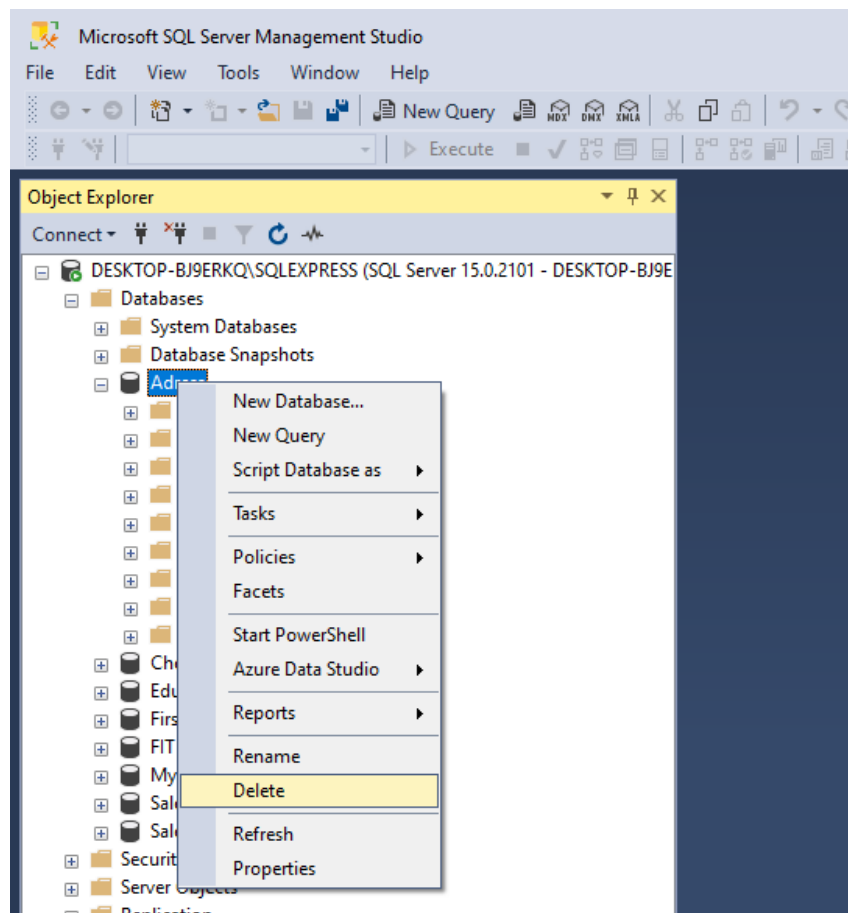


Рис. 2.8 Видалення бази даних в середовищі SQL Server Management Studio

В результаті видалення бази даних звільняється весь зарезервований під неї дисковий простір і видаляються всі об'єкти, які в ній зберігались.

Видалену базу даних можна відновити лише за допомогою резервної копії, якщо вона була попередньо створена.

2.6 Створення, заповнення і видалення таблиці бази даних в середовищі SQL Server Management Studio

Створення таблиць БД в SQL Server Management Studio можливе за допомогою скриптів на мові SQL або графічного дизайнера таблиць. Для того, щоб відкрити графічний дизайнер таблиці, у провіднику об'єктів SSMS розкрийте вузол бази даних FIT, натисніть на його підвузол Tables правою кнопкою миші і в контексті меню виберіть New -> Table (рис. 2.9).

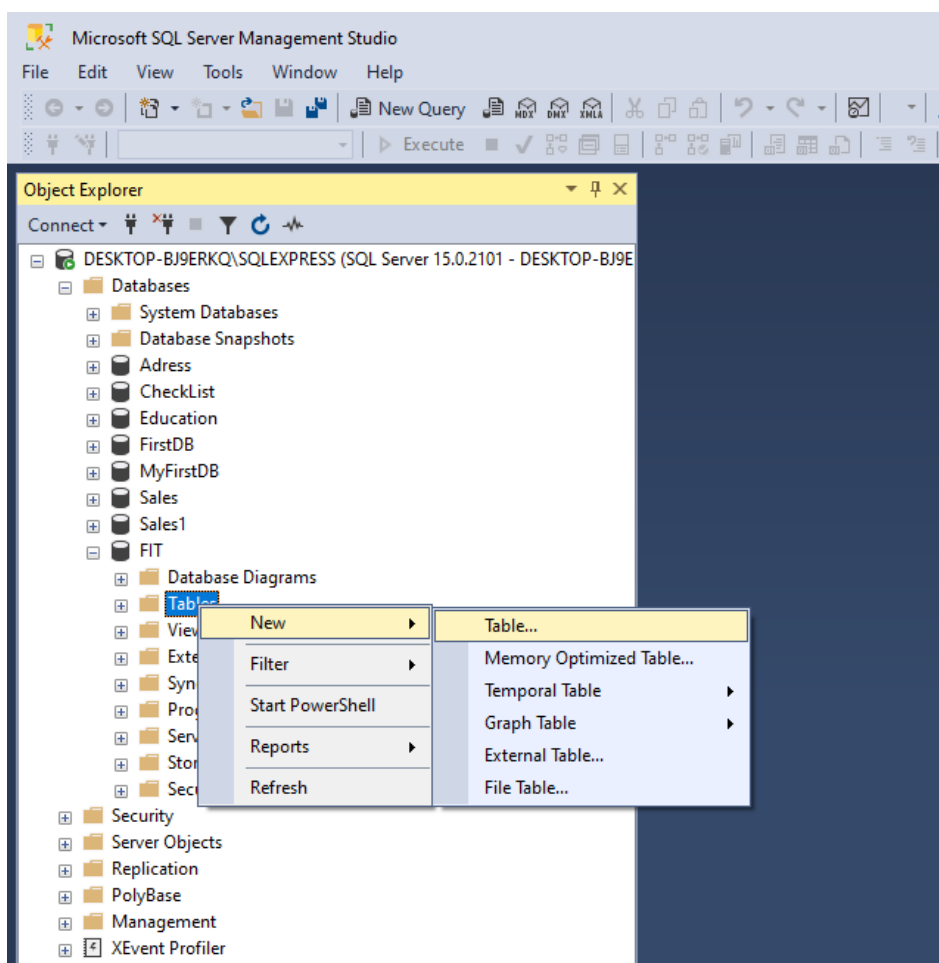


Рис. 2.9 Створення нової таблиці в SQL Server Management Studio

Після цього відкриється дизайнер таблиці, який містить три колонки:

Column Name — ім'я стовпця визначає, які будуть стовпці (поля) у створюваній таблиці.

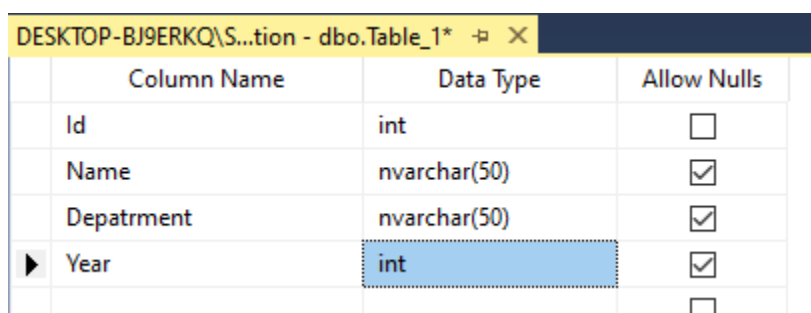
Data Type — тип даних стовпця визначає, які дані можуть зберігатися в кожному стовпці створюваної таблиці.

Allow Nulls – індикатор незаданих значень, який визначає, чи може бути відсутнім значення у стовпці під час додавання нового запису (нового рядка), тобто чи може комірка у цьому стовпці бути порожньою.

Приклад: створення таблиці з даними студентів на нашому факультеті (рис. 2.10). Для цього в дизайнері таблиці створіть чотири стовпці:

- Id – унікальний ідентифікатор студента,
- Name – ім'я студента,
- Department – кафедра,
- Year – рік навчання.

У першого і четвертого стовпця вкажіть тип `int`, а у стовпців Name, Department – тип `nvarchar(50)`.



DESKTOP-BJ9ERKQ\S...tion - dbo.Table_1* -> X			
	Column Name	Data Type	Allow Nulls
	Id	int	☐
	Name	nvarchar(50)	☒
	Department	nvarchar(50)	☒
►	Year	int	☒

Рис. 2.10 Приклад створення таблиці з даними студентів

У вікні Properties, яка містить властивості таблиці, в поле Name введіть ім'я таблиці – Students. Ім'я таблиці повинно бути унікальним в рамках бази даних. В поле Identity введіть Id. Таким чином вказується, що стовпець Id буде ідентифікатором (рис. 2.11).

Зверніть увагу, що стовпець Id виконуватиме роль первинного ключа (primary key). Первинний ключ унікально ідентифікує кожен рядок. У ролі первинного ключа може виступати один стовпець, а може і декілька.

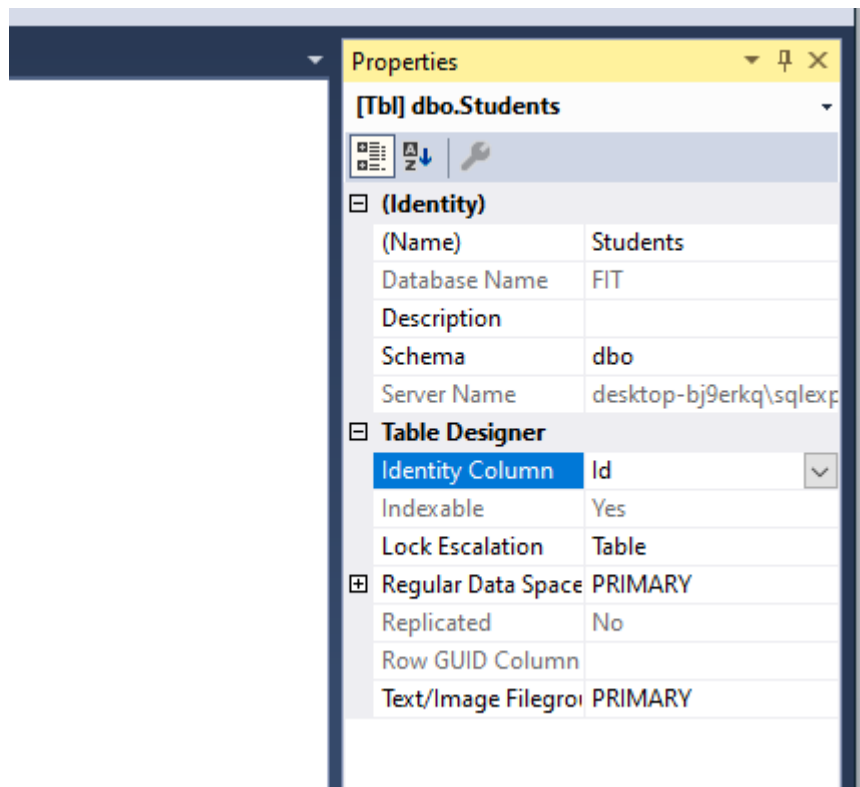


Рис. 2.11 Вікно Properties – властивості таблиці Students

Для встановлення первинного ключа натисніть правою кнопкою миші на стовпець Id і в контекстному меню виберіть пункт Set Primary Key (рис. 2.12). Після цього напроти поля Id повинен з'явитися золотий ключик. Цей ключик буде вказувати, що стовпець Id виконуватиме роль первинного ключа.

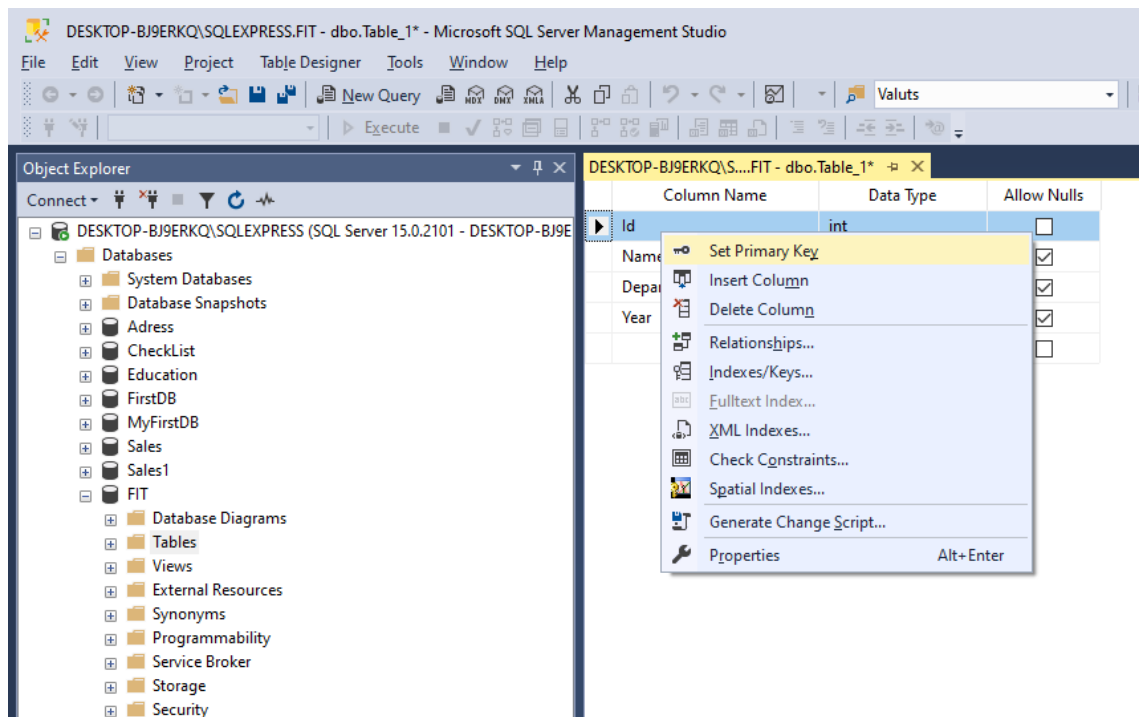


Рис. 2.12 Установлення первинного ключа стовпцю Id – Set Primary Key

Збережіть таблицю. Для цього на шапці вікна з таблицею натисніть праву кнопку миші і в контекстному меню виберіть опцію *Save Students* (рис. 2.13). Після цього у БД FIT, коли розкриємо гілку таблиці з'явиться нова таблиця з назвою *Students*. Можливо вам потрібно буде оновити БД, для цього слід на назві БД натиснути праву кнопку миші і у контекстному меню обрати опцію *Refresh*.

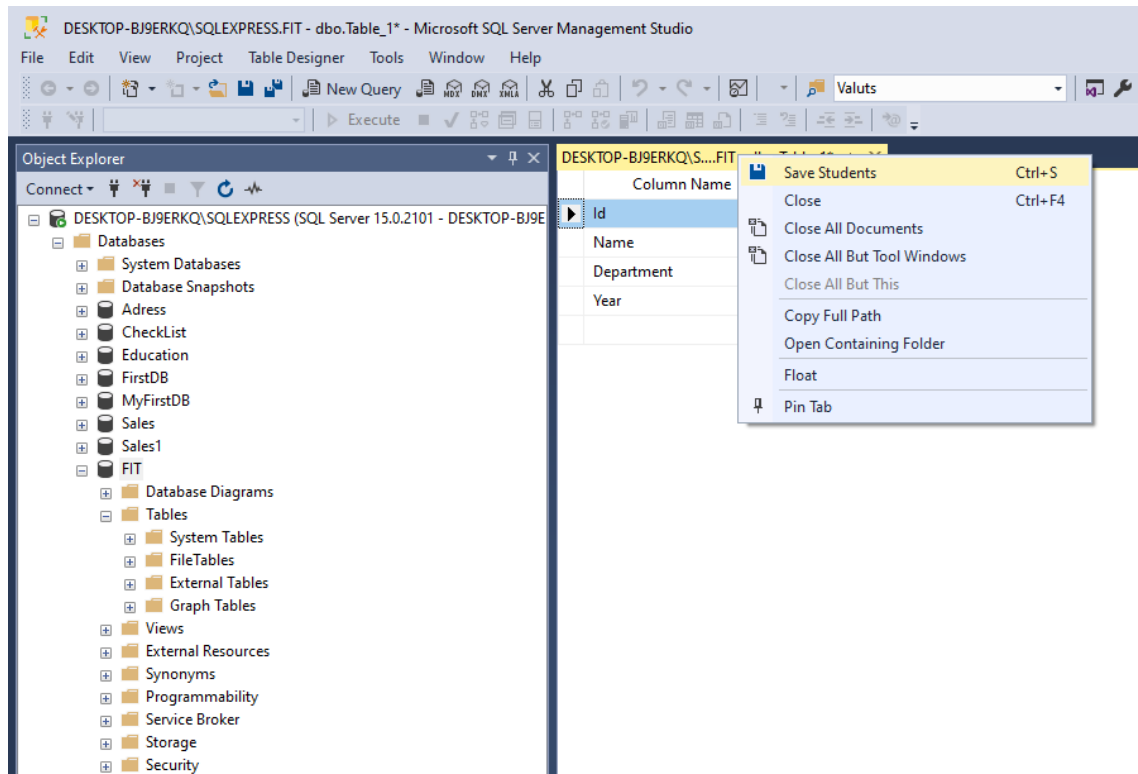


Рис. 2.13 Збереження таблиці Students бази даних FIT – Save Students

Натисніть правою кнопкою миші на назву таблиці у провіднику об'єктів SSMS. Відкриється контекстне меню з опціями, за допомогою цих опцій можна управляти таблицею (рис. 2.14). Опція *Delete* дає змогу видалити таблицю, опція *Rename* – змінити її ім'я. Опція *Design* відкриє вікно дизайнера таблиці, де можливо за необхідності вносити зміни в її структуру. Для додавання початкових даних виберіть опцію *Edit Top 200 Rows*.

Відкриється вікно з таблицею, яке дає змогу додавати і змінювати записи – рядки таблиці. Зазвичай відкривається 200 перших рядків, але, оскільки у нас таблиця тільки створена, то в ній не буде ніяких даних.

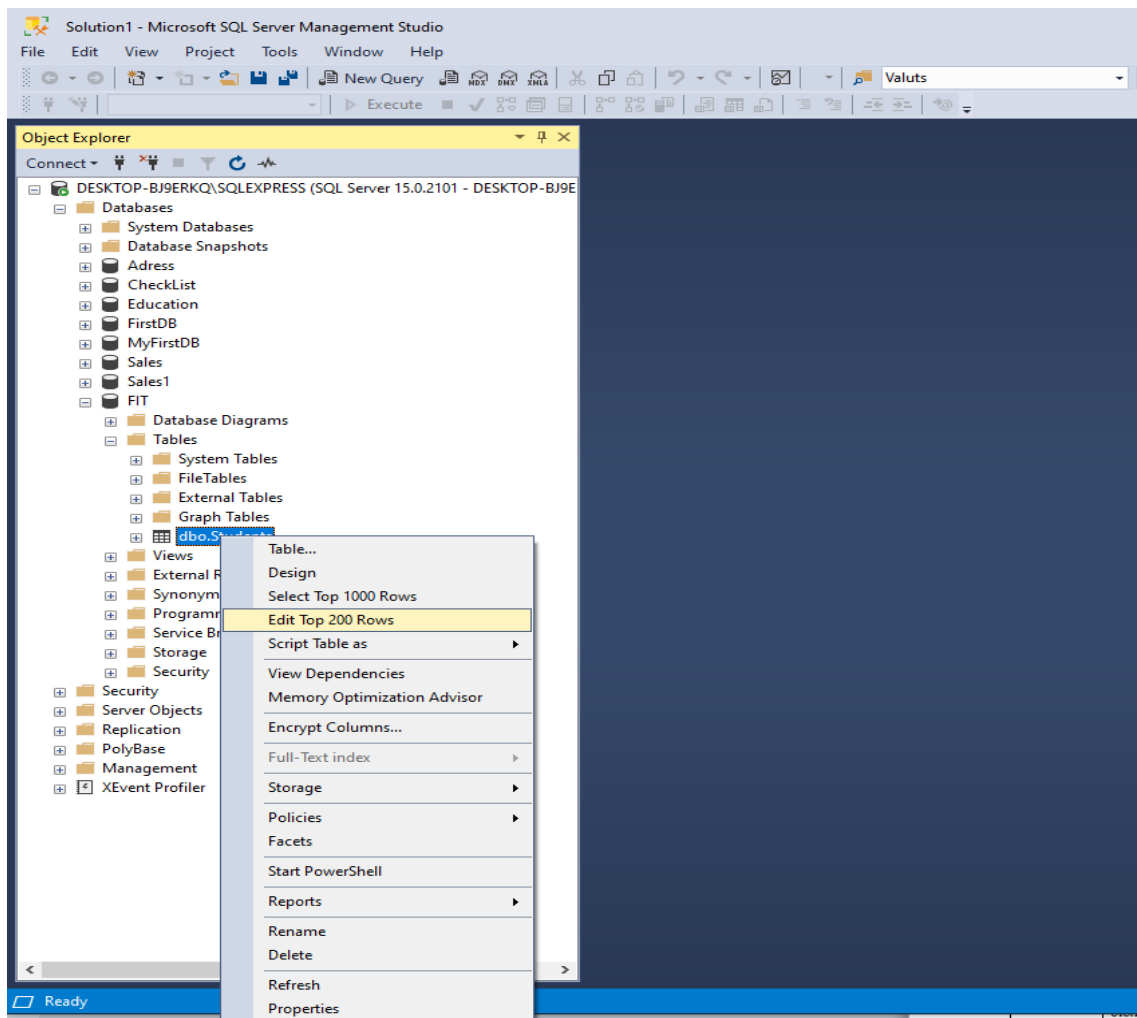


Рис. 2.14 Контекстне меню з опціями для управління таблицею. Опція Edit Top 200 Rows

Тепер заповніть даними створену таблицю. Додайте пару рядків (пару записів), вписавши необхідні дані про студентів у комірки відповідних стовпців (рис. 2.15).

DESKTOP-BJ9ERKQ\S...FIT - dbo.Students				
	Id	Name	Department	Year
	1	Ільченко Віктор	ІПЗ	1
»*	NULL	NULL	NULL	NULL

Рис. 2.15 Заповнення таблиці Students даними про студентів

Одна стандартна таблиця може містити до 1024 стовпців, а кількість записів (рядків) у таблиці обмежена лише обсягом виділеної для неї пам'яті або вільним дисковим простором на сервері, якщо Maxsize не задано.

ЗАВДАННЯ

Варіант завдання вибирається за номером студента в журналі групи. Варіантів у всіх завданнях по 10, якщо номер у журналі більший, виконуєте «по колу». Тобто 11 номер – 1 варіант, 12 – 2 варіант тощо.

Варіант	Опис завдання
1	Створити БД і дві таблиці за допомогою графічного дизайнера SQL Server Management Studio. Заповнити по 3 рядки в першій таблиці і 5 рядків у другій. БД – <i>успішність студентів ВНЗ</i> – складається з таких таблиць: факультети, кафедри. Таблиця факультети має такі атрибути: назва факультету, ПІБ декана, номер корпусу, номер телефону. Таблиця кафедра має такі атрибути: назва кафедри, факультет, ПІБ завідувача.
2	Створити БД і дві таблиці за допомогою командного рядка SQL. Заповнити по 3 рядки в першій таблиці і 5 рядків у другій в середовищі SQL Server Management Studio. БД – <i>інформаційна система супермаркету</i> – складається з таких таблиць: відділи, співробітники. Таблиця відділи має такі атрибути: назва відділу, кількість прилавків, кількість продавців, номер залу. Таблиця співробітники має такі атрибути: прізвище, ім'я, по батькові, відділ, рік народження, рік влаштування на роботу, стаж, посада, стать, адреса, місто, номер телефону.
3	Створити БД за допомогою графічного дизайнера SQL Server Management Studio. Створити першу таблицю в SQL Server Management Studio. Другу таблицю у командному рядку. Заповнити по 3 рядки у першій таблиці і 5 рядків у другій.

	БД – <i>інформаційна система військового округу</i> – складається з таких таблиць: місця дислокації, вид військ. Таблиця вид військ має такі атрибути: назва. Таблиця місця дислокації має такі атрибути: країна, місто, адреса, яку займає площу.
4	Створити БД за допомогою командного рядка SQL Server. Таблиці створити в SQL Server Management Studio.. Заповнити по 3 рядки у першій таблиці і 3 рядки у другій. БД – <i>інформаційна система бібліотеки</i> – складається з таких таблиць: бібліотеки, фонд бібліотеки. Таблиця бібліотеки має такі атрибути: назва, адреса, місто. Таблиця фонд бібліотеки має такі атрибути: назва фонду, бібліотека, кількість книг, кількість журналів, кількість газет, кількість збірок, кількість дисертацій, кількість рефератів.
5	Створити БД за допомогою графічного дизайнера SQL Server Management Studio. Таблиці створити за допомогою командного рядка. Заповнити по 5 рядків у першій таблиці і 3 рядки у другій. БД – <i>інформаційна система туристичного агентства</i> – складається з таких таблиць: пансіонати, вид житла. Таблиця пансіонати має такі атрибути: назва пансіонату, адреса, місто, країна, номер телефону, опис території, кількість кімнат, наявність басейну, наявність медичних послуг, наявність спа-салону, рівень пансіонату, відстань до моря. Таблиця вид житла має такі атрибути: назва (будинок, бунгало, квартира, 1-я кімната, 2-я кімната тощо), категорія житла (люкс, напівлюкс, тощо), пансіонат, опис умов проживання , ціна за номер на добу.
6	Створити БД і дві таблиці за допомогою графічного дизайнера SQL Server Management Studio. Заповнити по 3 рядки в

	першій таблиці і 5 рядків у другій. БД – <i>інформаційна система автотранспорту міста</i> – складається з таких таблиць: автотранспорт, маршрути. Таблиця автотранспорт має такі атрибути: назва транспорту (автобуси, таксі, маршрутні таксі, інший легковий транспорт, вантажний транспорт тощо), рік випуску, пробіг, кількість ремонтів, характеристика. Таблиця маршрути має такі атрибути: назва маршруту, транспорт, водій, графік роботи.
7	Створити БД і дві таблиці за допомогою командного рядка SQL. Заповнити по 3 рядки в першій таблиці і 5 рядків у другій в середовищі SQL Server Management Studio. БД – <i>інформаційна система поліклініки</i> – складається з таких таблиць: лікарі, відділення. Таблиця відділення має такі атрибути: назва відділення (хірургія, терапія, неврологія тощо), номер відділення, поверх, номери кімнат, ПІБ завідувача. Таблиця лікарі має такі атрибути: прізвище, ім'я, по батькові, стаж роботи, спеціальність (терапевт, хірург, кардіолог тощо), посада (лікар, головний лікар, завідувач відділу, заступник завідувача відділу тощо), номер відділення, в якому він працює.
8	Створити БД за допомогою графічного дизайнера SQL Server Management Studio. Створити першу таблицю в SQL Server Management Studio. Другу таблицю у командному рядку. Заповнити по 3 рядки у першій таблиці і 5 рядків у другій. БД – <i>інформаційна система лікарні</i> – складається з таких таблиць: лікарі, пацієнти. Таблиця лікарі має такі атрибути: ID, прізвище, ім'я, по батькові, спеціальність (сімейний лікар, терапевт, хірург, кардіолог тощо), звання (немає, лікар-спеціаліст, заслужений

	лікар), назва лікарні та адреса лікарні, в якій він працює. Таблиця пацієнти має такі атрибути: прізвище, ім'я, по батькові, адреса проживання, місто, вік, стать, ID сімейного лікаря або терапевта, з яким у цього пацієнта підписана угода.
9	Створити БД за допомогою командного рядка SQL Server. Таблиці створити в SQL Server Management Studio. Заповнити по 3 рядки у першій таблиці і 3 рядки у другій. БД – <i>інформаційна система бібліотек міста</i> – складається з таких таблиць: бібліотеки, читальні зали. Таблиця бібліотеки має такі атрибути: назва, адреса, місто. Таблиця читальні зали має такі атрибути: назва читального залу, бібліотека, кількість одиниць літератури, кількість посадкових місць, час роботи, поверх, кількість співробітників.
10	Створити БД за допомогою графічного дизайнера SQL Server Management Studio. Таблиці створити за допомогою командного рядка. Заповнити по 5 рядків у першій таблиці і 3 рядки у другій. БД – <i>інформаційна система автосалону</i> – складається з таких таблиць: автомобілі, марка автомобіля. Таблиця марка автомобіля має такі атрибути: назва марки, країна виробник, завод виробник, адреса. Таблиця автомобілі має такі атрибути: назва автомобіля, марка, рік виробництва, колір, категорія, ціна.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Які найпоширеніші типи даних в SQL?
2. Яким чином можуть бути оголошеними цілі числа в SQL?
3. Яким чином можуть бути оголошеними десяткові числа в SQL?

4. Яким чином можуть бути оголошеними числа з плаваючою крапкою в SQL?
5. Яким чином можуть бути оголошеними рядки символів сталої довжини в SQL?
6. Яким чином можуть бути оголошеними рядки символів змінної довжини в SQL?
7. Яким чином можуть бути оголошеними грошові величини в SQL?
8. Яким чином можуть бути оголошеними дата та час в SQL?
9. Яким чином можуть бути оголошеними булеві величини в SQL?
10. Яким чином можуть бути оголошеними довгий текст в SQL?
11. Яким чином можуть бути оголошеними неструктуровані потоки байтів в SQL?
12. Яким чином можуть бути оголошеними нелатинські символи в SQL?
13. Чи є мова SQL строго типізованою?
14. Що таке первинний ключ?

Лабораторна робота №3

Тема: Робота з Azure Data Studio

План

1. Azure Data Studio;
2. Додавання даних у таблицю;
3. Завдання;
4. Питання для самоперевірки.

3.1 Azure Data Studio

Azure Data Studio — це середовище для роботи з даними, що використовує сукупність локальних і хмарних платформ даних. Після запуску Azure Data Studio відкриється стартове вікно для підключення до бази даних (рис. 3.1).

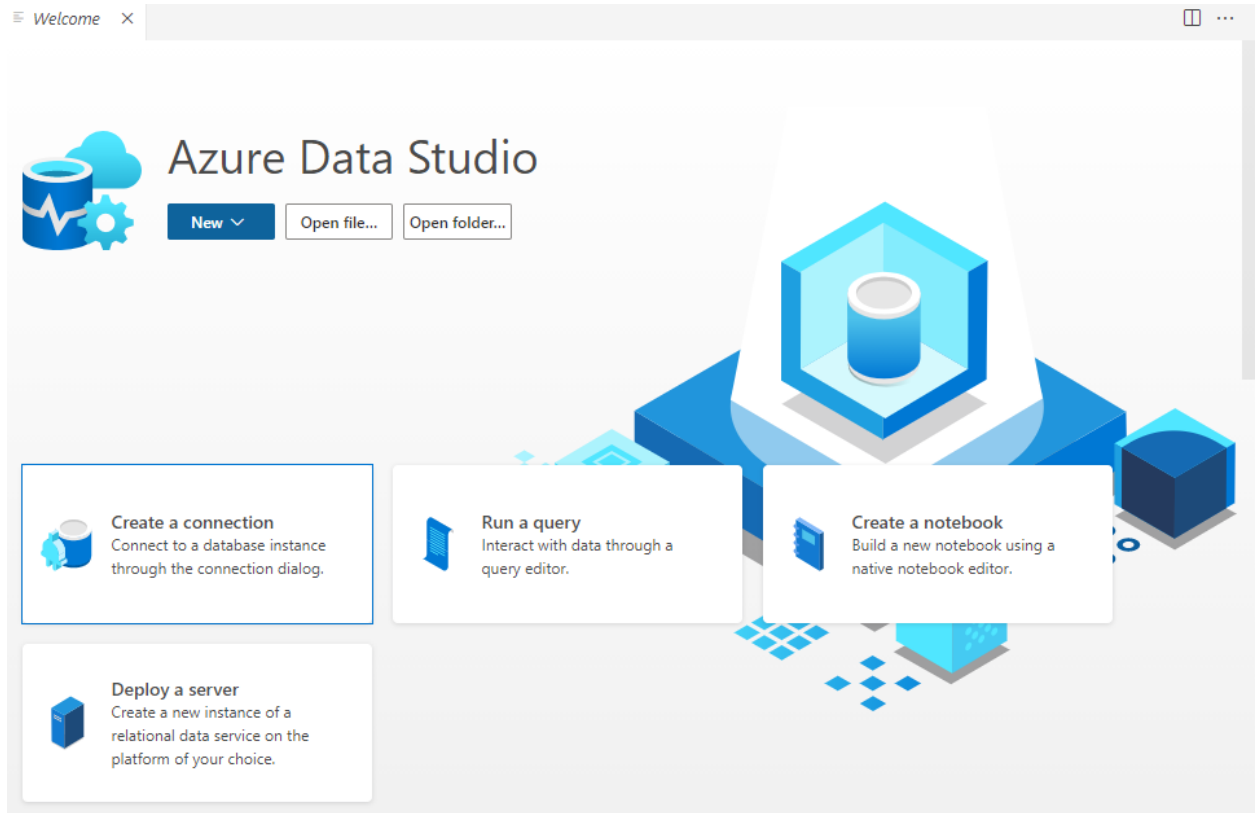


Рис. 3.1 Стартове вікно Azure Data Studio

Перший крок, який потрібно зробити для роботи з даними в Azure Data Studio – це створити з'єднання із сервером. Натисніть на кнопку **Create a connection**, щоб відкрити вікно для налаштування параметрів підключення (рис. 3.2).

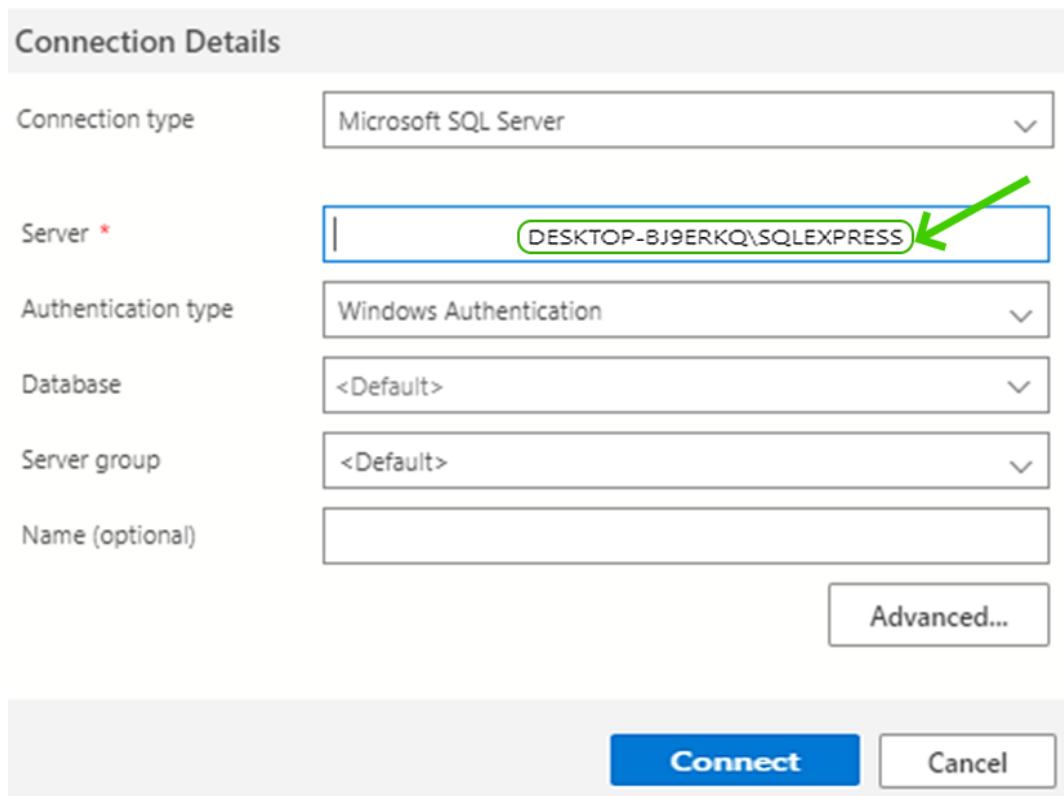


Рис. 3.2 Вікно налаштування параметрів підключення до сервера

Параметри підключення:

- Тип підключення – SQL Server Management Studio;
- Ім'я сервера – введіть ім'я комп'ютера, на якому запущено SQL Server. У цьому прикладі – DESKTOP-BJ9ERKQ\SQLEXPRESS – це локальний хост, оскільки підключення йде до локальної БД;
- Тип автентифікації – рекомендовано «Автентифікація Windows», яка дає змогу використовувати свої облікові дані Windows для автентифікації на SQL Server;
- Ім'я бази даних – <за замовчуванням>;
- Група серверів – <за замовчуванням>;
- Ім'я (за потреби) – цей параметр дає змогу вказати зрозуміле ім'я для вашого сервера.

Якщо натиснути кнопку Advanced, то вводиться така інформація:

- Тип перевірки довжини: Ім'я входу SQL;
- Ім'я користувача: ім'я користувача SQL Server;
- Пароль: пароль для SQL Server;

Введіть тільки ім'я свого комп'ютера, на якому запущено SQL Server, інші параметри лишити за замовчуванням і натисніть «**Connect**», щоб підключитися до SQL Server. У Azure Data Studio відкриється нове вікно, вкладення «Home» (рис. 3.3).

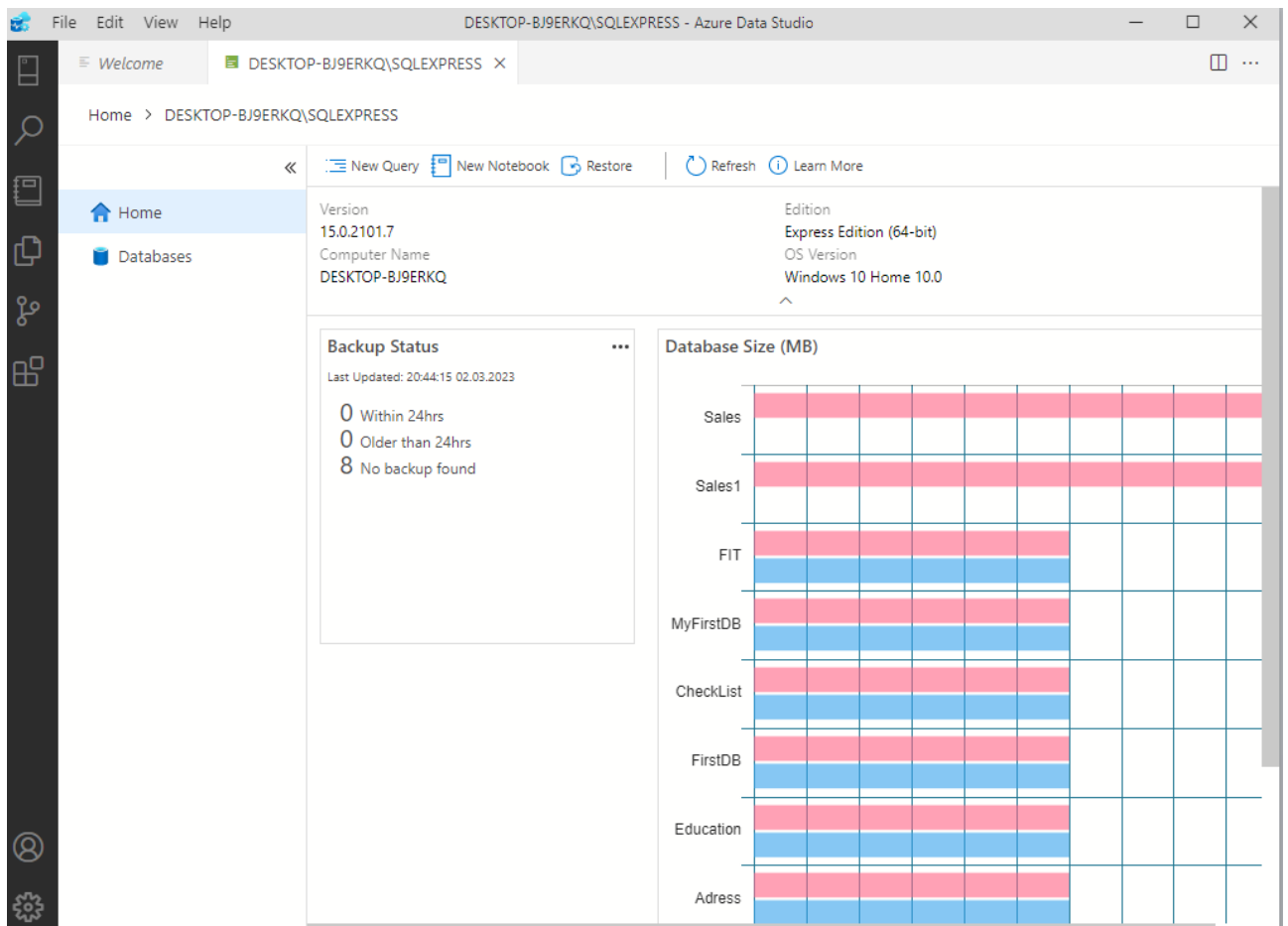


Рис. 3.3 Вкладення «Home» в Azure Data Studio після підключення до сервера

Перейдіть на вкладення «Database», щоб побачити всі БД на нашому сервері (рис. 3.4).

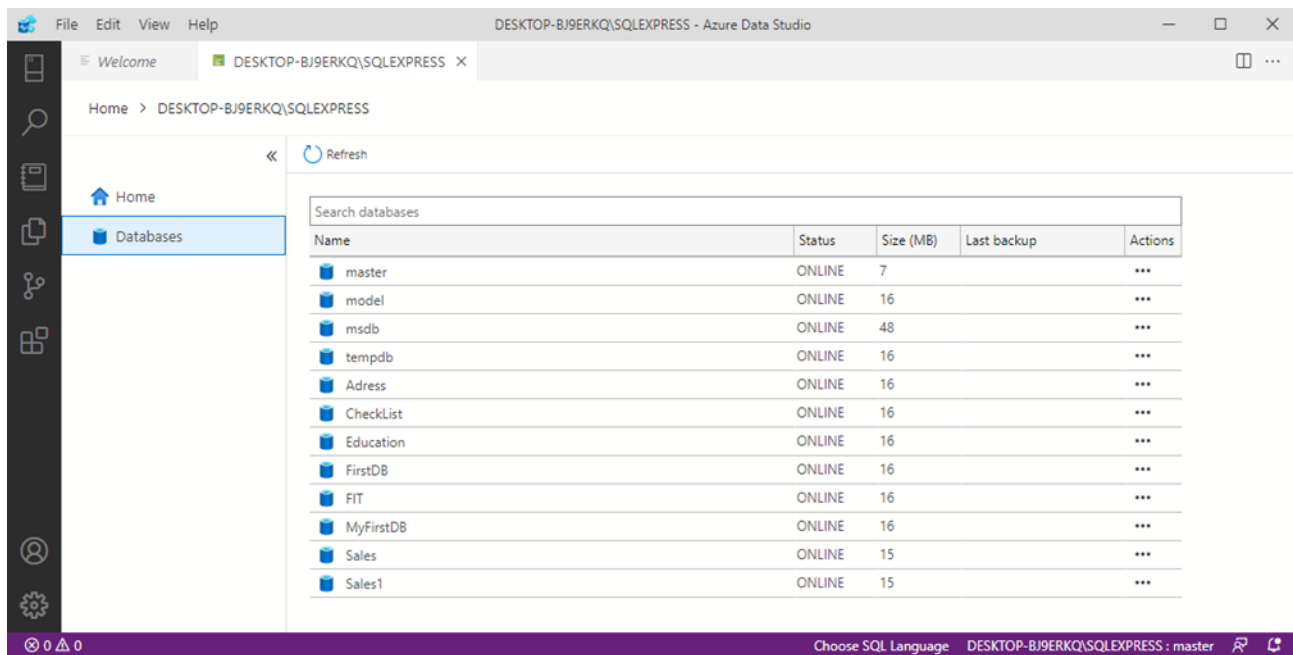


Рис. 3.4 Вкладення Database в Azure Data Studio

Щоб переглянути таблиці, які є у БД, достатньо натиснути лівою кнопкою миші на назві потрібної БД. На рис. 3.5 показано таблиці бази даних FIT.

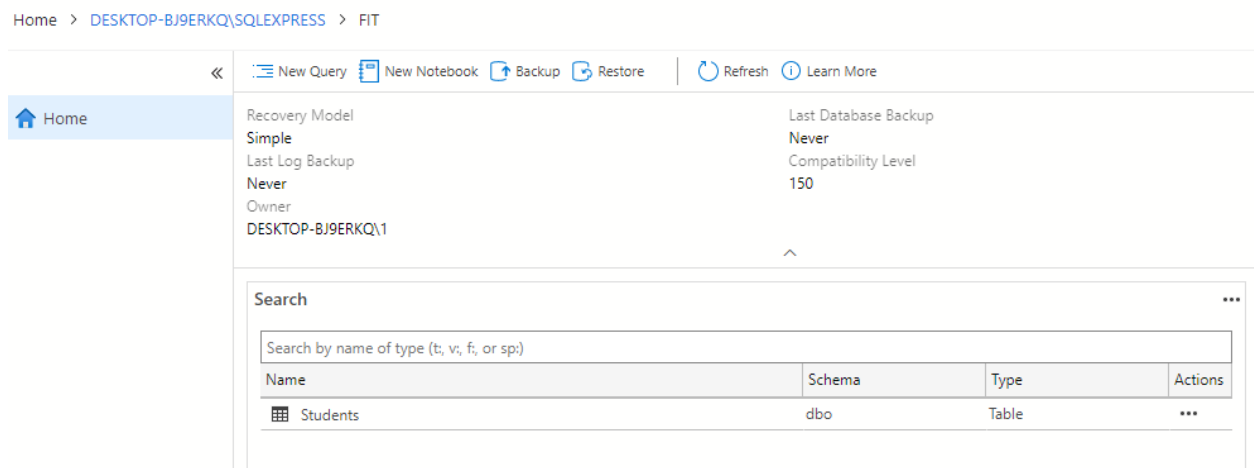


Рис. 3.5 Таблиці бази даних FIT

Якщо потрібно повернутися назад, тисніть на кнопку «», яка розташована зліва вгорі (над «Home»).

3.2 Додавання даних в таблицю

На цьому етапі у базі даних FIT лише одна таблиця – Students. Для того, щоб додати дані в цю таблицю, натисніть кнопку **New Query** (рис. 3.6)

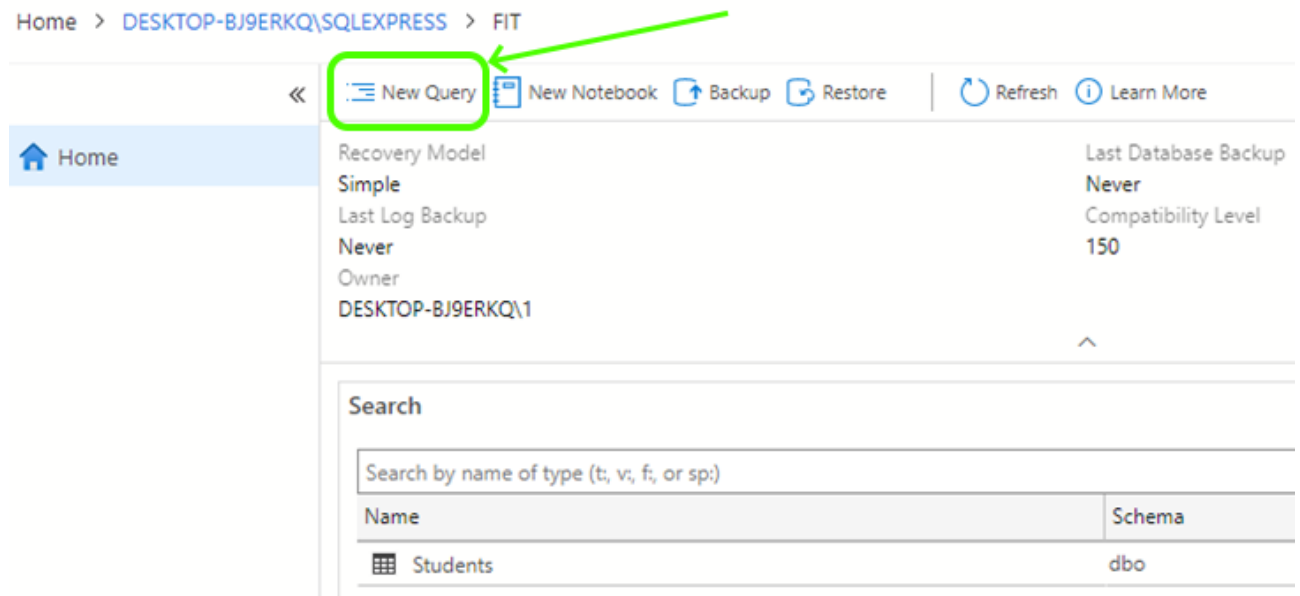


Рис. 3.6 Кнопка для створення нового запиту

Відкриється нова закладка для створення запиту. Наберіть такий код, як на рис. 3.7 і натисніть кнопку **Run**.

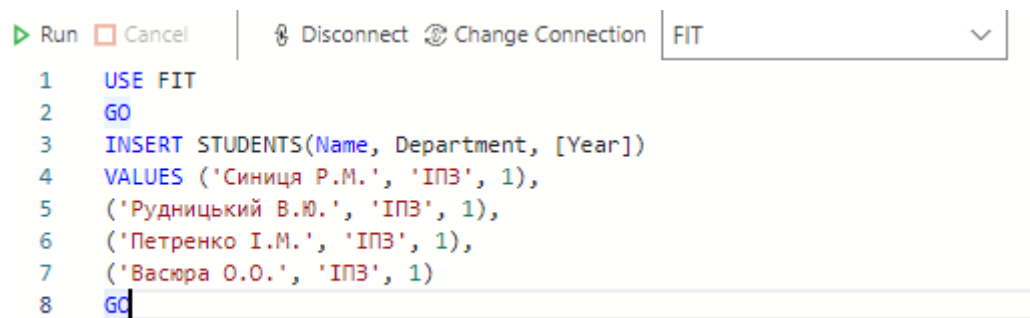


Рис. 3.7 Запит на додавання нових записів до таблиці Students

Щоб переглянути, що зараз зберігається у таблиці, перейдіть на закладку з таблицями. Біля таблиці Students у колонці Actions натисніть на три крапки – відкриється контексте меню (рис. 3.8).

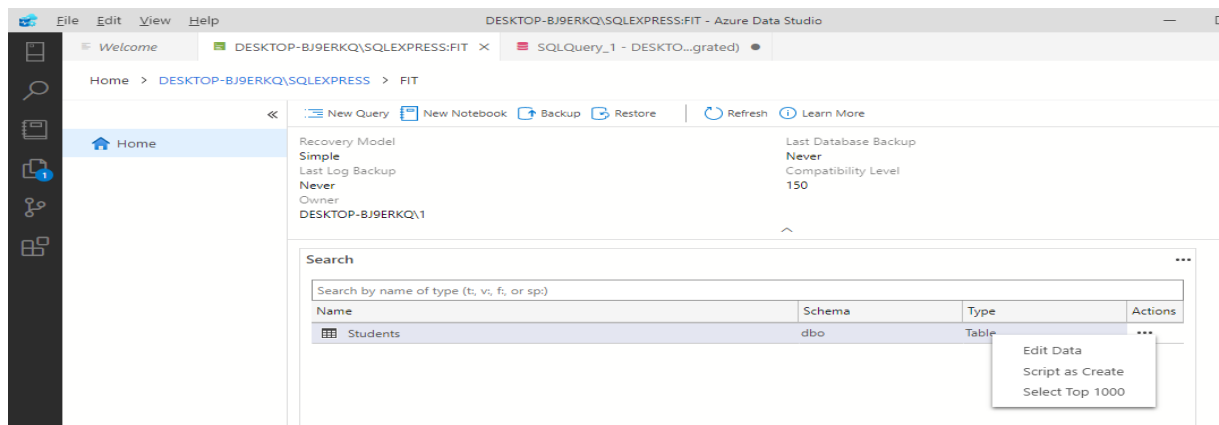


Рис. 3.8 Контекстне меню Actions

Виберіть пункт Select Top 1000 – відобразиться зміст таблиці (Рис. 3.9).

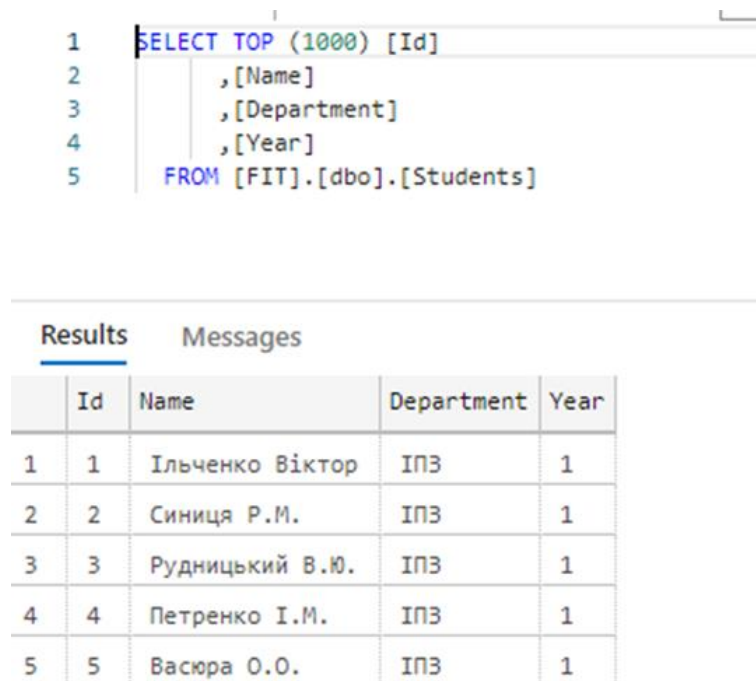


Рис. 3.9 Результат перегляду таблиці засобами Azure Data Studio

Можете переконатися, що записи, додані за допомогою запиту (рис. 3.7), дійсно додалися в кінець таблиці Students.

ЗАВДАННЯ

Варіант	Опис завдання
1	1. За допомогою запиту додати у другу таблицю 5 записів, запит

	виконати у командному рядку. 2. У командному рядку вивести БД за замовчуванням.
2	1. За допомогою запиту додати у другу таблицю 5 записів, запит виконати в Azure Data Studio. 2. У командному рядку вивести список БД, не використовуючи збережену процедуру.
3	1. За допомогою запиту додати у другу таблицю 5 записів, запит виконати у командному рядку. 2. Створити таблиці частини і роти. Таблиця частини має поля: номер частини, вид військ, місце дислокації. Таблиця роти має поля: номер роти, номер частини.
4	1. За допомогою запиту додати у першу таблицю 2 записи. Запит виконати у командному рядку. У другу таблицю додати 2 записи, запит виконати в Azure Data Studio. 2. Створити таблицю Співробітники з полями: ПІБ, оклад, бібліотека, телефон
5	1. За допомогою запиту додати у першу таблицю 5 записів, запит виконати в Azure Data Studio. 2. У командному рядку перевірити випуск MS SQL Server.
6	1. За допомогою запиту додати у другу таблицю 5 записів, запит виконати у командному рядку. 2. У командному рядку перевірити автентифікацію SQL Server. Пояснити результат.
7	1. За допомогою запиту додати у другу таблицю 5 записів, запит виконати в Azure Data Studio. 2. У командному рядку перерахувати змінні середовища. Пояснити результат.
8	1. За допомогою запиту додати у другу таблицю 5 записів, запит виконати у командному рядку. 2. У командному рядку перевірити середовище на чутливість до

	регістру. Пояснити результат.
9	1. За допомогою запиту додати у першу таблицю 2 записи. Запит виконати у командному рядку. У другу таблицю додати 2 записи, запит виконати в Azure Data Studio. 2. У командному рядку вивести список БД, не використовуючи збережену процедуру.
10	1. За допомогою запиту додати у першу таблицю 5 записів, запит виконати в Azure Data Studio. 2. У командному рядку перерахувати змінні середовища. Пояснити результат.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Дайте визначення понять база даних і таблиця?
2. Назвіть основні типи даних, які використовуються в Transact-SQL?
3. Чим відрізняються рядкові типи NChar і NVarChar?
4. У чому полягає особливість використання типу Bit?
5. Проведіть аналіз числових типів з фіксованою і плаваючою комами.
6. Яка команда використовується для видалення бази даних?
7. Які команди використовуються для створення бази даних і таблиць?
9. Що означає параметр not null в описі поля таблиці, що створюється?
10. Навіщо потрібен первинний ключ (Primary Key)?
13. Яким чином в мові SQL відбувається додавання даних в таблицю?

Лабораторна робота №4

Тема: Вибірки даних, редагування і видалення даних

План

1. Вибірки даних;
2. Сортування. ORDER BY;
3. TOP – вибірка з вказаної кількості перших рядків таблиці;
4. OFFSET і FETCH – діапазон вибірки;
5. WHERE – фільтрація;
6. Логічні оператори;
7. IS NULL;
8. Оператори фільтрації;
9. Видалення даних. Команда DELETE;
10. Оновлення даних. Команда UPDATE;
11. Завдання;
12. Питання для самоперевірки.

4.1 Вибірки даних

Оператор **SELECT** призначений для вибірки даних з бази даних в SQL.

Цей оператор відноситься до підмножини DML і має такий синтаксис:

```
SELECT [DISTINCT | ALL]
{* | <величина> [, <величина> ...]}
[INTO: Змінна [,: Змінна ...]]
FROM <tableref> [, <tableref> ...]
[WHERE <умова пошуку>]
[GROUP BY Колонка [, Колонка ...]]
[HAVING <умова пошуку>]
[UNION [ALL] <select_expr>]
[ORDER BY <список сортування>];
```

<Величина> = {Колонка | : Змінна | <Константа>
| <Вираз> | <Функція>
| udf ([<величина> [, <величина> ...]])
| **NULL** | **USER**} [**AS Псевдонім**]

<Константа> = Число | 'Рядок'

<Вираз> = SQL вираз, що повертає одиничне значення

<Функція> =

COUNT (* | [**ALL**] <величина> | **DISTINCT** <величина>)
| **SUM** ([**ALL**] <величина> | **DISTINCT** <величина>)
| **AVG** ([**ALL**] <величина> | **DISTINCT** <величина>)
| **MAX** ([**ALL**] <величина> | **DISTINCT** <величина>)
| **MIN** ([**ALL**] <величина> | **DISTINCT** <величина>)
| **CAST** (<величина> **AS** <тип даних>)
| **UPPER** (<величина>)
| **GEN_ID** (Імя_Генератора, <величина>)

<Tableref> = {<joined_table> | table | view
| procedure [(<величина> [, <величина> ...])]}
[Псевдонім]

<Joined_table> = <tableref> <join_type> **JOIN** <tableref>
ON <умова пошуку> | (<Joined_table>)

<Join_type> = [**INNER**] | {**LEFT** | **RIGHT** | **FULL**} [**OUTER**]

<Умова пошуку> =

<Величина> <оператор порівняння>
{<Величина> | (<Select_one>)}
| <Величина> [**NOT**] **BETWEEN** <величина> **AND** <величина>
| <Величина> [**NOT**] **LIKE** <величина>
| <Величина> [**NOT**] **IN**

(<Величина> [, <величина> ...] | <select_list>)
 | <Величина> **IS [NOT] NULL**
 | <Величина> {> = | <=} <Величина>
 | <Величина> [**NOT**] {= | <| >} <Величина>
 | {**ALL** | **SOME** | **ANY**} (<select_list>)
 | **EXISTS** (<select_expr>)
 | **SINGULAR** (<select_expr>)
 | <Величина> [**NOT**] **CONTAINING** <величина>
 | <Величина> [**NOT**] **STARTING [WITH]** <величина>
 | (<Умова пошуку>)
 | **NOT** <умова пошуку>
 | <Умова пошуку> **OR** <умова пошуку>
 | <Умова пошуку> **AND** <умова пошуку>

<Оператор порівняння> =

{= | <| > | <= | > = | ! <| !> | <> | !=}

<Select_one> = оператор SELECT, що вибирає одну колонку і повертає лише одне значення

<Select_list> = оператор SELECT, що вибирає одну колонку, який повертає нуль або багато значень

<Select_expr> = оператор SELECT, що вибирає кілька величин і повертає нуль або багато значень

<Список сортування> =

{Колонка | Номер} [**ASC** | **DESC**]

[, <Список сортування> ...]

Деякі параметри, що входять в оператор SELECT, описані в табл. 4.1.

Опис параметрів оператора SELECT

Параметр	Опис
DISTINCT ALL	DISTINCT – запобігає дублюванню даних, які будуть вилучені. ALL (за замовчуванням) - призведе до вилучення всіх
{* <величина> [, <величина> ...]}	Зірочка (*) означає, що треба вивести всі колонки із зазначених таблиць. <Величина> [, <величина> ...] - виводить список зазначених величин
INTO :змінна [, : змінна...]	Використовується тільки в тригерах і збережених процедурах для операторів SELECT , які повертають не більше одного рядка. Вказується список змінних, в які виводяться величини
FROM <tableref> [, <tableref> ...]	Вказує список таблиць, переглядів і збережених процедур, з яких беруться дані. Список може включати з'єднання
table	Ім'я існуючої в базі даних таблиці
view	Ім'я існуючого бази даних перегляду
procedure	Ім'я існуючої збереженої процедури, призначеної для використання в операторі SELECT
Псевдоним	Коротке альтернативне ім'я для таблиці, перегляду або колонки. Після опису в <tableref>, псевдонім може використовуватися для посилань на таблицю або перегляд
join_type	Задає тип з'єднання, яке може бути внутрішнім або зовнішнім.
WHERE <умова_пошуку>	Визначає умову, яка обмежує кількість отриманих рядків
GROUP BY Колонка [,	Розбиває результат запиту на групи, що включають всі рядки з ідентичними значеннями, зазначеними в списку

Параметр	Опис
Колонка ...]	
HAVING <умова_пошуку>	Використовується спільно з GROUP BY. Задає умову, яка обмежує кількість груп, що виводяться.
UNION [ALL]	Об'єднує результати декількох запитів. Всі запити мають отримувати однакову кількість стовпців, тип даних кожного стовпця першого запиту повинен збігатися з типом даних інших запитів, імена стовпців в різних запитах можуть відрізнятися. Необов'язковий параметр ALL
ORDER BY <список сортування >	Вказує колонки, за якими буде проводитися сортування вказаних рядків. Можна вказувати або імена колонок, або їх порядкові номери у списку видобутих колонок. Якщо вказати ASC , то рядки будуть видаватися в порядку зростання значень полів, що відсортовані, якщо DESC – в порядку спадання.

Приклад 4.1.

Нехай маємо БД *check* яка складається з таких таблиць (рис. 4.2):

1. *Фізичні особи – Person*
2. *Контрагенти – Kontragent*
3. *Співробітники – Employee*

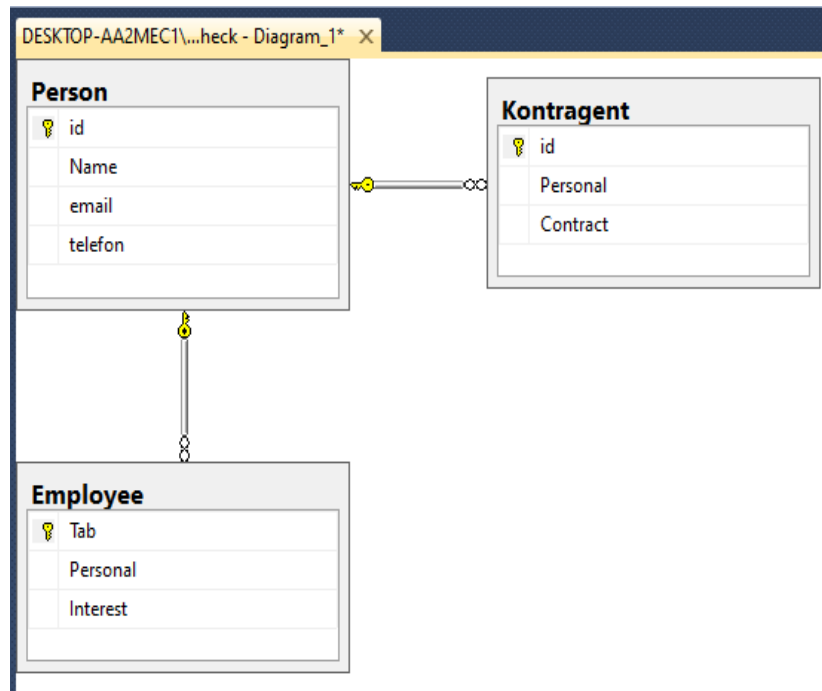


Рис. 4.1 Діаграма бази даних *check*

Додаємо нові дані у таблицю Person за допомогою запиту:

```

USE [check]
GO

INSERT Person (Name, email, telefon)
VALUES ('Іванов І.П.', 'abs.gmail.com', '0675144145'),
      ('Петров О.О.', 'qwerty.gmail.com', '0506137478'),
      ('Кравченко К.К.', 'travnik.gmail.com', '0506137478'),
      ('Мартинюк С.Л.', 'rav.ukr.net', '0735137575'),
      ('Синиця О.В.', 'nirvana.gmail.com', '0666337338'),
      ('Висоцький М.В.', 'ratusha.ukr.net', '0962141566');
GO
  
```

В результаті виконання цього запиту у кінці таблиці Person з'явилося 6 нових записів з даними фізичних осіб (рис. 4.2).

	id	Name	email	telefon
	1	Іванов Іван	ivanov@ukr.net	0671947186
	2	Петров Петро	petrov@ukr.net	0733211466
	3	Вагнер Дмитро	NULL	NULL
	4	Ільченко Ната...	NULL	NULL
	10011	Іванов І.П.	abs.gmail.com	0675144145
►	10012	Петров О.О.	qwerty.gmail.c...	0506137478
	10013	Кравченко К.К.	travnik.gmail.c...	0506137478
	10014	Мартинюк С.Л.	rav.ukr.net	0735137575
	10015	Синиця О.В.	nirvana.gmail.c...	0666337338
	10016	Висоцький М.В.	ratusha.ukr.net	0962141566
*	NULL	NULL	NULL	NULL

Рис. 4.2 Таблиця Person

Для отримання певної вибірки даних застосовується команда SELECT. У спрощеному вигляді вона має такий синтаксис:

```
SELECT список_стовпців FROM ім'я_таблиці
```

Застосуємо команду SELECT для отримання імен фізичних осіб з таблиці Person (рис. 4.3).

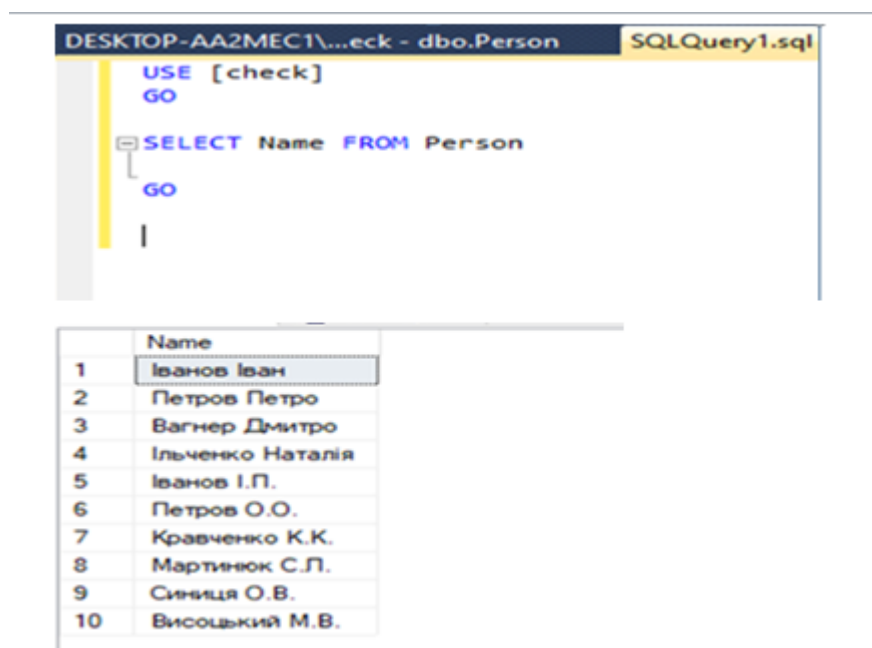


Рис. 4.3 Вибірка усіх імен фізичних осіб з таблиці Person

Приклад 4.2.

Застосуємо команду SELECT для отримання ПІБ осіб та їх електронних скриньок в одному стовпці (рис. 4.4).

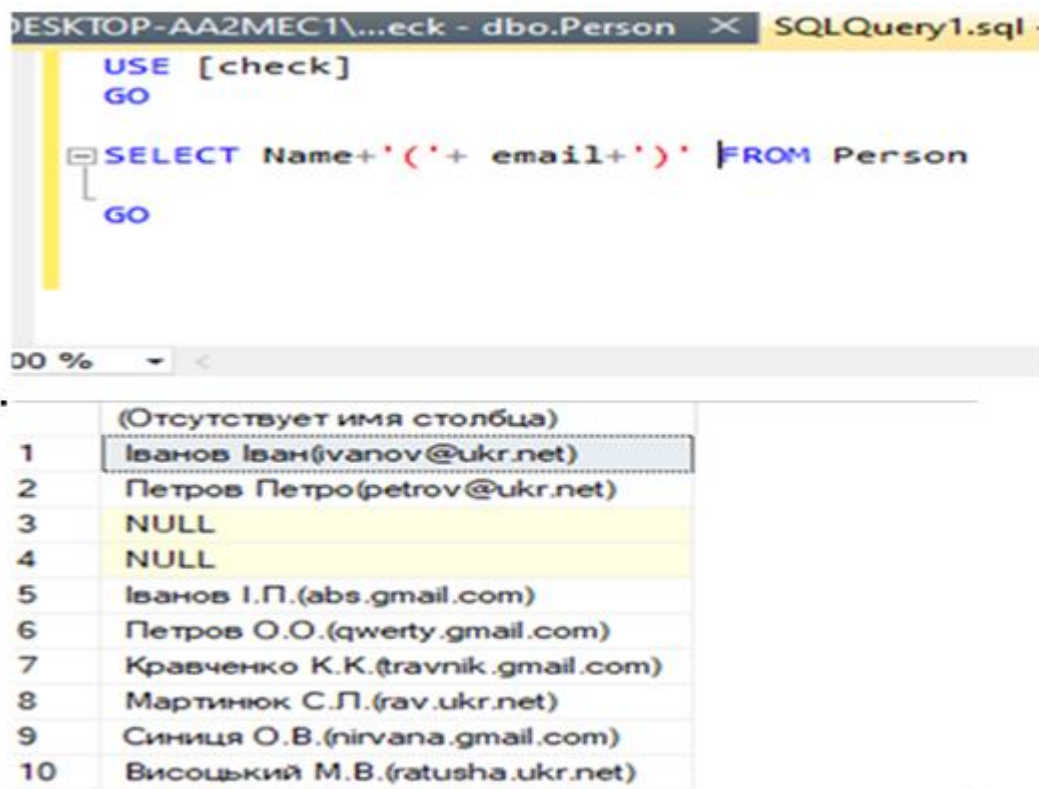


Рис. 4.4 Вибірка ПІБ осіб та їх електронних скриньок

Приклад 4.3.

Знову застосуємо команду SELECT для отримання ПІБ осіб та їх електронних скриньок в одному стовпці, але дамо ім'я цьому стовпцю (рис. 4.5).

Так як у деяких людей немає електронної скриньки, замість їх значень вивелось значення NULL.

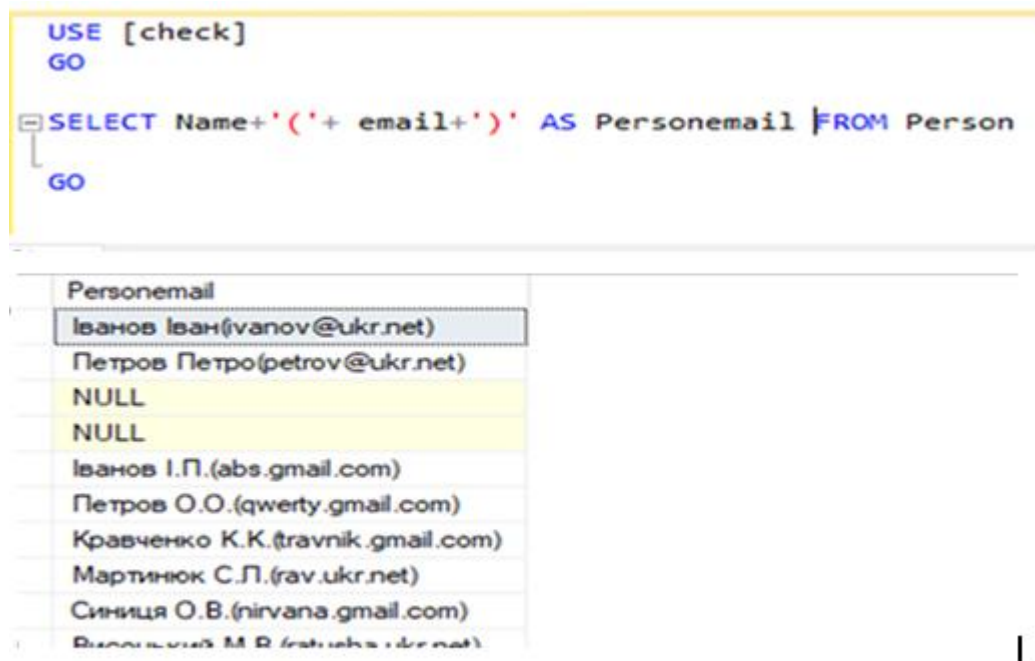


Рис. 4.5 Вибірка ПІБ осіб та їх електронних скриньок в одному стовпці з іменем Personemail

Виведемо результат цього запиту у віртуальну таблицю Notes (рис. 4.6).

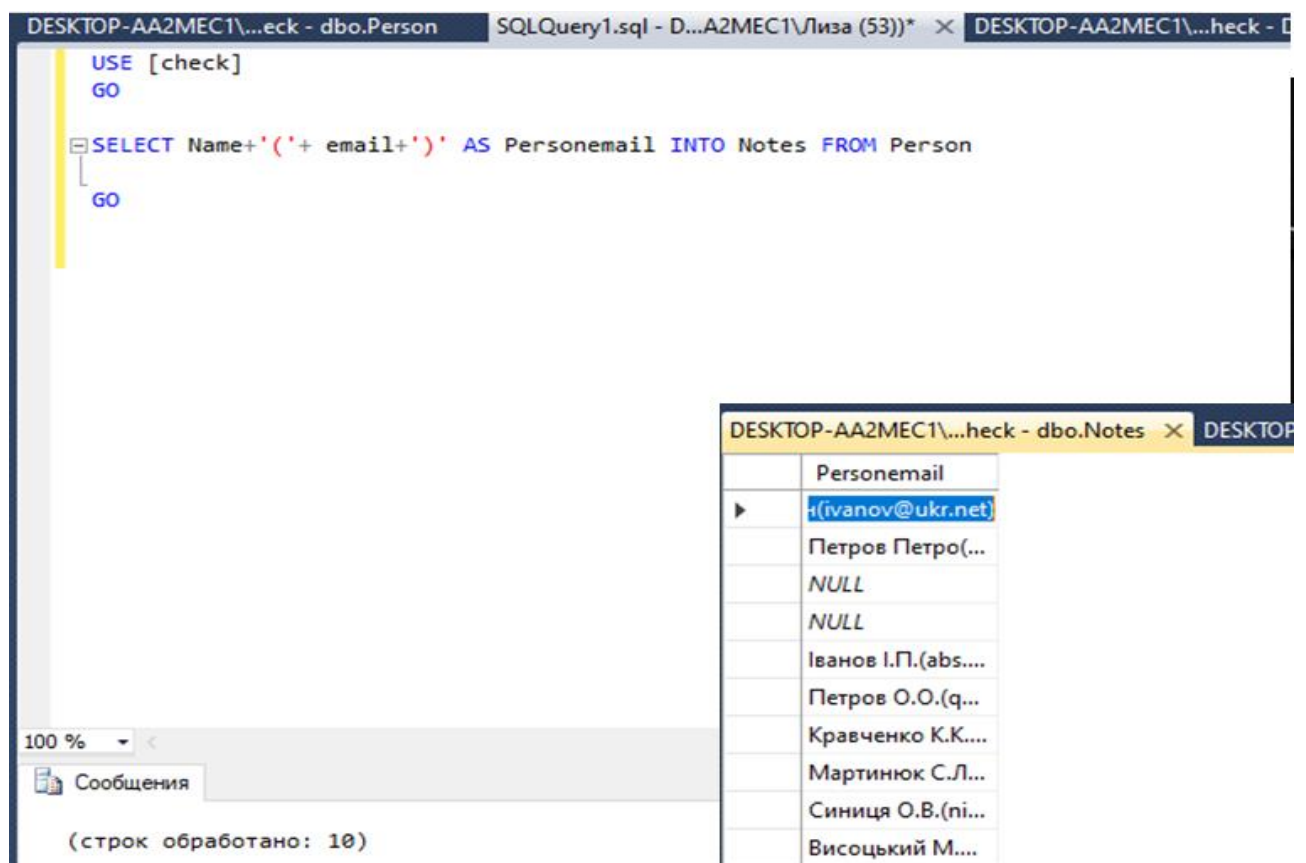


Рис. 4.6 Виведення вибірки у віртуальну таблицю Notes

4.2 Сортвання. ORDER BY

Оператор **ORDER BY** призначений для сортування вибірки даних, отриманої за допомогою оператору SELECT з бази даних в SQL.

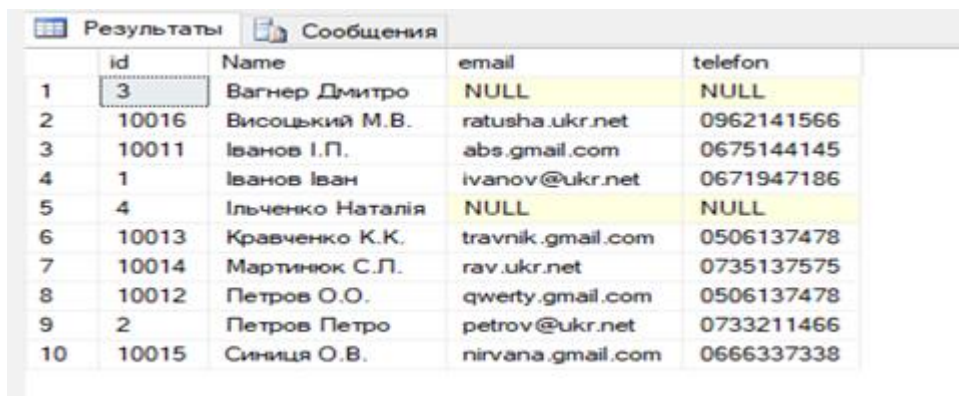
Приклад 4.4.

Застосуємо команду SELECT для отримання усіх фізичних осіб з таблиці Person, але вже за допомогою команди ORDER BY відсортуємо за алфавітом цю вибірку за стовпцем Name:

```
USE [check]
GO

SELECT * FROM Person
ORDER BY Name ;|
GO
```

Результат виконання скрипту показано на рис. 4.7.



The screenshot shows a window titled 'Результаты' (Results) with a tab 'Сообщения' (Messages). It displays a table with 5 columns: 'id', 'Name', 'email', and 'telefon'. The table contains 10 rows of data, sorted by the 'Name' column in ascending order. The first row has 'id' 3 and 'Name' 'Вагнер Дмитро'. The last row has 'id' 10015 and 'Name' 'Синица О.В.'. Some rows have NULL values for 'email' and 'telefon'.

	id	Name	email	telefon
1	3	Вагнер Дмитро	NULL	NULL
2	10016	Висоцький М.В.	ratusha.ukr.net	0962141566
3	10011	Іванов І.П.	abs.gmail.com	0675144145
4	1	Іванов Іван	ivanov@ukr.net	0671947186
5	4	Ільченко Наталія	NULL	NULL
6	10013	Кравченко К.К.	travnik.gmail.com	0506137478
7	10014	Мартинюк С.П.	rav.ukr.net	0735137575
8	10012	Петров О.О.	qwerty.gmail.com	0506137478
9	2	Петров Петро	petrov@ukr.net	0733211466
10	10015	Синица О.В.	nirvana.gmail.com	0666337338

Рис. 4.7 Вибірка фізичних осіб, відсортована за алфавітом за стовпцем Name

Приклад 4.5.

Отримаємо вибірку ПІБ осіб та їх електронних скриньок в одному стовпці. Вибірку відсортуємо в зворотному алфавітному порядку (від Я до А), вказавши ключове слово *DESC*:

```
USE [check]
GO

SELECT Name+'('+email+')' AS Personemail FROM Person
ORDER BY Personemail DESC ;

GO
```

Результат виконання скрипту показано на рис. 4.8.

	Personemail
1	Синиця О.В.(nirvana.gmail.com)
2	Петров Петро(petrov@ukr.net)
3	Петров О.О.(qwerty.gmail.com)
4	Мартинюк С.П.(rav.ukr.net)
5	Кравченко К.К.(travnik.gmail.com)
6	Іванов Іван(ivanov@ukr.net)
7	Іванов І.П.(abs.gmail.com)
8	Висоцький М.В.(ratusha.ukr.net)
9	NULL
10	NULL

Рис. 4.8 Відсортована в зворотному алфавітному порядку вибірка ПІБ осіб та їх електронних скриньок з таблиці Person в одному стовпці

Таку ж вибірку, але вже відсортовану у прямому порядку (від Я до А) можна отримати, якщо вказати параметр ASC:

```
USE [check]
GO

SELECT Name+'('+email+')' AS Personemail FROM Person
ORDER BY Personemail ASC ;

GO
```

Приклад 4.6.

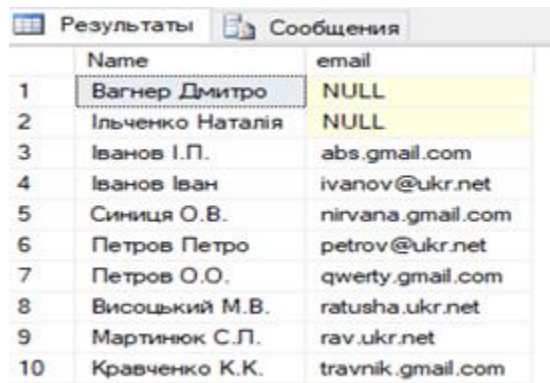
За допомогою команди ORDER BY можна сортувати за декількома стовпцями. Отримаємо вибірку лише двох стовпців – імен та поштових скриньок фізичних осіб з таблиці Person. Вибірку відсортуюмо за алфавітом за двома стовпцями – спочатку за email, а потім за Name:

```
USE [check]
GO

SELECT Name, email FROM Person
ORDER BY email, Name ;

GO
```

Результат запишемо у два стовпці (рис. 4.9).



	Name	email
1	Вагнер Дмитро	NULL
2	Ільченко Наталія	NULL
3	Іванов І.П.	abs.gmail.com
4	Іванов Іван	ivanov@ukr.net
5	Синиця О.В.	nirvana.gmail.com
6	Петров Петро	petrov@ukr.net
7	Петров О.О.	qwerty.gmail.com
8	Висоцький М.В.	ratusha.ukr.net
9	Мартинюк С.П.	rav.ukr.net
10	Кравченко К.К.	travnik.gmail.com

Рис. 4.9 Відсортована спочатку за email, а потім за Name вибірка ПІБ осіб та їх електронних скриньок з таблиці Person в двох стовпцях

Зверніть увагу, що спочатку сортування відбувалося за стовпцем, назва якого йде першою після ключового слова ORDER BY, а потім за стовпцем, назва якого що вказана другою.

4.3 TOP – вибірка з вказаної кількості перших рядків таблиці

Для того, щоб виконати вибірку з деякого числа перших рядків таблиці використовують після селекції ключове слово TOP.

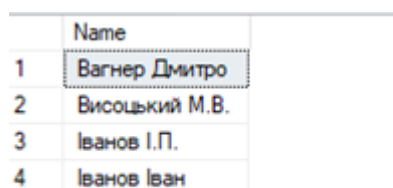
Приклад 4.7.

Виведемо відсортовану вибірку, яка складається лише з перших чотирьох записів таблиці Person і з одного стовпця Name:

```
USE [check]
GO

SELECT TOP 4 Name FROM Person
ORDER BY Name ;
GO
```

Результат:



	Name
1	Вагнер Дмитро
2	Висоцький М.В.
3	Іванов І.П.
4	Іванов Іван

Приклад 4.8.

Все те ж саме, що і в минулому прикладі, але цього разу виведемо 75% всіх записів:

```
USE [check]
GO

SELECT TOP 75 Percent Name FROM Person
ORDER BY Name ;
GO
```

Результат:

	Name
1	Вагнер Дмитро
2	Висоцький М.В.
3	Іванов І.П.
4	Іванов Іван
5	Ільченко Наталія
6	Кравченко К.К.
7	Мартинюк С.П.

4.4 OFFSET і FETCH – діапазон вибірки

Ключові слова OFFSET і FETCH є параметрами команди ORDER BY. Вони дають змогу обмежити кількість рядків, які повертаються за запитом. Інакше кажучи, ці параметри дають змогу визначити діапазон рядків таблиці, з якого буде зроблена вибірка.

OFFSET визначає кількість рядків, які потрібно пропустити, перш ніж почати повертати рядки за запитом. Після оператора FETCH ставиться ключове слово FIRST або NEXT, а потім вказується кількість рядків, які повертаються за запитом. Ці вибрані рядки (FETCH) вже йдуть після пропущених рядків (OFFSET). У спрощеному вигляді запит має такий синтаксис:

```
SELECT список_стовпців FROM ім'я_таблиці
ORDER BY вираз
OFFSET кількість_пропусків {ROW|ROWS}
FETCH { FIRST | NEXT } кількість_потрібних_рядків { ROW | ROWS } ONLY
```

Приклад 4.9.

Виведемо всі записи стовпця Name із таблиці Person, ігноруючи перших два рядка:

```
USE [check]
GO

SELECT Name FROM Person
ORDER BY id OFFSET 2 ROW;
GO
```

Результат:

	Name
1	Вагнер Дмитро
2	Ільченко Наталія
3	Іванов І.П.
4	Петров О.О.
5	Кравченко К.К.
6	Мартинюк С.Л.
7	Синиця О.В.
8	Висоцький М.В.

Приклад 4.10.

Виведемо три записи стовпця Name із таблиці Person, ігноруючи перших два рядка:

```
USE [check]
GO

SELECT Name FROM Person
ORDER BY id OFFSET 2 ROW
FETCH NEXT 3 ROWS ONLY;
GO
```

Результат:

	Name
1	Вагнер Дмитро
2	Ільченко Наталія
3	Іванов І.П.
4	Петров О.О.
5	Кравченко К.К.
6	Мартинюк С.Л.
7	Синиця О.В.
8	Висоцький М.В.

4.5 WHERE – фільтрація

Фільтрація – це процес селекції даних за деякою умовою, яка в SQL задається оператором WHERE.

В T-SQL можна застосовувати такі операції порівняння:

- = Порівняння на рівність;
- <> Порівняння на нерівність;
- < Менше ніж;
- > Більше ніж;
- !< Не менш ніж;
- !> Не більше;
- <= Менше ніж або дорівнює;
- >= Більше ніж або дорівнює.

Приклад 4.11.

Знайдемо людину в таблиці Person, у якої електронна скринька має назву abc.gmail.com:

```
USE [check]
GO

SELECT * FROM Person
WHERE email= 'abc.gmail.com'
GO
```

Результат:

	id	Name	email	telefon
1	10011	Іванов І.П.	abc.gmail.com	0675144145

4.6 Логічні оператори

Для об'єднання декількох умов в одному виразі можуть використовуватися логічні оператори. В T-SQL є такі логічні оператори:

- AND: операція логічного І. Вона об'єднує два вирази:

вираз1 AND вираз2

Тільки якщо обидва цих вирази одночасно будуть істинними, то і загальна умова оператора AND також буде істинною.

- OR: операція логічного АБО. Вона також об'єднує два вирази:

вираз1 OR вираз2

Якщо хоча б один з цих виразів є істинним, то загальна умова оператора OR також буде істинною.

- NOT: операція логічного заперечення. Якщо вираз в цій операції False, то загальна умова True.

NOT вираз

Якщо ці оператори зустрічаються в одному виразі, то спочатку виконується NOT, потім AND і в кінці OR.

Приклад 4.12.

Потрібно знайти всіх людей із таблиці Person, у яких електронна скринька буде або abc.gmail.com або qwerty.gmail.com:

```
USE [check]
GO

SELECT * FROM Person
WHERE email= 'abc.gmail.com' OR email= 'qwerty.gmail.com'
GO
```

Результат:

	id	Name	email	telefon
1	10011	Іванов І.П.	abc.gmail.com	0675144145
2	10012	Петров О.О.	qwerty.gmail.com	0506137478

4.7 IS NULL

Оператор IS NULL перевіряє чи пусте значення у стовпці таблиці.

Приклад 4.13.

Виведемо всіх людей із таблиці Person, у яких вказані їх електронні скриньки:

```
USE [check]
GO

SELECT * FROM Person
WHERE email IS NOT NULL;
GO
```

Результат:

	id	Name	email	telefon
1	1	Іванов Іван	ivanov@ukr.net	0671947186
2	2	Петров Петро	petrov@ukr.net	0733211466
3	10011	Іванов І.П.	abs.gmail.com	0675144145
4	10012	Петров О.О.	qwerty.gmail.com	0506137478
5	10013	Кравченко К.К.	travnik.gmail.com	0506137478
6	10014	Мартинюк С.Л.	rav.ukr.net	0735137575
7	10015	Синиця О.В.	nirvana.gmail.com	0666337338
8	10016	Висоцький М.В.	ratusha.ukr.net	0962141566

4.8 Оператори фільтрації

Оператор фільтрації IN перевіряє чи входить значення стовпця таблиці в заданий масив.

Приклад 4.14.

Із таблиці Person виберемо всю інформацію про людей, прізвища яких Іванов Іван, Кравченко К.К. і Синиця О.В.:

```
USE [check]
GO

SELECT * FROM Person
WHERE Name IN ('Іванов Іван', 'Кравченко К.К.', 'Синиця О.В.') ;
GO
```

Результат:

	id	Name	email	telefon
1	1	Іванов Іван	ivanov@ukr.net	0671947186
2	10013	Кравченко К.К.	travnik.gmail.com	0506137478
3	10015	Синиця О.В.	nirvana.gmail.com	0666337338

Оператор BETWEEN визначає діапазон значень, яким має відповідати вираз. Після ключового слова BETWEEN вказується початкове і кінцеве значення діапазону таким чином:

WHERE вираз [NOT] BETWEEN початкове_значення AND кінцеве_значення.

Оператор LIKE приймає шаблон рядка, якому має відповідати вираз.

WHERE вираз [NOT] LIKE 'шаблон_рядка'

Шаблон вказується в одинарних лапках. Для визначення шаблону можуть застосовуватися ряд спеціальних символів підстановки:

- % Відповідає будь-якому підрядку, наприклад, 'hello w%'. Підрядок може мати будь-яку кількість символів, а може не містити жодного символу;
- _ Відповідає будь-якому одиночному символу, наприклад, 'w_rd';
- [] Відповідає одному символу, який входить в список символів, вказаних в квадратних дужках, наприклад, [bcdfghklmnpqrstvxz];
- [-] Відповідає одному символу з певного діапазону, наприклад, [k-r];
- [^] Відповідає одному символу, який не входить в список чи діапазон після символу ^, наприклад, [^aeiouy] чи [^n-q].

Шаблон може містити будь-які символи (зокрема спеціальні) і будь-яку їх кількість, наприклад, 'hell[^a-d] w[aeiouy]rd'.

Приклад 4.15.

Із таблиці Person виведемо всіх людей, які використовують скриньку Google.

```
USE [check]
GO

SELECT * FROM Person
WHERE email LIKE '%gmail%';
GO
```

Результат:

	id	Name	email	telefon
1	10011	Іванов І.П.	abs.gmail.com	0675144145
2	10012	Петров О.О.	qwerty.gmail.com	0506137478
3	10013	Кравченко К.К.	travnik.gmail.com	0506137478
4	10015	Синиця О.В.	nirvana.gmail.com	0666337338

4.9 Видалення даних. Команда DELETE

Для видалення застосовується команда DELETE:

```
DELETE [FROM] ім'я_таблиці
WHERE умова _ видалення
```

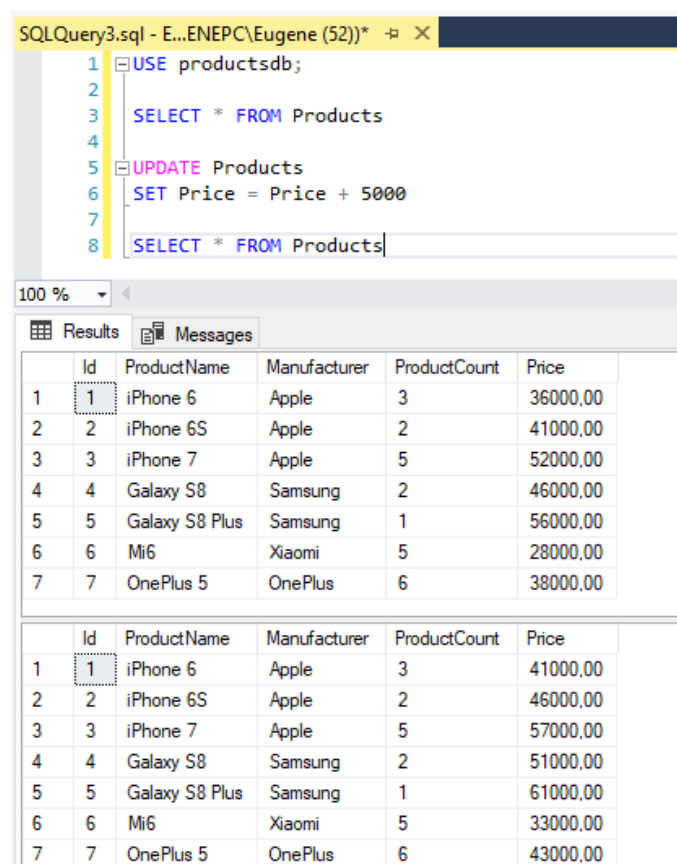
4.10 Оновлення даних. Команда UPDATE

Для зміни вже наявних рядків в таблиці застосовується команда UPDATE. Вона має такий формальний синтаксис:

```
UPDATE ім'я_таблиці  
SET стовпець1 = значення1, стовпець2 = значення2, ... стовпецьN = значенняN  
[FROM вибірка AS псевдонім_вибірки]  
[WHERE умова_оновлення]
```

Приклад 4.16.

Збільшимо у всіх товарів ціну на 5000 (рис. 4.10).



The screenshot shows a SQL query window titled 'SQLQuery3.sql - E...ENEPC\Eugene (52))*'. The query contains the following statements:

```
1 USE productsdb;  
2  
3 SELECT * FROM Products  
4  
5 UPDATE Products  
6 SET Price = Price + 5000  
7  
8 SELECT * FROM Products
```

Below the query window, the 'Results' tab displays two tables. The first table shows the initial state of the 'Products' table, and the second table shows the state after the update.

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	41000,00
2	2	iPhone 6S	Apple	2	46000,00
3	3	iPhone 7	Apple	5	57000,00
4	4	Galaxy S8	Samsung	2	51000,00
5	5	Galaxy S8 Plus	Samsung	1	61000,00
6	6	Mi6	Xiaomi	5	33000,00
7	7	OnePlus 5	OnePlus	6	43000,00

Рис. 4.10 Збільшення ціни в таблиці Product

ЗАВДАННЯ

1. Використовуючи командну консоль, напишіть запити на зміну трьох записів у своїх таблицях.
2. Використовуючи командну консоль, видаліть один запис із таблиці.
3. Напишіть запити в Azure Data Studio:

Варіант	Опис завдання
1	1. Вивести всі факультети, на яких вивчення інформаційних технологій є головним завданням: ФІТ, кібернетика, радіофізика тощо. 2. Вивести всі факультети, які знаходяться у певному корпусі. 3. Вивести всі кафедри, у яких прізвище завідуючого кафедри починається на літеру 'С'.
2	1. Вивести всі відділи супермаркету для певного залу. 2. Вивести всі відділи супермаркету, кількість прилавків у яких перевищує 3. 3. Вивести всіх співробітників супермаркету, телефон яких використовує зв'язок Lifecell.
3	1. Вивести всі частини за певним місцем дислокації. 2. Вивести всі роти відсортовані за номером у зворотному до алфавітного порядку. 3. Вивести місця дислокації військ в Україні, площа яких перевищує 10 кв. км.
4	1. Вивести всіх співробітників бібліотеки у алфавітному порядку, зарплата яких не більша ніж 5000 грн. 2. Вивести всі бібліотеки, розташовані у м. Києві. 3. Зробити вибірку для фондів бібліотеки, в якій вивести сумарну кількість книжок, журналів тощо.
5	1. Створити таблицю клієнти з полями: ПІБ, вік, кількість дітей, телефон. 2. Створити таблицю тур з полями: назва, пансіонат, вартість. 3. Вивести всіх клієнтів, телефон яких використовує зв'язок KYIVSTAR.
6	1.Створити таблицю Водії з полями: ПІБ, транспорт, місто проживання, номер телефону, вік, стаж роботи. 2.Вивести всіх водіїв, телефон яких використовує зв'язок

	Vodafone. Вибірка повинна бути відсортованою за віком водіїв за зростанням. 3. Вивести всіх робітників з обслуговуючого персоналу, які проживають у м. Києві і стаж роботи яких перевищує 5 років.
7	1. Створити таблицю Пацієнти з полями: ПІБ, стать, вік, сімейний лікар. 2. Створити таблицю історії хвороби, яка має такі атрибути: пацієнт, лікар, діагноз, лікування, дата захворювання, дата виліковування, вид лікування (амбулаторне, стаціонарне). Додати в неї 5 записів. 3.3 таблиці пацієнти вивести всіх чоловіків, прізвища яких відсортовані в алфавітному порядку.
8	1. Вивести всіх пацієнтів, які проживають у м. Києві та вік яких перевищує 30 років. 2. Вивести всіх лікарів, які мають звання лікар-спеціаліст. 3. Вивести всіх пацієнток, прізвища яких повинні бути відсортовані у зворотному до алфавітного порядку.
9	1. Створити таблицю Читачі з полями: ПІБ читача, вік читача, бібліотека. 2. Вивести прізвища і ім'я читачів бібліотеки за алфавітним порядком. 3. Вивести всіх читачів, вік яких більше ніж 12 років.
10	1. Створити таблицю Покупці з полями: ПІБ, стать, місто проживання, вік. 2. Вивести всіх покупців-жінок. 3. Вивести всіх покупців, віком не менше ніж 20 років, які проживають у м. Києві.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Для чого призначена мова визначення даних?
2. Для чого призначена мова маніпулювання даними?

3. Яким чином можливо перебрати всі записи із таблиці?
4. Які є в SQL логічні оператори?
5. Яким чином відбувається сортування в SQL?
6. Як в SQL задається умова відбору?

Лабораторна робота №5

Тема: Багатотабличні запити. Запити на з'єднання

План

1. З'єднання таблиць;
2. Додавання даних до зв'язаних таблиць
3. Декартів добуток таблиць
4. З'єднання таблиць
5. Завдання;
6. Питання для самоперевірки.

5.1 З'єднання таблиць

Розглянемо таке завдання. Нехай потрібно написати застосунок, який буде виставляти рахунки клієнтам (рис. 5.1).

Зразок заповнення платіжного доручення		
Одержувач	Товариство з обмеженою відповідальністю "С.В.С.ГРУП"	
Код	37165039	КРЕДИТ сак. N
Банк одержувача	Код банку	26001313318
АТ "Райффайзен Банк Аваль"	380805	

Рахунок на оплату № 5 від 14 листопада 2018 р.

Постачальник: Товариство з обмеженою відповідальністю "С.В.С.ГРУП"
П/р 26001313318, Банк АТ "Райффайзен Банк Аваль", МФО 380805
03131, м.Київ, вул.Любомирська, буд.3, тел.: (044) 3618907,
код за ЄДРПОУ 37165039, ІПН 371650326508, № свід. 100308810

Покупець: Київський національний університет імені Тараса Шевченка

Договір: Основний договір

№	Товари (роботи, послуги)	Кіл-сть	Од.	Ціна без ПДВ	Сума без ПДВ
1	Труба водостічна ПЕ RAL 3011 д.150, L=1250мм	50	шт	222,80	11 140,00
2	Нижній елемент труби ПЕ (коліно) д.150мм L=590 мм кут. 120	28	шт	318,46	8 916,88
3	Лійки водоприймальні ПЕ д.150мм L=300 мм	5	шт	413,46	2 067,30
4	Кронштейн для кріплення водостічних труб (хомут 20*300мм з пластиною 20*120мм)	72	шт	160,23	11 536,56

Всього: 33 660,74
Сума ПДВ: 6 732,15
Всього із ПДВ: 40 392,89

Рис. 5.1 Платіжні рахунки клієнтам

Для цього потрібна БД, яка буде включати такі таблиці:

1. *Фізичні особи*
2. *Контрагенти*
3. *Співробітники*
4. *Товари*
5. *Документи*

Вже маємо створену у 4 лабораторній роботі БД check, яка містить три перші таблиці (рис. 5.2):

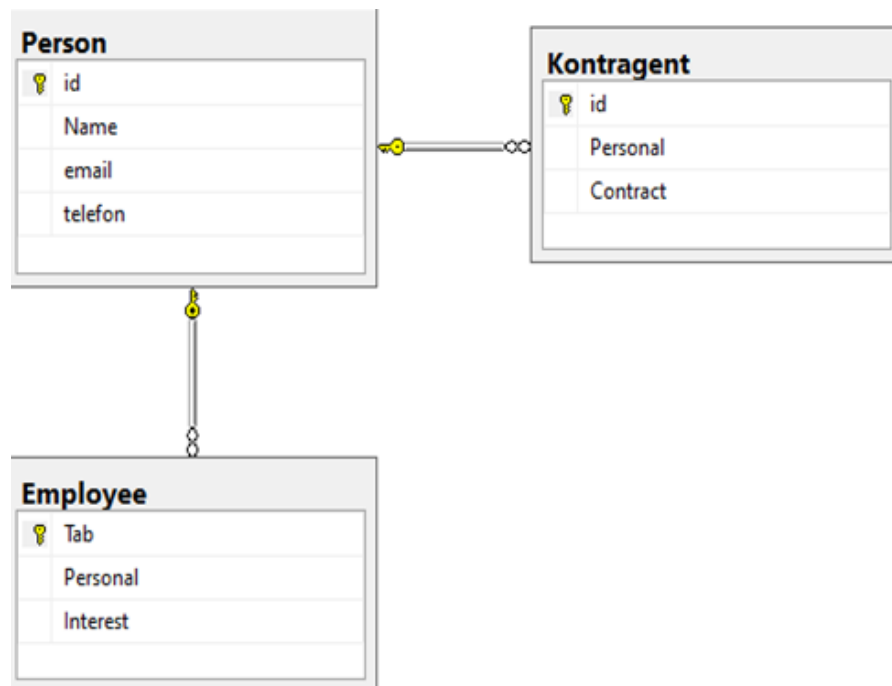


Рис. 5.2 Діаграма бази даних check, яка містить 3 таблиці

Потрібно створити ще дві таблиці: Goods і Docs, в яких буде зберігатися інформація про товари і документи відповідно. Створимо таблицю Goods засобами Microsoft SQL Management Studio (рис. 5.3).

	Id	int	☐
	Name	nvarchar(50)	☒
	Price	real	☒
			☐

Рис. 5.3 Створення таблиці Goods з трьома полями – ID, назва і ціна товару

Заповнимо цю таблицю даними за допомогою такого запиту:

```

USE [check]
GO

INSERT INTO [dbo].[goods]
(
    [Name]
    , [Price]
)
VALUES
(
    'Пальто жін. синє', 2500.00),
(
    'Пальто жін. зимнє, демісизонне', 4500.00),
(
    'Пальто жін. чорне', 12400.00),
(
    'Пальто унісекс. чорне', 5000.00),
(
    'Пальто чол. клітинка', 12500.00),
(
    'Пальто чол. чорне', 6500.00);

GO

```

У результаті отримаємо таблицю, яка має 6 записів (рис. 5.4).

	Id	Name	Price
►	1	Пальто жін. синє	2500
	2	Пальто жін. зимнє, демісиз...	4500
	3	Пальто жін. чорне	12400
	4	Пальто унісекс. чорне	5000
	5	Пальто чол. клітинка	12500
	6	Пальто чол. чорне	6500
*	NULL	NULL	NULL

Рис. 5.4 Таблиця Goods, заповнена даними

Кожен платіжний документ виписує співробітник компанії для клієнта. Тобто таблиця, яка зберігає інформацію про документи, повинна посилатися на співробітника із таблиці Employee і на клієнта із таблиці Kontragent. Водночас в кожному платіжному документі може бути багато товарів, а кожен товар може бути в багатьох платіжних документах – маємо зв'язок «багато-до-багатьох». Такий зв'язок потребує додаткової таблиці, в якій буде відображатися id документа, що містить певний товар, і id цього товару.

Створимо таблицю, яка зберігає інформацію про документи, за допомогою запити:

```

CREATE TABLE Docs
(
    Num INT PRIMARY KEY IDENTITY,
    KontragentId Numeric,
    EmployeeId Numeric,
    CreatedAt Date,
    CONSTRAINT FK_Doc_To_Kontragent FOREIGN KEY (KontragentId) REFERENCES Kontragent (Id),
    CONSTRAINT FK_Doc_To_Employee FOREIGN KEY (EmployeeId) REFERENCES Employee (Tab)
);

```

Створимо таблицю зв'язків між документами і товарами за допомогою запиту:

```

USE [check]
GO

```

```

CREATE TABLE GoodsDoc
(
    Num INT ,
    GoodsId INT,
    Q INT,
    PRIMARY KEY(Num, GoodsId),
    CONSTRAINT FK_Doc_To_Goods FOREIGN KEY (GoodsId) REFERENCES Goods (Id),
    CONSTRAINT FK_Doc_To_Doc FOREIGN KEY (Num) REFERENCES Docs (Num)
)

```

У цьому випадку ми пов'язали таблиці, указавши обмеження FOREIGN KEY. Синтаксис цього оператора такий:

CONSTRAINT *назва_обмеження* **FOREIGN KEY** (*назва_зовнішнього_ключа*)
REFERENCES *таблиця_що_зв'язується* (*Первинний_ключ_цієї_таблиці*)

У результаті отримаємо базу даних, яка має 5 зв'язаних таблиць (рис. 5.3).

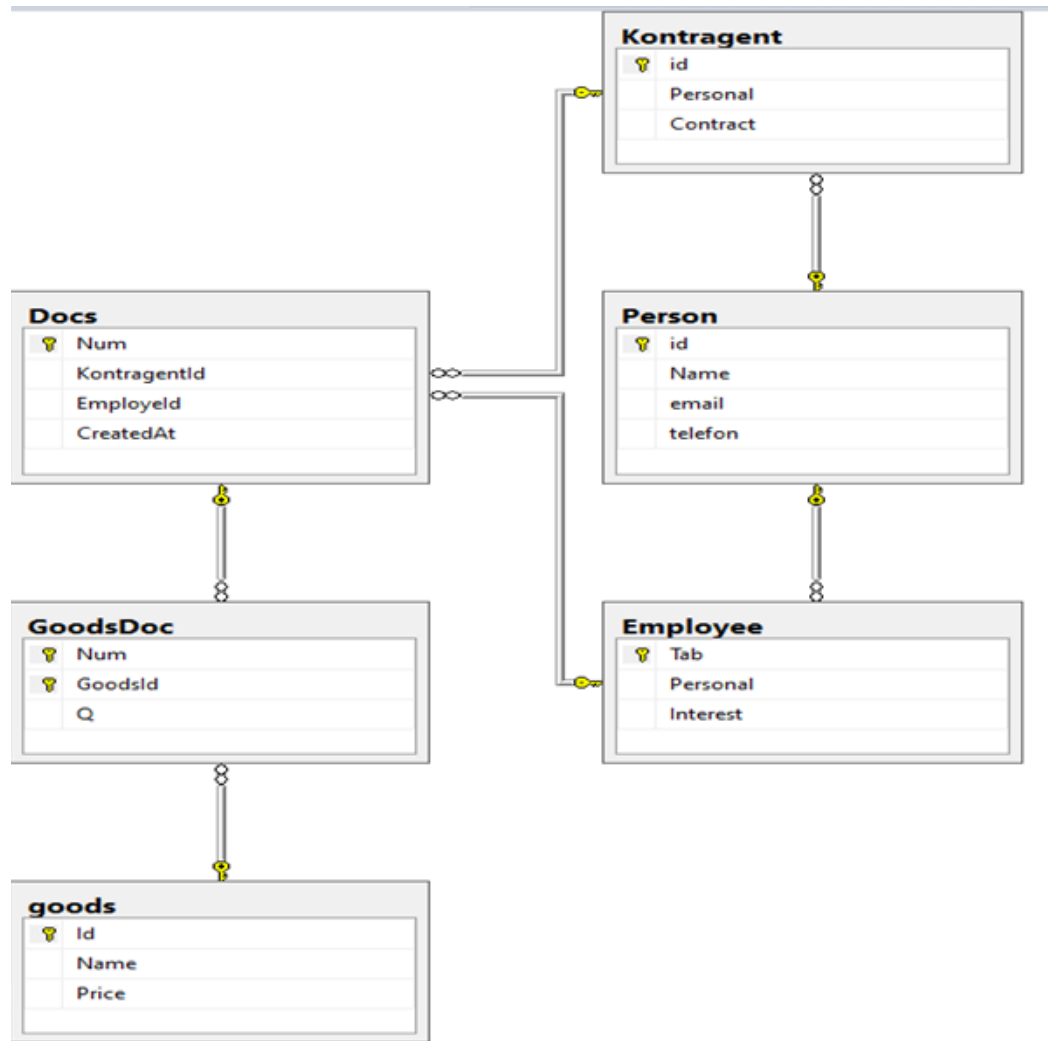


Рис. 5.5 Діаграма бази даних check, яка містить 5 таблиць

5.2 Додавання даних до зв'язаних таблиць

Для додавання записів до таблиць використовується оператор INSERT INTO.

Спрощений запит на додавання має такий синтаксис:

```
INSERT INTO назва_цільової_таблиці (поле1, поле2, ...)
VALUES (значення1, значення2, ...)
```

Для зв'язування таблиць замість кожного значення, що міститься у круглих дужках після ключового слова VALUES використовується такий запис:

```
SELECT поле1, поле2... FROM назва_таблиці_джерела
```

Оператор SELECT може включати ключове слово WHERE для фільтрування даних:

```
SELECT поле1, поле2...
```

```
FROM назва_таблиці_джерела WHERE вираз_для_фільтрування
```

Зверніть увагу на порядок розташування назв полів після операторів SELECT та INSERT INTO, оскільки дані з першого вказаного після SELECT стовпця «поле1» таблиці «назва_таблиці_джерела», будуть незалежно від назви поля заповнені в перший вказаний після оператора INSERT INTO стовпець «поле1» таблиці «назва_цільової_таблиці».

Приклад 5.1.

Додамо відфільтровані дані у таблицю Docs, яка пов'язана з таблицями Kontragent та Employee:

```
INSERT INTO Docs ( KontragentId, EmployeeId,
CreatedAt) VALUES (
(SELECT id FROM Kontragent WHERE id = 1),
(SELECT Tab FROM Employee WHERE Tab = 1),
GetDate());
```

Результат:

DESKTOP-AA2MEC1\....check - dbo.Docs X SQLQuery2.sql - D...A2MEC1\Лиза (59)**				
	Num	KontragentId	EmployeeId	CreatedAt
▶	1	1	1	2021-09-21
•	NULL	NULL	NULL	NULL

Зверніть увагу, що в третій стовпець додано поточну дату.

Додамо дані у таблицю GoodsDoc, яка пов'язана з таблицями Docs і Goods:

```
USE [check]
GO

INSERT INTO GoodsDoc ( Num, GoodsId, Q) VALUES (
(SELECT Num FROM Docs WHERE Num = 1),
(SELECT Id FROM Goods WHERE Id = 1),
5);|
```

Зверніть увагу, що в третій стовпець додано просто число 5.

Результат:

	Num	GoodsId	Q
▶	1	1	5
•	NULL	NULL	NULL

Ще додамо відфільтровані дані у таблицю Docs:

```
USE [check]
GO

INSERT INTO Docs ( KontragentId, EmployeeId, CreatedAt) VALUES (
(SELECT id FROM Kontragent WHERE id = 2),
(SELECT Tab FROM Employee WHERE Tab = 2),
'2021-05-21');|
```

Додамо одразу декілька записів у таблицю GoodsDoc:

```
INSERT INTO GoodsDoc ( Num, GoodsId, Q) VALUES (
(SELECT Num FROM Docs WHERE Num = 3),
(SELECT Id FROM Goods WHERE Id = 4),
2), (
(SELECT Num FROM Docs WHERE Num = 3),
(SELECT Id FROM Goods WHERE Id = 2),
3), (
(SELECT Num FROM Docs WHERE Num = 3),
(SELECT Id FROM Goods WHERE Id = 3),
4)|
```

Загальний результат виконання усіх цих записів подано на рис. 5.6.

PERSON

	id	Name	email	telefon
▶	1	Іванов Іван	ivanov@ukr.net	0671947186
	2	Петров Петро	petrov@ukr.net	0733211466
	3	Вagner Дмитро	NULL	NULL
	4	Ільченко Ната...	NULL	NULL
	10011	Іванов І.П.	abs@gmail.com	0675144145
	10012	Петров О.О.	qwerty@gmail.c...	0506137478
	10013	Кравченко К.К.	travnik@gmail.c...	0506137478
	10014	Мартинюк С.Л.	rav.ukr.net	0735137575
	10015	Синиця О.В.	nirvana@gmail.c...	0666337338
	10016	Висоцький М.В.	ratusha.ukr.net	0962141566
•	NULL	NULL	NULL	NULL

EMPLOYEE

	Tab	Personal	Interest
▶	1	1	5
	2	2	10
•	NULL	NULL	NULL

KONTRAGENT

	id	Personal	Contract
▶	1	3	105
	2	4	106
•	NULL	NULL	NULL

DOC

	KontraгентId	EmployeId	CreatedAt
	1	1	2021-09-21
		2	2021-02-21
		2	2021-05-21

Goods

	Id	Name	Price
▶	1	Пальто жін. синє	2500
	2	Пальто жін. зимнє, деміс...	4500
	3	Пальто жін. чорне	12400
	4	Пальто унісекс. чорне	5000
	5	Пальто чол. клітинка	12500
	6	Пальто чол. чорне	6500
•	NULL	NULL	NULL

GoodsDoc

	Num	GoodsId	Q
▶	1	1	5
	2	1	2
	2	2	5
	2	3	4
	3	2	3

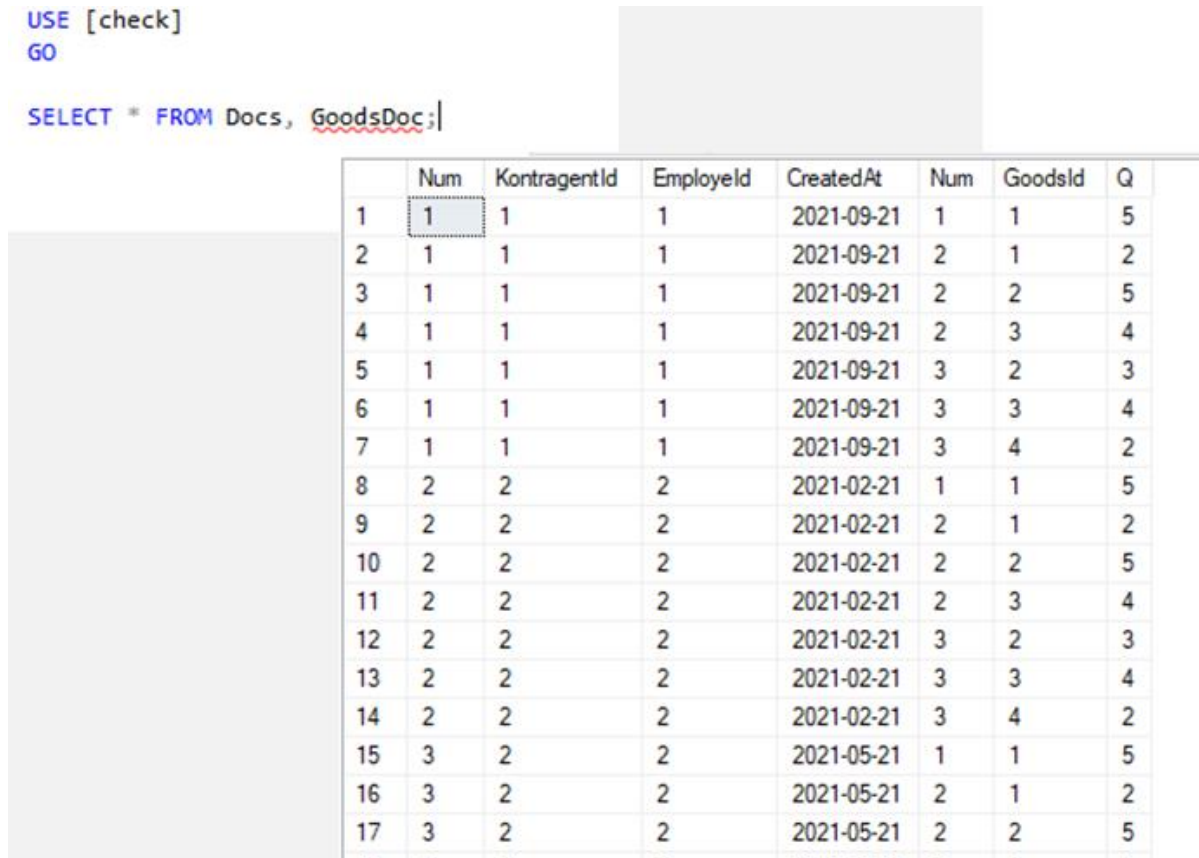
Рис. 5.6 Усі таблиці бази даних check

5.3 Декартів добуток таблиць

В результаті декартового добутку двох таблиць отримуємо таблицю, яка складається із усіх можливих об'єднань записів однієї таблиці із записами другої таблиці.

Приклад 5.2.

Зробимо декартів добуток таблиць Docs і GoodsDoc (рис. 5.7).



```
USE [check]
GO

SELECT * FROM Docs, GoodsDoc;
```

	Num	KontragentId	EmployeId	CreatedAt	Num	GoodsId	Q
1	1	1	1	2021-09-21	1	1	5
2	1	1	1	2021-09-21	2	1	2
3	1	1	1	2021-09-21	2	2	5
4	1	1	1	2021-09-21	2	3	4
5	1	1	1	2021-09-21	3	2	3
6	1	1	1	2021-09-21	3	3	4
7	1	1	1	2021-09-21	3	4	2
8	2	2	2	2021-02-21	1	1	5
9	2	2	2	2021-02-21	2	1	2
10	2	2	2	2021-02-21	2	2	5
11	2	2	2	2021-02-21	2	3	4
12	2	2	2	2021-02-21	3	2	3
13	2	2	2	2021-02-21	3	3	4
14	2	2	2	2021-02-21	3	4	2
15	3	2	2	2021-05-21	1	1	5
16	3	2	2	2021-05-21	2	1	2
17	3	2	2	2021-05-21	2	2	5

Рис. 5.7 Декартів добуток таблиць Docs і GoodsDoc

Зверніть увагу, що стовпці йдуть у тому ж порядку, в якому вказані імена таблиць у запиті після ключового слова FROM. Тобто спочатку йдуть усі стовпці таблиці Docs, а потім усі стовпці таблиці GoodsDoc.

Кількість рядків у результаті буде рівна добутку кількості рядків обох таблиць (Docs і GoodsDoc).

Якщо потрібно об'єднувати записи за певним критерієм, то використовується оператор фільтрації (WHERE).

Приклад 5.3.

Зробимо декартів добуток таблиць Docs і GoodsDoc, але вже об'єднаємо тільки ті записи, у яких співпадають значення стовпців Num за допомогою ключового слова WHERE. Виведемо лише номер документу, дату документу і id товару:

```
USE [check]
GO

SELECT Docs.Num, Docs.CreatedAt, GoodsDoc.GoodsId FROM Docs, GoodsDoc
WHERE Docs.Num = GoodsDoc.Num;
```

Результат:

	Num	CreatedAt	GoodsId
1	1	2021-09-21	1
2	2	2021-02-21	1
3	2	2021-02-21	2
4	2	2021-02-21	3
5	3	2021-05-21	2
6	3	2021-05-21	3
7	3	2021-05-21	4

5.4 З'єднання таблиць

З'єднання таблиць може бути вказано за допомогою оператора **WHERE** або оператора **FROM**. Можна виділити такі категорії: внутрішні, зовнішні та перехресні з'єднання.

Внутрішні з'єднання

Внутрішні з'єднання використовують оператори порівняння для встановлення відповідності записів з двох сутностей на основі значень загальних стовпців в кожній таблиці.

Приклад 5.4.

Виведемо телефони всіх співробітників

```
USE [check]
GO

SELECT Person.Name, Person.telefon, Employee.Tab
FROM Person
JOIN Employee ON Person.Id = Employee.Personal;
```

Результат:

	Name	telefon	Tab
1	Іванов Іван	0671947186	1
2	Петров Петро	0733211466	2

Зовнішні з'єднання.

Зовнішні з'єднання бувають лівими, правими і повними.

Якщо зовнішні з'єднання задаються в FROM, вони вказуються з одним з таких наборів ключових слів:

- LEFT JOIN або LEFT OUTER JOIN
- RIGHT JOIN або RIGHT OUTER JOIN
- FULL JOIN або FULL OUTER JOIN

Результуючий набір лівого зовнішнього з'єднання включає всі рядки з лівої таблиці, заданої в реченні LEFT OUTER, а не тільки ті, в яких сполучаються стовпці, що відповідають один одному. Якщо рядок в лівій таблиці не має збігів з рядком в правій таблиці, результуючий набір рядків містить значення NULL для всіх стовпців списку вибору з правої таблиці.

Приклад 5.5.

Виведемо інформацію про співробітників з використанням LEFT JOIN:

```
USE [check]
GO

SELECT *
FROM Person
LEFT JOIN Employee ON Person.Id = Employee.Personal;
```

Результат:

	id	Name	email	telefon	Tab	Personal	Interest
1	1	Іванов Іван	ivanov@ukr.net	0671947186	1	1	5
2	2	Петров Петро	petrov@ukr.net	0733211466	2	2	10
3	3	Вагнер Дмитро	NULL	NULL	NULL	NULL	NULL
4	4	Ільченко Наталія	NULL	NULL	NULL	NULL	NULL
5	10011	Іванов І.П.	abs@gmail.com	0675144145	NULL	NULL	NULL
6	10012	Петров О.О.	qwerty@gmail.com	0506137478	NULL	NULL	NULL
7	10013	Кравченко К.К.	travnik@gmail.com	0506137478	NULL	NULL	NULL
8	10014	Мартинюк С.П.	rav.ukr.net	0735137575	NULL	NULL	NULL
9	10015	Синиця О.В.	nirvana@gmail.com	0666337338	NULL	NULL	NULL
10	10016	Висоцький М.В.	ratusha.ukr.net	0962141566	NULL	NULL	NULL

Праве зовнішнє з'єднання є зворотним для лівого зовнішнього з'єднання. Повертаються всі рядки правої таблиці. Для лівої таблиці повертаються значення NULL кожен раз, коли рядок правої таблиці не має збігів рядків в лівій таблиці.

Приклад 5.6.

Виведемо інформацію про співробітників з використанням RIGHT JOIN:

```
USE [check]
GO

SELECT *
FROM Person
RIGHT JOIN Employee ON Person.Id = Employee.Personal;
```

Результат:

	id	Name	email	telefon	Tab	Personal	Interest
1	1	Іванов Іван	ivanov@ukr.net	0671947186	1	1	5
2	2	Петров Петро	petrov@ukr.net	0733211466	2	2	10

Повне зовнішнє з'єднання повертає всі рядки з правої і лівої таблиці. Кожен раз, коли рядок не має відповідних значень в іншій таблиці, стовпці списку вибору іншої таблиці містять значення NULL. Якщо між таблицями є відповідність, весь рядок результуючого набору містить значення даних з базових таблиць.

Перехресні з'єднання

Перехресне з'єднання повертає всі рядки з лівої таблиці. Кожен рядок з лівої таблиці з'єднується з усіма рядками з правої таблиці. Перехресні з'єднання називаються також декартовим добутком.

Таблиці або подання в реченні FROM можуть зазначатися в будь-якому порядку з внутрішнім з'єднанням або повним зовнішнім з'єднанням. Однак важливий порядок таблиць або подань, заданих з використанням лівого або правого зовнішнього з'єднання.

Приклад 5.7.

```
USE [check]
GO

SELECT *
FROM Person
CROSS JOIN Employee ;
```

Результат:

Результаты		Сообщения					
	id	Name	email	telefon	Tab	Personal	Interest
1	1	Іванов Іван	ivanov@ukr.net	0671947186	1	1	5
2	2	Петров Петро	petrov@ukr.net	0733211466	1	1	5
3	3	Вагнер Дмитро	NULL	NULL	1	1	5
4	4	Ільченко Наталія	NULL	NULL	1	1	5
5	10011	Іванов І.П.	abs.gmail.com	0675144145	1	1	5
6	10012	Петров О.О.	qwerty.gmail.com	0506137478	1	1	5
7	10013	Кравченко К.К.	travnik.gmail.com	0506137478	1	1	5
8	10014	Мартинюк С.П.	rav.ukr.net	0735137575	1	1	5
9	10015	Синиця О.В.	nirvana.gmail.com	0666337338	1	1	5
10	10016	Висоцький М.В.	ratusha.ukr.net	0962141566	1	1	5
11	1	Іванов Іван	ivanov@ukr.net	0671947186	2	2	10
12	2	Петров Петро	petrov@ukr.net	0733211466	2	2	10
13	3	Вагнер Дмитро	NULL	NULL	2	2	10
14	4	Ільченко Наталія	NULL	NULL	2	2	10
15	10011	Іванов І.П.	abs.gmail.com	0675144145	2	2	10
16	10012	Петров О.О.	qwerty.gmail.com	0506137478	2	2	10
17	10013	Кравченко К.К.	travnik.gmail.com	0506137478	2	2	10
18	10014	Мартинюк С.П.	rav.ukr.net	0735137575	2	2	10

ЗАВДАННЯ

Продовжуйте працювати зі своєю БД. Таблиці мають бути вже зв'язані зовнішнім ключем.

Виконайте в консолі:

1. Декартовий добуток двох таблиць із БД. Таблиці на ваш вибір.
2. Ліве зовнішнє з'єднання двох таблиць із БД (Таблиці ті самі).
3. Праве зовнішнє з'єднання двох таблиць із БД (Таблиці ті самі).
4. Виконайте запити засобами Microsoft SQL Management Studio – індивідуальні завдання за варіантом (подані нижче). Побудуйте діаграму БД.

Варіант	Опис завдання
1	1. Створити таблицю групи з полями: номер групи, кафедра. Додати в неї 3 записи. 2. Створити таблицю студенти з полями: номер залікової книги, ПІБ, група. Додати в неї 10 записів. 3. Створити таблицю викладачі з полями: Табельний номер, ПІБ, кафедра.
2	1. Створити таблицю посади з полями: назва посади, оклад, відділ. Зв'язати її з таблицями Співробітники і Відділи. Додати в неї 5 записів. Змінити записи в таблиці Співробітники. 2. Створити таблицю товари з полями: Назва товару, одиниця виміру, ціна товару, виробник. 3. Створити таблицю продажу товарів, яка матиме такі атрибути: Співробітник-продавець товару, дата і час продажу, назва і кількість товару. Додати в неї 5 записів. Зв'язати її з іншими таблицями.
3	1. Створити таблицю особового складу військовослужбовців, яка має такі атрибути: ПІБ, рота, посада, рік народження, рік надходження на службу, вислуга (кількість років), нагороди, участь у військових заходах. Зв'язати з іншими таблицями. Додати в неї 5 записів. 2. Вивести весь особовий склад з назвою частини, роти, виду військ. 3. Вивести інформацію про військовослужбовців, і про їх дислокацію певної частини на певну дату.
4	1. Створити таблицю поповнення фонду бібліотеки, яка має такі атрибути: фонд, співробітник (який додав літературу у фонд), дата, назва джерела літератури, тип літератури, видавництво, дата видання, кількість примірників. Додати в неї 5 записів. 2. Вивести назви і адреси бібліотек, фонд яких перевищує 1000 екземплярів з типом літератури «Наукова».

	3. Вивести інформацію про бібліотеку, в якій працює певний співробітник.
5	1. Створити таблицю продажу путівок, яка має такі атрибути: ПІБ клієнта, назва пансіонату, вид житла, дата заїзду, дата від'їзду, наявність дітей, наявність мед. страховки, к-ть чоловік, ціна за 1 особу, загальна сума. Додати в неї 5 записів. 2. Вивести всіх клієнтів пансіонату, дата заїзду яких не пізніше вказаної дати (наприклад, 01.09.2023 р.) для певного туру по Європі. 3. Вивести всі тури відсортовані в алфавітному порядку за назвою.
6	1. Створити таблицю гаражне господарство, яка має такі атрибути: назва гаража, транспорт на ремонті, вид ремонту, дата надходження, дата видачі після ремонту, результат ремонту, ремонтна бригада. Додати в неї 5 записів. 2. Вивести всіх водіїв, і назви транспорту, які працюють за певним маршрутом та їх автівка належить до певного гаражного господарства. 3. Вивести інформацію про транспорт, який зараз перебуває на ремонті.
7	1. Вивести інформацію про відділення, в якому лікувався певний пацієнт. 2. Вивести всю інформацію про лікарів і про відділення в яких вони працюють. 2. Вивести всі відомості про пацієнтів, яких лікував певний лікар, з певним діагнозом, з датою захворювання до вказаної дати (наприклад, не пізніше, ніж 01.09.2023 р.).
8	1. Створити таблицю операцій, яка має такі атрибути: опис операції (видалення апендициту, пластична операція тощо), лікар, дата операції, пацієнт, результат операції. Додати в неї 5 записів.

	2. Вивести всі відомості про пацієнтів, яких оперував певний хірург, операція яких проходила після вказаної дати (наприклад, не раніше, ніж 01.09.2023 р.). 3. Вивести інформацію про лікарів, які оперували певної дати.
9	1. Створити таблицю література з полями: автор, назва, жанр, бібліотека, видавець, рік видання. 2. Створити таблицю карток читача, яка має такі атрибути: читач, література, дата видачі, термін видачі, вид видачі, наявність застави. 3. Вивести всі відомості про літературу, яку читає певний читач.
10	1. Створити таблицю Співробітники, яка має наступні поля: ПІБ, стаж роботи, відсоток від продажу. 2. Створити таблицю продажів автомобілів. Таблиця продажів автомобілів має такі атрибути: дата, співробітник, автомобіль, покупець. Додати в неї 5 записів. 3. Вивести всі відомості про автомобілі, які продані певним співробітником за вказаний період часу, наприклад, з 01.08.2023 р. до 31.08.2023 р.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Назвіть алгоритм складання багатотабличних запитів.
2. Що таке з'єднання?
3. Які категорії з'єднань у реляційних БД ви знаєте?
4. Назвіть різницю між декартовим добутком і перехресним з'єднанням.
5. Покажіть різницю між лівим зовнішнім з'єднанням і правим?

Лабораторна робота №6

Тема: Створення облікових записів, користувачів, ролей

План

1. Система безпеки SQL сервера;
2. Створення облікового запису;
3. Створення користувача;
4. Створення ролей;
5. Завдання;
6. Питання для самоперевірки.

6.1 Система безпеки SQL сервера

Користувачі БД можуть мати різні права і ролі по відношенню до конкретної БД: одні користувачі можуть додавати, видаляти, редагувати записи у БД, інші лише переглядати дані з цієї БД задля безпеки і збереження даних. Тому створення користувача і надання йому ролей в SQL має велике значення.

В SQL Server впроваджено два рівні безпеки: на рівні сервера і на рівні баз даних. На рівні сервера система безпеки працює з обліковим записом (login), автентифікацією і з вбудованими ролями сервера. На рівні бази даних система безпеки працює з користувачами (user), роллю в базі даних (рис. 6.1).

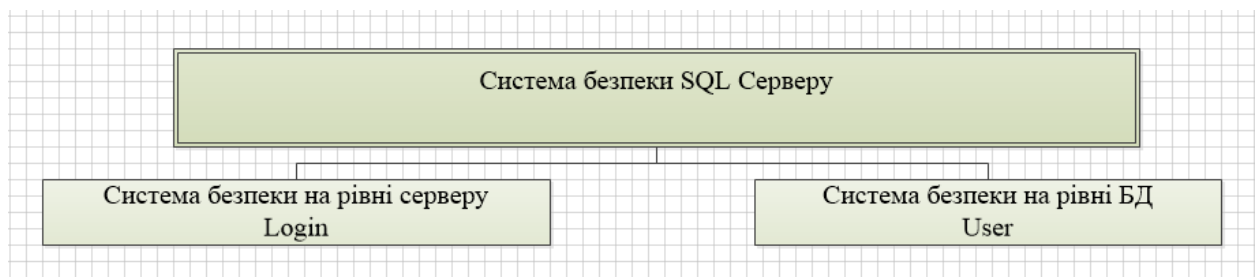


Рис. 6.1 Організація системи безпеки в SQL сервері

6.2 Створення облікового запису

Створення облікового запису може відбуватися шляхом використання системних процедур або засобами мови SQL.

Взагалі, для створення облікового запису потрібно виконати три етапи:

1. Створити обліковий запис;
2. Додати доступ до БД;
3. Надати йому привілеї.

Розглянемо спочатку створення облікового запису сервера шляхом системних процедур. Щоб додати новий обліковий запис використовують процедуру `sp_addlogin`, яка має такий формат:

```
sp_addlogin  
[@login=] 'обліковий_запис'  
[, [@password=] 'пароль'  
[, [@defdb=] 'база_даних_за_замовчуванням']
```

Приклад 6.1.

Створимо новий обліковий запис сервера з ім'ям Kseniia і паролем – 2. Для цього створимо новий запит:

```
exec sp_addlogin @loginame = 'Kseniia', @passwd='2'
```

Виконаємо цей запит і перевіримо чи правильно спрацював. Для цього у провіднику об'єктів SSMS відкриємо гілку Security, а потім теку Logins (рис. 6.2).

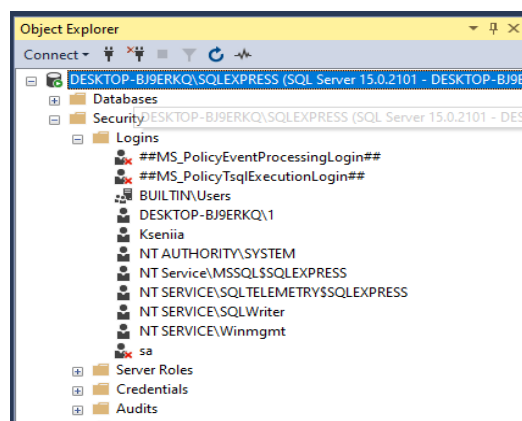


Рис. 6.2 В теці Logins з'явився обліковий запис Kseniia

Також створення облікового запису може відбуватися за допомогою команди CREATE. Синтаксис команди для SQL Server у цьому разі такий:

```
CREATE LOGIN login_name { WITH <option_list1> | FROM <sources> }
```

<option_list1> ::=

```
PASSWORD = { 'password' | hashed_password HASHED } [ MUST_CHANGE ]  
[ , <option_list2> [ ,... ] ]
```

<option_list2> ::=

```
SID = sid  
| DEFAULT_DATABASE = database  
| DEFAULT_LANGUAGE = language  
| CHECK_EXPIRATION = { ON | OFF}  
| CHECK_POLICY = { ON | OFF}  
| CREDENTIAL = credential_name
```

<sources> ::=

```
WINDOWS [ WITH <windows_options>[ ,... ] ]  
| CERTIFICATE certname  
| ASYMMETRIC KEY asym_key_name
```

<windows_options> ::=

```
DEFAULT_DATABASE = database  
| DEFAULT_LANGUAGE = language
```

Приклад 6.2.

Повторимо попередній приклад, але ім'я користувача вже буде Ivanov, а пароль – qwerty1111. Створимо новий запит:

```
| create login Ivanov with password = 'qwerty'
```

Перевіримо чи правильно спрацював запит (рис. 6.3).

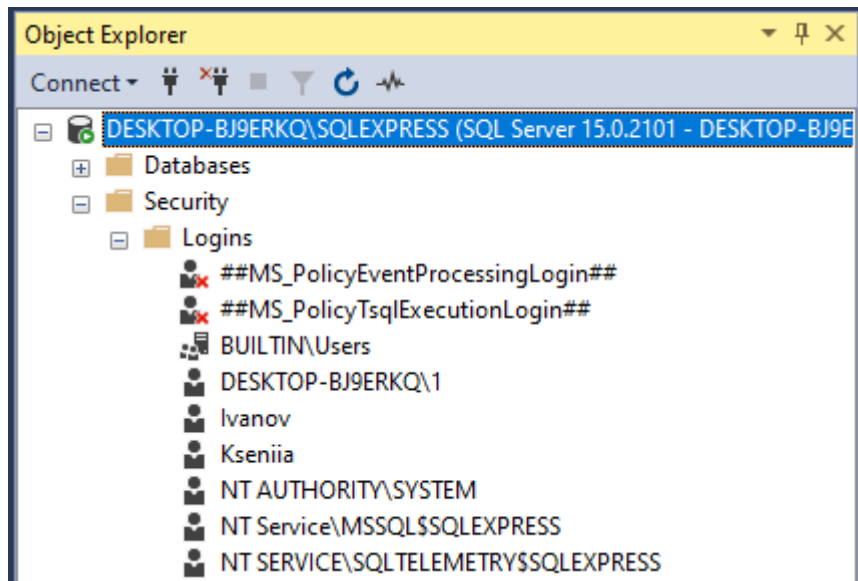


Рис. 6.3 В теці Logins з'явився обліковий запис Ivanov

Запит виконався успішно.

6.3. Створення користувача

Для роботи з користувачами в SQL Server використовують або засоби SQL, або збережені процедури.

Обліковий запис користувача асоціюється з користувачем конкретної бази даних, до якої має доступ., Щоб додати користувача за допомогою збереженої процедури, використовують процедуру `sp_adduser`, яка має такий формат:

```
sp_adduser
[@loginame=] 'обліковий_запис'
[, [@name_in_db=] 'ім'я_користувача']
[, [@grpname=] 'ім'я_ролі'].
```

Приклад 6.3.

Створимо користувача з ім'ям `abc_operator` для облікового запису `Kseniia`. Для рішення цього завдання створимо і виконаємо новий запит:

```
exec sp_adduser 'Kseniia', 'abc_operator'
```

Для створення користувача БД використовується команда CREATE, яка має такий формат:

<https://learn.microsoft.com/en-us/sql/t-sql/statements/create-user-transact-sql?view=sql-server-ver16>

-- Syntax for SQL Server, Azure SQL Database, and Azure SQL Managed Instance

-- Syntax Users based on logins in master

```
CREATE USER user_name
[
    { FOR | FROM } LOGIN login_name
]
[ WITH <limited_options_list> [ ,... ] ]
[ ; ]
```

-- Users that authenticate at the database

```
CREATE USER
{
    windows_principal [ WITH <options_list> [ ,... ] ]

    | user_name WITH PASSWORD = 'password' [ , <options_list> [ ,... ] ]
    | Azure_Active_Directory_principal FROM EXTERNAL PROVIDER
}
[ ; ]
```

-- Users based on Windows principals that connect through Windows group logins

```
CREATE USER
{
```

```

    windows_principal [ { FOR | FROM } LOGIN windows_principal ]
    | user_name { FOR | FROM } LOGIN windows_principal
}
[ WITH <limited_options_list> [ ,... ] ]
[ ; ]

```

-- Users that cannot authenticate

```

CREATE USER user_name
{
    WITHOUT LOGIN [ WITH <limited_options_list> [ ,... ] ]
    | { FOR | FROM } CERTIFICATE cert_name
    | { FOR | FROM } ASYMMETRIC KEY asym_key_name
}
[ ; ]

```

<options_list> ::=

```

DEFAULT_SCHEMA = schema_name
| DEFAULT_LANGUAGE = { NONE | lcid | language name | language alias }
| SID = sid
| ALLOW_ENCRYPTED_VALUE_MODIFICATIONS = [ ON | OFF ] ]

```

<limited_options_list> ::=

```

DEFAULT_SCHEMA = schema_name ]
| ALLOW_ENCRYPTED_VALUE_MODIFICATIONS = [ ON | OFF ] ]

```

-- SQL Database syntax when connected to a federation member

```

CREATE USER user_name
[ ; ]

```

-- Syntax for users based on Azure AD logins for Azure SQL Managed Instance

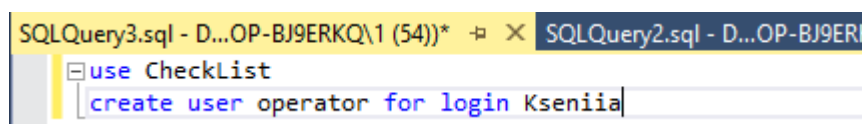
```
CREATE USER user_name  
[ { FOR | FROM } LOGIN login_name ]  
| FROM EXTERNAL PROVIDER  
[ WITH <limited_options_list> [ ,... ] ]  
[ ; ]
```

<limited_options_list> ::=

```
DEFAULT_SCHEMA = schema_name  
| DEFAULT_LANGUAGE = { NONE | Icid | language name | language alias }  
| ALLOW_ENCRYPTED_VALUE_MODIFICATIONS = [ ON | OFF ] ]
```

Приклад 6.4.

Створимо користувача з ім'ям operator для БД CheckList для облікового запису Kseniia. Для рішення цього завдання створимо і виконаємо новий запит:



The screenshot shows a SQL query window titled 'SQLQuery3.sql - D...OP-BJ9ERKQ\1 (54))' with a toolbar containing icons for save, undo, redo, and close. The query text is 'use CheckList' followed by 'create user operator for login Kseniia' on a new line.

Для того щоб перевірити, як він спрацював, у провіднику об'єктів SSMS розкриємо вузол бази даних CheckList, відкриємо гілку Security, а потім теку Users (рис. 6.4).

6.4 Створення ролей

Шляхом створення ролей надаються права і встановлюються обмеження користувачів при роботі з БД. Роль об'єднує користувачів в одну групу. Ролі можна створювати окремо для сервера і окремо для бази даних.

В SQL Server є так звані фіксовані ролі, які не можна видаляти чи редагувати, тобто, не можна змінювати права доступу. Фіксовані ролі також є окремо для сервера (табл. 6.1) і окремо для бази даних (табл. 6.2). Для надання користувачу прав, які мають фіксовані ролі, потрібно його включити у ці ролі.

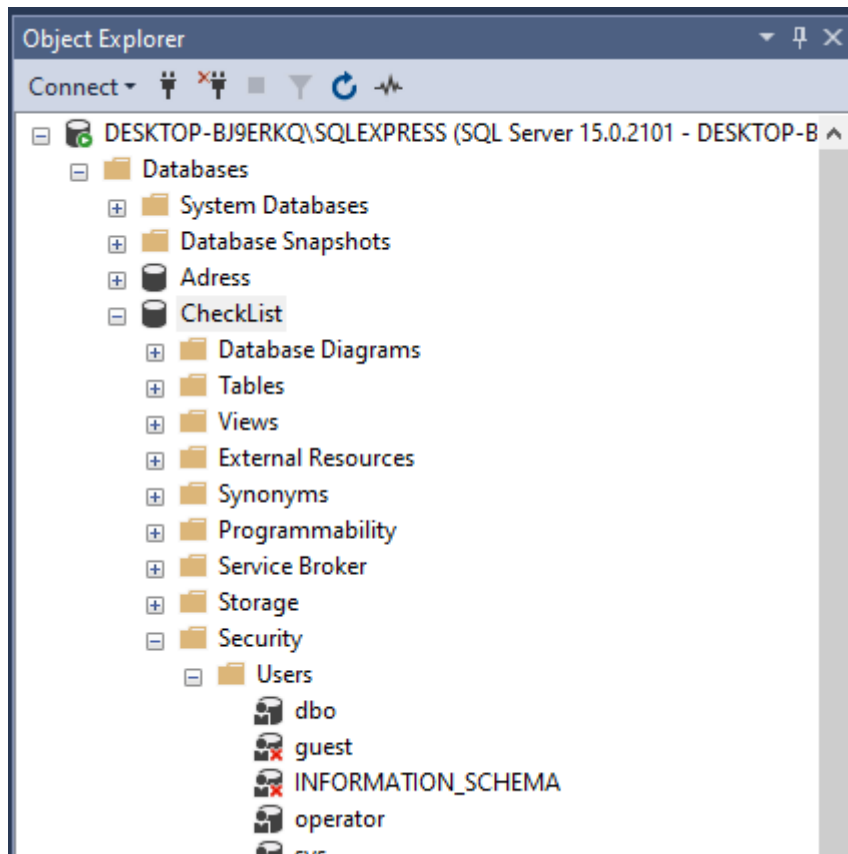


Рис. 6.4 В теці Users з'явився користувач operator

Таблиця 6.1

Фіксовані ролі сервера

Роль	Опис
Sysadmin	Право виконання будь-яких функцій SQL-сервера
Dbcreator	Право на створення бази даних.
Diskadmin	Право на управління файлами на диску.
Processadmin	Право на керування з'єднаннями, запусками і зупинкою екземплярів SQL Server
Securityadmin	Право на створення та управління обліковими записами, право «скинути» пароль облікового запису. Управління дозволами на рівні сервера і на рівні бази даних (при наявності доступу до бази даних).
Serveradmin	Об'єднує можливості ролей diskadmin і processadmin, дає змогу змінювати параметри конфігурації на рівні сервера і

Роль	Опис
	вимикати сервер.
Setupadmin	Право на додавання і видалення пов'язаних серверів.
Public	Кожен обліковий запис має цю роль

Таблиця 6.2

Фіксовані ролі баз даних

Роль	Опис
db_owner	Право повного доступу до бази даних
db_accessadmin	Право додавання і видалення користувачів
public	Роль за умовчанням, наявна в кожній базі даних
db_securityadmin	Право на управління складом ролей
db_backupoperator	Право на можливість створювати резервні копії бази даних.
db_ddladmin	Право на виконання DDL-інструкцій
db_datareader	Право на читання даних з усіх призначених для користувача таблиць, подань і функцій.
db_denydatareader	Заборона на читання даних з усіх призначених для користувача таблиць, подань і функцій.
db_datawriter	Право додавати, видаляти або змінювати дані в усіх призначених для користувача таблицях.
db_denydatawriter	Заборона додавати, змінювати або видаляти дані в призначених для користувача таблицях.

Різні дії по відношенню до ролі здійснюються за допомогою спеціальних процедур (табл. 6.3).

Таблиця 6.3

Системні процедури, які опрацьовують ролі

Функція	Параметри	Призначення
sp_addrole	[@rolename=] 'ім'я_ролі'	створення нової ролі

Функція	Параметри	Призначення
	[, [@ownername=] 'ім'я_власника']	
sp_addrolemember	[@rolename=] 'ім'я_ролі' [@membername=] 'ім'я_користувача'	додавання користувача до ролі
sp_droprolemember	[@rolename=] 'ім'я_ролі' [@membername=] 'ім'я_користувача'	видалення користувача з ролі
sp_droprole	[@rolename=] 'ім'я_ролі'	видалення ролі

Приклад 6.5.

Додамо роль з ім'ям operator_insert, яка надає право створення таблиці:

```
exec sp_addrole operator_insert
grant create table to operator_insert
```

Ролі можливо створювати також за допомогою команди CREATE.

Формат команди такий:

```
CREATE ROLE role_name [ AUTHORIZATION owner_name ]
```

Приклад 6.6.

Створимо обліковий запис для сервера adminIpz і для нього створимо користувача myAdmin. Далі створимо роль MainIpz і надамо їй права для перегляду записів БД, вставки, редагування та видалення. Роль дамо нашому новому користувачу. Виконаємо це за допомогою п'яти запитів:

```
CREATE LOGIN adminIpz
WITH PASSWORD = 'qwerty';

CREATE USER myAdmin
FOR LOGIN adminIpz;

CREATE ROLE MainIpz;

GRANT SELECT, INSERT, UPDATE, DELETE
TO MainIpz;

ALTER ROLE MainIpz
ADD MEMBER myAdmin;
```

Інструкцією GRANT можливо надати користувачеві або ролі деякі права, а інструкцією REVOKE – відкликати вибрані права.

Інструкцією ALTER можливо змінити обліковий запис, користувача чи роль, а інструкцією DROP – видалити їх.

ЗАВДАННЯ

1. Створити обліковий запис для свого сервера.
2. Створити користувача для цього облікового запису.
3. Створити два користувача для своєї БД.
4. Створити дві ролі: одна дає змогу лише переглядати БД, друга – повну маніпуляцію з даними в БД.

Всі запити виконати в Azure Data Studio.

Виконати запити:

Варіант	Опис завдання
1.	1.Створити таблицю дисципліни з атрибутами: Назва дисципліни, викладач, кафедра. Зв'язати її з іншими таблицями. Заповнити її 5 записами. 2. Створити таблицю успішність з полями: Дисципліна, Студент, оцінка. 3. Вивести всі факультети, кафедри та ПІБ студентів, успішність яких перевищує 90 балів
2	1.Вивести ПІБ та посади всіх співробітників супермаркету, ставка яких перевищує 15000 грн. Сортування в алфавітному порядку за ПІБ. 2. Вивести ПІБ співробітників, які зробили продаж певної дати. 3. Вивести всю інформацію про товари, які були продані певної дати.
3.	1. Вивести інформацію про вид військ, якій мав локацію на певній території. 2. Вивести інформацію про військовослужбовців і їх частину.

	3. Вивести інформацію про військовослужбовців, їх вид військ, номер частини і дислокацію.
4.	1. Вивести всю інформацію про бібліотеки, співробітники яких мають телефон, що використовує зв'язок Lifecell . 2. Вивести інформацію про співробітників певної бібліотеки, що поповнювали фонд певної дати. 3. Вивести інформацію про поповнення всіх бібліотек певної дати.
5	1.Вивести всі пансіонати у Києві, в яких є спа-салон. 2. Вивести всю інформацію про пансіонат і тур, де перебував певний клієнт. 3. Вивести інформацію про клієнтів, вік яких більше ніж 20 років.
6	1.3 таблиці маршрути вивести всі назви маршрутів в алфавітному порядку. 2.Вивести інформацію про водіїв, транспорт яких перебуває зараз на ремонті. 3. Вивести інформацію про те, який вид ремонту виконувався за останній місяць.
7	1. Вивести всіх пацієнтів, які лікуються у певному відділенні. 2. Вивести всі діагнози період лікування яких перевищує один тиждень. 3. Вивести всі відділи, які знаходяться на другому поверсі.
8	1.Вивести повну інформацію про пацієнтів та їх хірургів. 2. Вивести інформацію про пацієнтів, операція яких пройшла вдало. 3. Вивести інформацію про пацієнтів та їх операції.
9	1.Вивести всіх читачів бібліотеки початок прізвища яких, розташовані між літерою «С» та літерою «У». 2. Вивести інформацію про читачів та їх улюблені жанри. 3. Вивести всіх читачів певної бібліотеки, які зверталися до неї в останній місяць.
10.	1.Вивести прізвища, ім'я і по-батькові співробітників. Список

	повинен бути відсортований за стажом роботи. 2. Вивести інформацію про автомобілі та їх марки, які були продані за певний період. 3. Вивести інформацію про покупців, що купили автомобілі певної марки.
--	---

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Які рівні захисту SQL Server Ви знаєте?
2. Що таке обліковий запис SQL Server? Його призначення.
3. Що таке користувач в SQL Server? Його призначення.
4. Що таке роль SQL Server? Її призначення.
5. Як надати права для ролі?
6. Як надати права для користувача?
7. Як змінити права облікового запису?
8. Як видалити обліковий запис?
9. Як змінити права користувача?
10. Як видалити користувача?
11. Як змінити роль?
12. Як видалити роль?
13. У чому полягає відмінність між користувачами на рівні сервера і на рівні БД?
14. У чому полягає відмінність між ролями на рівні сервера і на рівні БД?

Лабораторна робота №7

Тема: Групування. Вбудовані функції

План

1. Вбудовані функції;
2. Агрегатні функції;
3. Символьні функції;

4. Числові функції;
5. Часові функції;
6. Запити з угрупованням рядків;
7. Завдання;
8. Питання для самоперевірки.

7.1 Вбудовані функції

Категорії, на які поділяються вбудовані функції SQL, подано в табл. 7.1

Таблиця 7.1

Категорії вбудованих функцій

Функція	Опис
Функції, які повертають набір рядків	Повертають об'єкт, який можна використовувати так само, як табличні посилання в SQL-інструкції.
Агрегатні функції	Обробляють колекцію значень і повертають одне результуюче значення.
Ранжуючі функції	Повертають ранжуючі значення для кожного рядка в секції.
Скалярні функції (опис надається далі)	Обробляють і повертають одиночне значення. Можна застосовувати всюди, де допустимим є вираз.
Функції та типи даних дати і часу	Виконують операції над вихідними значеннями дати і часу, повертають символічні та числові значення, а також дату і час.
Логічні функції	Виконання логічних операцій.
Математичні функції	Виконують обчислення, засновані на числових значеннях, переданих функції у вигляді аргументів, і повертають числові значення.

Функція	Опис
Функції метаданих	Повертають відомості про бази даних та об'єкти баз даних.
Функції безпеки	Повертають дані про користувачів і ролі.
Функції опрацювання символічних даних	Виконують операції над символічним (char або varchar) вхідним значенням і повертають символічне або числове значення.
Системні функції	Виконують операції над значеннями, об'єктами і параметрами примірника SQL Server і повертають відомості про них.
Системні статистичні функції	Повертають статистичні відомості про систему.
Функції обробки тексту та зображень	Виконують операції над текстовими або графічними вхідними значеннями або стовпцями.

7.2 Агрегатні функції

Аргументами агрегатних функцій можуть бути як стовпці таблиць, так і результати виразів над ними. Агрегатні функції і самі можуть включатися в інші арифметичні вирази.

До агрегатних належать такі функції:

- AVG – повертає середнє значення;
- MIN – повертає мінімум;
- CHECKSUM_AGG – повертає контрольну суму значень в групі. Значення NULL не враховуються;
- SUM – повертає суму;
- COUNT – повертає кількість;
- STDEV – повертає середньоквадратичне відхилення;

- COUNT_BIG – повертає кількість елементів у групі;
- STDEVP – повертає статистичне стандартне відхилення всіх значень в зазначеному виразі;
- GROUPING – перевіряє, чи є вказаний вираз стовпця в списку GROUP BY статистичними, чи ні. В результуючому наборі функція GROUPING повертає 1 (статистичний вираз) або нуль (нестатистичний вираз);
- VAR – повертає дисперсію;
- GROUPING_ID – повертає рівень групування;
- VARP – повертає статистичну дисперсію для заповнення всіх значень в зазначеному виразі;
- MAX – повертає максимум.

У стандарті SQL визначені такі види агрегатних функцій: унарні, бінарні, інверсного розподілу, гіпотетичні функції множин.

Загальний формат унарної агрегатної функції такий:

```
Ім'я_функції ([ALL | DISTINCT] вираз) [FILTER (WHERE умова)]
```

Ключове слово DISTINCT вказує, що функція повинна розглядати тільки різні значення аргументу, а ALL – всі значення, включаючи повтори (цей варіант використовується за умовчанням). Ключове слово FILTER дає змогу додатково відібрати рядки таблиці, стовпець яких використовується як аргумент функції.

Агрегатні функції застосовуються в конструкціях SELECT і HAVING.

Розглянемо деякі з агрегатних функцій на прикладах.

Приклад 7.1.

Функція COUNT має два формати. У першому випадку повертається кількість рядків вхідної таблиці, у другому випадку – кількість значень аргументу у вхідній таблиці:

```
COUNT (*)
```

```
COUNT ([DISTINCT | ALL] вираз)
```

Визначимо кількість товару у таблиці Goods:

```
USE [check]
GO

SELECT COUNT(*) As 'Кількість товару' FROM goods
GO
```

Результат:

	Кількість товару
1	8

Приклад 7.2.

Функція SUM підраховує суму значень аргументу для всіх рядків вхідної таблиці. Аргумент повинен мати числовий тип або бути часовим проміжком. Як аргумент може виступати ім'я стовпця або вираз над стовпцями вхідної таблиці. У цій функції також допускається використання ключових слів DISTINCT і ALL.

Визначимо ціну (загальну суму) товару у таблиці Goods:

```
USE [check]
GO

SELECT Sum(Price) As 'Сумарна ціна товару' FROM goods
WHERE Id > 10;
GO
```

Результат:

	Сумарна ціна товару
1	1000

Приклад 7.3.

Функція AVG. Агрегатна функція AVG підраховує середнє значення аргументу для всіх рядків вхідної таблиці. Аргумент повинен мати числовий тип або бути часовим проміжком. Як аргумент може виступати ім'я стовпця або вираз над стовпцями вхідної таблиці. Допускається використовувати ключові слова DISTINCT і ALL.

Визначимо середню ціну товарів у таблиці Goods:

```
USE [check]
GO

SELECT Avg(Price) As 'Середня ціна товару' FROM goods;
GO
```

Результат:

	Середня ціна товару
1	6790,125

Приклад 7.4.

Функції MIN і MAX дають змогу знаходити відповідно мінімальне (MIN) і максимальне (MAX) значення аргументу для всіх рядків вхідної таблиці. Хоча і в них допускається використання ключових слів DISTINCT і ALL, але вони не впливають на результат. Аргумент цих функцій може бути будь-якого типу, для якого визначено впорядкування, тобто числового, символьного і часового.

Визначимо максимальну ціну товарів у таблиці Goods:

```
USE [check]
GO

SELECT Max(Price) As 'Максимальна ціна товару' FROM goods;
GO
```

Результат:

	Максимальна ціна товару
1	12500

7.3 Символьні функції

В MS SQL Server є вбудовані функції для роботи із символьними рядками. Розглянемо деякі з них:

- LEN – повертає кількість символів у рядку без урахування кінцевих прогалів (пробілів);
- UPPER – переводить рядок у верхній регістр;
- LOWER – переводить рядок у нижній регістр;
- CHARINDEX – виконує пошук в рядку вказаного підрядка і повертає початкову позицію підрядка, якщо такий було знайдено, або 0, якщо рядок не містить такого підрядка;
- REPLACE – Замінює всі входження в рядок вказаного підрядка іншим вказаним підрядком:

REPLACE (рядок, шуканий_підрядок, підрядок_на_який_треба_замінити)

- Функції LTRIM, RTRIM і TRIM видаляють символи прогалів або інші вказані символи на початку рядка, в кінці рядка та з обох боків рядка відповідно;
- Функції LEFT і RIGHT повертають вказану кількість символів з початку або з кінця рядка;
- REVERSE – повертає обернений рядок, тобто усі символи початкового рядка будуть переставлені в зворотному порядку.
- CONCAT – повертає один рядок, який поєднує усі рядки-аргументи. Аргументів має бути не менше двох;
- STRING_AGG – повертає один рядок, який поєднує усі рядки-аргументи через рядок-роздільник (separator). Наприкінці рядка роздільник не додається. Аргументів має бути не менше двох;
- STRING_SPLIT – розділяє рядок на підрядки за вказаним рядком-роздільником (separator) і повертає таблицю з одним стовпцем, рядки якого є цими підрядками.

Приклад 7.5.

Отримаємо назви у верхньому регістрі всіх товарів з таблиці Goods:

```
USE [check]
GO

SELECT UPPER(Name) FROM goods;
GO
```

Результат:

1	ПАЛЬТО ЖІН. СИНЄ
2	ПАЛЬТО ЖІН. ЗИМНЄ, ДЕМІСІЗОННЕ
3	ПАЛЬТО ЖІН. ЧОРНЕ
4	ПАЛЬТО УНІСЕКС. ЧОРНЕ
5	ПАЛЬТО ЧОЛ. КЛІТИНКА
6	ПАЛЬТО ЧОЛ. ЧОРНЕ
7	BALL
8	BAG

7.4 Числові функції

Числові (або математичні) функції виконують обчислення з використанням заданих числових аргументів, і повертають числові значення.

Стандарт SQL пропонує ряд числових функцій з очевидною семантикою. Частина функцій перераховано нижче:

- ABS – повертає абсолютне значення аргументу;
- DEGREES – повертає для значення кута в радіанах відповідне значення в градусах;
- RAND – повертає псевдовипадкове значення типу float від 0 до 1;
- EXP – повертає експоненту від першого аргументу, тобто число e , піднесене до заданого степеня, який вказано як другий аргумент;
- ROUND – повертає числове значення першого аргументу, округлене до зазначеної довжини або точності, яку вказано як другий аргумент;
- FLOOR – повертає найбільше ціле число, менше або рівне вказаному числовому значенню аргументу;
- LOG – натуральний логарифм аргументу;
- LOG10 – десятковий логарифм аргументу;
- SIN – синус аргументу;
- SQRT – корінь квадратний аргументу;
- PI – число 3.14;
- SQUARE – повертає квадрат аргументу;
- POWER – повертає значення першого аргументу, піднесене до заданого степеня, який вказано як другий аргумент;
- TAN – тангенс аргументу;

7.5. Часові функції

Ці функції використовують як аргумент типи даних дати, часу, часової позначки або часового проміжку. Тип значення, що повертається не завжди відповідає типу аргументу.

Функції дати і часу в SQL поділяються на:

- Функції, які отримують значення системної дати і часу;
- Функції, які отримують компоненти дати і часу;
- Функції, які отримують значення дати і часу з їх компонентів;
- Функції, які отримують різницю дати і часу;
- Функції, які змінюють значення дати і часу;
- Функції, які встановлюють або отримують формат сеансу;
- Функції, що перевіряють дату й час.

Приклад 7.6.

Функція GETDATE повертає поточну локальну дату і час на основі системного годинника у вигляді об'єкта datetime:

```
USE [check]
GO

SELECT GETDATE() |
```

Результат:

1	2021-10-19 20:20:23.327
---	-------------------------

Приклад 7.7.

Функція DAY повертає день з дати, яка передається як аргумент. Аналогічно функція MONTH повертає місяць дати, а функція YEAR повертає рік з дати.

Виведемо з поточної дати тільки день:

```
USE [check]
GO

SELECT Day(GETDATE() )|
```

Приклад 7.8.

Функція DATENAME: повертає частину цієї дати у вигляді рядка. Назва потрібної частини дати передається першим аргументом, а сама дата передається другим аргументом:

```
USE [check]
GO

|
SELECT DATENAME(month, GETDATE())
```

Приклад 7.9.

Функція DATEPART повертає частину цієї дати у вигляді числа. Назва потрібної частини дати передається першим аргументом (як і для DATENAME), а сама дата передається другим аргументом.

```
USE [check]
GO

SELECT DATEPART(month, GETDATE())
```

Приклад 6.10.

Функція DATEADD повертає дату, яка є результатом складання числа до певного компоненту дати. Першим аргументом передається компонент дати, другим – кількість, яка додається, третім – сама дата, до якої треба додати значення другого аргументу:

```
SELECT DATEADD(month, 4, Getdate())
SELECT DATEADD(day, 8, Getdate())
SELECT DATEADD(day, -3, Getdate())
```

Приклад 7.11.

Функція DATEDIFF повертає різницю між двома датами. Першим аргументом передається компонент дати, який вказує, в яких одиницях варто вимірювати різницю. Другим і третім аргументами передаються дати, які потрібно порівняти:

```
USE [check]
GO

SELECT DATEDIFF(year, Getdate(), '2021-9-1')
SELECT DATEDIFF(month, Getdate(), '2021-9-1')
SELECT DATEDIFF(day, Getdate(), '2021-9-1')
```

Приклад 7.12.

Функція EOMONTH повертає дату останнього дня того місяця, довільну дату якого передано як аргумент:

```
USE [check]
GO

SELECT EOMONTH('2021-10-22')
```

Приклад 7.13.

Функція DATEFROMPARTS за роком, місяцем і дня створює дату:

```
SELECT DATEFROMPARTS(2021, 10, 22)
```

7.6 Запити з угрупованням рядків

Часто під час створення звітів з'являється потреба у формуванні проміжних вибірок з тих записів, які мають якісь спільні ознаки. Саме для цього призначена конструкція GROUP BY. Вона дає змогу всю множину рядків таблиці розділити на групи за ознакою рівності значень одного або декількох стовпців (і виразів над ними).

Фраза GROUP BY повинна розташовуватися після фрази WHERE (якщо вона відсутня, то після фрази FROM). Загальний синтаксис фрази GROUP BY такий:

```
SELECT список_стовпців FROM ім'я_таблиці  
[WHERE <умова_пошуку>]  
GROUP BY вираз [, вираз ...]
```

За наявності фрази GROUP BY фраза SELECT застосовується до кожної групи, сформованої фразою угруповання. В цьому випадку і дія агрегатних функцій, зазначених у фразі SELECT, буде поширюватися не на всю результуючу таблицю, а тільки на рядки в межах кожної групи. Кожен вираз в списку фрази SELECT має приймати єдине значення для групи, тобто воно може бути:

- константою;
- агрегатною функцією, яка оперує всіма значеннями аргументами в межах групи і поєднує їх в одне значення (наприклад, в суму);
- виразом, ідентичним тому, який стоїть у фразі GROUP BY;
- виразом, який об'єднує наведені вище варіанти.

Приклад 7.14.

Візьмемо таблицю goods бази даних check (рис. 7.1).

DESKTOP-AA2MEC1\...heck - dbo.goods X SQLQuery7.sql - (l...			
	Id	Name	Price
	1	Пальто жін. си...	12321
	2	Пальто жін. зи...	4500
	3	Пальто жін. чо...	12400
	4	Пальто унісекс...	5000
	5	Пальто чол. кл...	12500
	6	Пальто чол. ч...	6500
	10	Ball	100
	11	Bag	1000
	13	Ball	50
»*	NULL	NULL	NULL

Рис. 7.1 Таблиця goods бази даних check

Згрупуємо товари за назвою і порахуємо їх кількість:

```
USE [check]
GO
```

```
SELECT Name,
COUNT(*)
FROM goods
GROUP BY Name;
```

Результат:

	Name	
1	Bag	1
2	Ball	2
3	Пальто жін. зимнє, демісизонне	1
4	Пальто жін. синє	1
5	Пальто жін. чорне	1
6	Пальто унісекс. чорне	1
7	Пальто чол. клітинка	1
8	Пальто чол. чорне	1

Приклад 7.15.

Групування за стовпцями. Візьмемо таблицю Person (рис. 7.2), в якій зберігається інформація про фізичних осіб: id, ім'я, електронна скринька, телефон.

	id	Name	email	telefon
►	1	Іванов Іван	ivanov@ukr.net	0671947186
	2	Петров Петро	petrov@ukr.net	0733211466
	3	Вагнер Дмитро	NULL	NULL
	4	Ільченко Ната...	NULL	NULL
	10011	Іванов І.П.	abs.gmail.com	0675144145
	10012	Петров О.О.	qwerty.gmail.c...	0506137478
	10013	Кравченко К.К.	travnik.gmail.c...	0506137478
	10014	Мартинюк С.Л.	rav.ukr.net	0735137575
	10015	Синиця О.В.	nirvana.gmail.c...	0666337338
	10016	Висоцький М.В.	ratusha.ukr.net	0962141566

Рис. 7.2 Таблиця Person бази даних check

І візьмемо таблицю Employee (рис. 7.3), в якій зберігається інформація про співробітників: табельний номер, id із таблиці Person, відсоток від продажу, який отримують ці співробітники.

	Tab	Personal	Interest
	1	1	5
	2	2	10
	3	10016	5
►*	NULL	NULL	NULL

Рис. 7.3 Таблиця Employee бази даних check

Визначимо кількість співробітників, які отримують певний відсоток:

```
SELECT count (*) AS "К-сть працівників"
FROM Person p, Employee e
WHERE e.Personal = p.id
GROUP BY e.Interest;
```

Результат:

	К-сть працівників
1	2
2	1

Фраза WHERE перевіряє на відповідність умові кожного рядка вихідних таблиць. Для відбору рядків серед отриманих груп слід застосовувати фразу HAVING. Вона відіграє таку ж роль для груп, що і фраза WHERE для вихідних таблиць, і може використовуватися лише за наявності фрази GROUP BY. У

реченні SELECT фрази WHERE, GROUP BY і HAVING обробляються в такому порядку:

- фразою WHERE відбираються рядки, які задовольняють вказаній в ній умові;
- фраза GROUP BY групує відібрані рядки;
- фразою HAVING відбираються групи, що задовольняють вказаній в ній умові.

Приклад 7.16.

Виконаємо попередній приклад, але рахувати будемо тільки тих співробітників, у яких відсоток від продажу більше ніж 5.

```
USE [check]  
GO
```

```
SELECT count (*) AS "К-сть працівників"  
FROM Person p, Employee e  
WHERE e.Personal = p.id  
GROUP BY e.Interest  
having e.Interest > 5;
```

Результат:

	К-сть працівників
1	1

ЗАВДАННЯ

Варіант	Опис завдання
1	1. Розрахувати середню оцінку студентів. 2. Вивести оцінки по всім дисциплінам всіх студентів. 3. Вивести всіх студентів, які за певним предметом, отримали певну оцінку. 4. Вивести всіх студентів максимальна оцінка яких не більша ніж 60 балів.
2	1. Вивести інформацію про співробітника, який зробив продаж на максимальну суму за останній місяць. 2. Вивести кількість посад, сума окладу яких більша ніж 5000.

	3. Згрупувати всі товари за виробником. 4. Вивести всі товари певного виробника.
3	1. Знайти найменшу площу дислокації частини. 2. Обрахувати кількість частин, площа дислокації яких більша ніж 10 000 м². 3. Згрупувати весь особовий склад за ротами. 4. Вивести особовий склад певної роти.
4	1. Знайти середню кількість книжок, журналів тощо для всіх фондів. 2. Порахувати кількість літератури за певним типом. 3. Згрупувати співробітників за бібліотеками. 4. Вивести інформацію про фонд певної бібліотеки.
5	1. Знайти різницю між датами заїзду і від'їзду клієнтів певного пансіонату. 2. Вивести кількість пансіонатів, які знаходяться в певному місті. 3. Вивести інформацію про пансіонати, згруповані на стовпчиком Місто. 4. Вивести всі пансіонати, які знаходяться в певному місті.
6	1. Порахувати кількість маршрутів в господарстві. 2. Порахувати загальний пробіг всіх автобусів. 3. Згрупувати транспорт за назвою. 4. Вивести інформацію про транспорт для певного маршруту.
7	1. Порахувати, скільки днів прохворів певний пацієнт з певною хворобою. 2. Порахувати загальну кількість пацієнтів, згруповану за віком. 3. Вивести всіх пацієнтів, згрупованих за певною хворобою і відсортованих за віком. 4. Вивести всіх пацієнтів певного лікаря, які лікувалися в певний період.
8	1. Порахувати кількість операцій, які відбулися в певну дату. 2. Порахувати кількість лікарів, які мають звання лікар-

	спеціаліст. 3. Вивести інформацію про всіх пацієнтів, яких оперував певний хірург у вказану дату. 4. Вивести ПІБ всіх лікарів, які мають звання лікар-спеціаліст.
9	1. Порахувати всіх читачів бібліотеки. 2. Згрупувати читачів (порахувати їх кількість) за жанром літератури. 3. Вивести всіх читачів певного жанру літератури. 4. Вивести інформацію про літературу, яку читав певний читач.
10	1. Порахувати середню ціну автомобілів. 2. Порахувати середню ціну автомобілів певної марки. 3. Вивести всі автомобілі певної марки. 4. Вивести всі відомості про автомобілі певної марки, які продані певним співробітником за вказаний період часу, наприклад, з 01.08.2022 по 31.08.2022.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Які можна виділити категорії вбудованих функцій SQL?
2. Назвіть приклади агрегатних функцій.
3. Назвіть функції для роботи з символьною інформацією.
4. Назвіть функції для роботи з датами.
5. Що таке угруповання?
6. Назвіть етапи угруповання?
7. Назвіть особливості застосування оператора HAVING.

Лабораторна робота №8

Тема: Основи програмування за допомогою вбудованої мови SQL.

Подання

План

1. Арифметичні і логічні операції;
2. Ідентифікатори та змінні;
3. Операторські дужки;
4. Умовна конструкція IF;
5. Цикл WHILE;
6. Функції і збережені процедури;
7. Подання;
8. Завдання;
9. Питання для самоперевірки.

8.1 Арифметичні і логічні операції

Операнди, які застосовуються в арифметичних і логічних операціях в SQL, подано в табл. 8.1

Таблиця 8.1

Операнди SQL

Знак	Призначення	Знак	Призначення
*	Знак множення	“ “	В них заключають строкові значення,
-	Знак віднімання	‘ ‘	В них заключають строкові значення,
%	Залишок від ділення двох чисел	<>	Не дорівнює

Знак	Призначення	Знак	Призначення
+	Знак додавання або конкатенації (об'єднання двох рядків)	[]	Аналог лапок, в них можна розміщувати назви ідентифікаторів, якщо в їх назвах є прогалини (пробіли)
=	Знак рівності	!<	Не менше, ніж
<=	Менше або дорівнює	!>	Не більше, ніж
>=	Більше або дорівнює	<	Більше
!=	Не дорівнює	>	Менше
@	Ставиться перед іменем змінної	.	Розділяє батьківські і підлеглі об'єкти
@@	Вказує на системні функції	/	Знак ділення
--	Однорядковий коментар або коментар з поточної позиції і до кінця рядка	/* */	Багаторядковий коментар

8.2 Ідентифікатори та змінні

Ідентифікатори – це імена об'єктів, на які можна посилатися в програмі. Найменування об'єктів відбувається за такими правилами:

- перший символ складається з букв англійського алфавіту або "#", "@";
- решта додатково з цифр і «\$»;
- ім'я ідентифікатора не повинно збігатися з зарезервованим словом.

Для збереження розрахованих проміжних значень використовуються змінні, ім'я яких повинно починатися з @.

Синтаксис в позначеннях MS SQL Server:

```
DECLARE @ ім'я_змінної1 тип_змінної (розмір),, @
ім'я_змінноїN тип_змінної (розмір)
```

Приклад 8.1.

```
DECLARE @a INT, @b numeric (10,2)
DECLARE @str CHAR (20)
```

Присвоєння значень змінним відбувається за допомогою оператора **SET**, синтаксис:

SET @ ім'я_змінної = значення

Приклад 8.2.

```
DECLARE @a INT, @b numeric (10,2)
SET @a = 20
SET @b = (@a + @a) / 15
SELECT @b --вивід на екран результату
```

Результат:

1	2.00
---	------

Якщо змінній присвоюємо значення результату запиту, то використовуємо оператор **SELECT**.

Приклад 8.3.

Присвоїмо змінній **a** кількість рядків в таблиці **goods**:

```
use [check]
DECLARE @a INT
SELECT @a = COUNT (*) FROM goods
SELECT @a
```

Результат:

1	10
---	----

Те саме можна отримати іншим способом:

```
use [check]
DECLARE @a INT
SET @a = (SELECT COUNT (*) FROM goods)
SELECT @a
```

Результат:

1	10
---	----

Приклад 8.4.

Присвоїмо змінній str значення стовпчика Name з таблиці goods:

```
use [check]
DECLARE @str CHAR (30)
SELECT @str = Name FROM goods
SELECT @str
```

Результат:

1	Galaxy S7
---	-----------

Створення тимчасової таблиці відбувається через змінну типу TABLE.

Змінна для тимчасової таблиці оголошується за допомогою оператора DECLARE із зазначенням в дужках стовпців таблиці, їх типів, розмірів, значень за замовчуванням, а також індексів типу PRIMARY KEY або UNIQUE.

Приклад 8.5.

Створимо таблицю і додаємо в неї два рядочки:

```
use [check]
DECLARE @mytable TABLE (id INT, myname CHAR (20) DEFAULT 'Іванов Іван')
INSERT INTO @mytable (id) VALUES (1)
INSERT INTO @mytable (id, myname) VALUES (2, 'Ігор Троцький')
SELECT * FROM @mytable
```

Результат:

	id	myname
1	1	Іванов Іван
2	2	Ігор Троцький

Приклад 8.6.

Створення тимчасової таблиці з двома полями, в яку буде додано рядки, як результат виконання запиту вибірки даних з таблиці goods:

```
use [check]
DECLARE @mytable TABLE (id INT, myname CHAR (255) DEFAULT 'Введіть назву')
INSERT @mytable SELECT id, Name FROM goods
SELECT * FROM @mytable
```

Результат:

	id	myname
1	1	Пальто жін. синє
2	2	Пальто жін. зимнє, демісизонне
3	3	Пальто жін. чорне
4	4	Пальто унісекс. чорне
5	5	Пальто чол. клітинка
6	6	Пальто чол. чорне
7	10	Ball
8	11	Bag
9	13	Ball
10	1013	Galaxy S7

8.3 Операторські дужки

Для конструкцій умовних і циклічних розгалужень використовують операторські дужки. Операторські дужки повинні містити хоча б один оператор. Операторські дужки мають такий вид:

```
BEGIN -- це відкриття операторської дужки
оператор1
оператор2
...
операторN
END -- це закриття операторської дужки
```

8.4 Умовна конструкція IF

Синтаксис умовної конструкції в SQL має вигляд:

```
IF умова
набір_операторів_1 ELSE
набір_операторів_2
```

Приклад 8.7.

Використаємо операторські дужки і умовні конструкції для пошуку відповіді на питання: «кількість товарів більше 10, так чи ні?». Для цього напишемо і виконаємо такий запит:

```

DECLARE @a INT
DECLARE @str CHAR (30)
SET @a = (SELECT COUNT (*) FROM goods)
IF @a > 10 BEGIN
    SET @str = 'Кількість товарів більше 10'
    SELECT @str
END ELSE
BEGIN
    SET @str = 'Кількість товарів =' + str (@a)
    SELECT @str
END

```

Результат:

1	Кількість товарів = 10
---	------------------------

8.5 Цикл WHILE

Синтаксис циклу в SQL має вигляд:

```

WHILE Умова
набір_операторів_1
BREAK
набір_операторів_2
CONTINUE

```

Конструкції **BREAK** і **CONTINUE** є необов'язковою. Цикл можна примусово зупинити, якщо в його тілі виконати команду **BREAK**. Якщо ж потрібно почати цикл заново, не чекаючи виконання всіх команд в тілі, необхідно виконати команду **CONTINUE**.

Приклад 8.8.

Оголосимо змінну *a*. Змінна *a* в циклі збільшує своє значення у випадковому порядку. Перевіримо, щоб значення *a* не перевищувало 100. Також напишемо умову примусового виходу з циклу, якщо значення змінної *a* потрапляє в діапазон від 40 до 50.

```

DECLARE @a INT
SET @a = 1
WHILE @a < 100
BEGIN
    PRINT @a --| вивід на екран значення змінної
    IF (@a > 40) AND (@a < 50)
        BREAK --вихід і виконання 1-ї команди за циклом
    ELSE
        SET @a = @a + rand() * 10
    CONTINUE
END
PRINT @a

```

Результат:

```

1
2
10
18
18
22
24
26
33
35
41
41

```

З результату видно, що спрацювала умова примусового виходу з циклу, як значення змінної а стало рівним 41.

8.6 Функції і збережені процедури

Функції і збережені процедури використовуються в SQL Server для реалізації складних поширених алгоритмів обробки даних або різних адміністративних дій створення облікових записів, отримання інформації про об'єкти бази даних, управління властивостями сервера і баз даних, управління підсистемою реплікації і автоматизації тощо.

Функції і процедури можуть бути викликані клієнтськими програмами, іншими функціями або збереженими процедурами, а також тригерами.

У будь-якому випадку необхідно вказати ім'я функції або процедури, що зберігається, і список аргументів. Аргументи зіставляються параметрам відповідної функції або процедури, яка зберігається. Водночас типи аргументів

і параметрів повинні збігатися або допускати автоматичні перетворення типів. Якщо для деякого параметра встановлено значення за замовчуванням і це значення підходить для даного виклику, то відповідний аргумент може бути опущений.

Створення, зміна та видалення функцій і процедур проходить за допомогою таких команд:

- CREATE FUNCTION,
- CREATE PROCEDURE,
- ALTER FUNCTION,
- ALTER PROCEDURE,
- DROP FUNCTION,
- DROP PROCEDURE.

При створенні функції вказується тип значення, а в тілі функції обов'язково задається команда **RETURN**, за якою слідує вираз для обчислення значення, яке повертається.

У **тілі процедури** використання команди **RETURN** (зазвичай, без подальшого виразу) не обов'язково. Коли цієї команди немає, вихід з процедури відбуватиметься після виконання останньої команди процедури.

Тіло, як функції, так і збереженої процедури, починається ключовим словом **AS**. Так як кожна з них зберігається як окремий об'єкт, то для того, щоб вказати кінець тіла, не потрібно записувати якесь спеціальне ключове слово або знак.

За командами створення функції або процедури перераховуються імена параметрів, що починаються з символу **@**, і їх типи, а також значення за замовчуванням.

Для *функції* цей список записується в круглих дужках, після яких записується ключове слово **RETURNS** (повернення) і тип значення, що повертається.

Для *збереженої процедури*, круглі дужки не використовуються, і задавати тип значення, яке повертається, не потрібно.

Для тіла функції часто використовують ключове слово **begin** після ключового слова **as** і ключове слово **end** в кінці тіла.

Додаткові опції функції або процедури, які задаються ключовим словом **with** до початку тіла.

У SQL Server можна створювати **функції трьох класів**:

- **Scalar** – повертають звичайне скалярне значення;
- **Inline** – повертають таблицю динамічної структури, яка створюється єдиною командою тіла функції **SELECT**;
- **Multi-statement** – повертає звичайну таблицю заданої структури, водночас кількість команд в тілі функції не обмежується.

Створення функцій відбувається за такою командою:

CREATE FUNCTION (*)**,

яка створює **визначену користувачем функцію** в SQL Server. Обумовлена користувачем функція є підпрограмою **SQL** або середовища CLR, яка приймає параметри, виконує дії, такі як складні обчислення, а потім повертає результат цих дій у вигляді значення. Значення, яке повертається, може бути скалярним значенням або таблицею. За допомогою цієї інструкції можна створити підпрограму, яку можна повторно використовувати такими способами:

- В інструкціях Transact-SQL, наприклад **SELECT**.
- У застосунках, які викликають функцію.
- У визначенні іншої користувацької функції.
- Для параметризації уявлення або поліпшення функціональності індексованого подання.
- Для визначення стовпця таблиці.
- Для визначення обмеження **CHECK** на стовпець.
- Для заміни збереженої процедури.

Синтаксис створення скалярної функції має такий вид:

```

--Transact-SQL Scalar Function Syntax
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }
  [ ,...n ]
]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END
[ ; ]

```

Аргументи:

- **TABLE** – вказує, що значенням функції, яке повертається, є таблиця. Функціям з табличним значенням можуть передаватися тільки константи і @local_variables.
- У вбудованих функціях з табличним значенням значення TABLE, яке повертається, визначається за допомогою єдиної інструкції **SELECT**. Вбудовані функції не мають відповідних змінних, які повертаються.
- **select_stmt** - Одиночна інструкція **SELECT**, що визначає значення, яке повертається вбудованою функцією з табличним значенням.

У функціях припустимі такі інструкції:

- Інструкції присвоювання.
- Інструкції управління потоком, за винятком інструкцій **TRY ... CATCH**.
- Інструкції **DECLARE**, які оголошують локальні змінні і локальні курсори.
- Інструкції **SELECT**, які містять списки вибору з виразами, які привласнюють значення локальних змінних.
- Операції над локальними курсорами, які оголошуються, відкриваються, закриваються і звільняються в тілі функції. Допустимі тільки ті інструкції **FETCH**, які пропозицією **INTO** привласнюють значення локальним змінним. Інструкції **FETCH**, які повертають дані клієнта, неприпустимі.

- Інструкції **INSERT**, **UPDATE** і **DELETE**, які змінюють локальні табличні змінні.
- Інструкції **EXECUTE**, які викликають розширені процедури.

Обмеження:

- Визначені користувачем функції не можуть виконувати дії, які змінюють стан бази даних.
- Визначені користувачем функції не можуть містити конструкцію **OUTPUT INTO**, ціллю якої є таблиця.
- Визначені користувачем функції можуть бути **вкладеними**, тобто з однієї функції може бути викликана інша. Вкладеність визначених користувачем функцій не може перевищувати 32 рівнів.

Приклад 8.9.

У цьому прикладі показано створення скалярної визначеної користувачем функції **ISOweek**, яка отримує в якості аргументу дату і обчислює номер місяця, а потім за номером визначає назву місяця. Для правильної роботи цієї функції перед її викликом повинна бути виконана інструкція **SET DATEFIRST 1**.

У цьому прикладі також показано використання пропозиції **EXECUTE AS** для вказання контексту безпеки, в якому може бути виконана збережена процедура. Параметр **CALLER** вказує, що процедура буде виконана в контексті користувача, який її викликає. Перш ніж функцію буде створено, виконується перевірка на наявність в базі даних функції з таким самим ім'ям. Якщо функція з таким ім'ям вже була створена раніше, то вона буде видалена за допомогою команди **DROP FUNCTION** і створена заново **CREATE FUNCTION**.

Запит:

```

USE [check];
GO
IF OBJECT_ID (N'dbo.ISOweek', N'FN') IS NOT NULL
    DROP FUNCTION dbo.ISOweek;
GO
CREATE FUNCTION dbo.ISOweek (@DATE date)
RETURNS CHAR(15)
WITH EXECUTE AS CALLER
AS
BEGIN
    DECLARE @man int;
    DECLARE @ISOweek char(15);
    SET @man= MONTH(@DATE)
    IF (@man=1) SET @ISOweek='Січень';
    IF (@man=2) SET @ISOweek='Лютий';
    IF (@man=3) SET @ISOweek='Жовтень';
    IF (@man=4) SET @ISOweek='Квітень';
    IF (@man=5) SET @ISOweek='Травень';
    IF (@man=6) SET @ISOweek='Червень';
    IF (@man=7) SET @ISOweek='Липень';
    IF (@man=8) SET @ISOweek='Серпень';
    IF (@man=9) SET @ISOweek='Вересень';
    IF (@man=10) SET @ISOweek='Жовтень';
    IF (@man=11) SET @ISOweek='Листопад';
    IF (@man=12) SET @ISOweek='Грудень';
    RETURN(@ISOweek);
END
GO
SET DATEFIRST 1;
SELECT dbo.ISOweek('12.04.2004') AS 'Місяць';

```

Результат:

	Місяць
1	Квітень

Приклад 8.10.

Створимо функцію з табличним значенням, яка буде повертати назви та ціни товарів, ціна яких більша за вказану. Застосуємо цю функцію, щоб з таблиці товарів отримати усі товари, ціна яких більша за 5000:

```

USE [check];
GO

IF OBJECT_ID (N'dbo.PriceByStore', N'FN') IS NOT NULL
DROP FUNCTION dbo.PriceByStore;
GO

CREATE FUNCTION PriceByStore(@storeid int)
RETURNS TABLE
AS
RETURN
(
SELECT Name AS "Good", Price From goods
WHERE Price>@storeid
);
GO

SET DATEFIRST 1;
SELECT * FROM dbo.PriceByStore(5000) ;

```

Результат:

	Good	Price
1	Пальто жін. чорне	12400
2	Пальто чол. клітинка	12500
3	Пальто чол. чорне	6500

Збережені процедури – це набір команд, які складаються з одного або декількох операторів SQL або функцій і зберігаються в базі даних у відкомпільованому вигляді.

Типи збережених процедур бувають такі:

- **Системні збережені процедури** призначені для виконання різних адміністративних дій. Практично всі дії з адміністрування сервера виконуються з їх допомогою.
- **Користувацькі збережені процедури** реалізують ті чи інші дії. Збережені процедури – повноцінний об’єкт бази даних. Внаслідок цього кожна збережена процедура розташовується в конкретній базі даних, де і виконується.
- **Тимчасові збережені процедури** існують лише деякий час, після чого автоматично знищуються сервером. Вони діляться на локальні і глобальні.

Синтаксис оператора створення нової або зміни існуючої збереженої процедури в MS SQL Server:

```
--Transact-SQL Stored Procedure Syntax
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
      [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY] ]
    ] [ ,...n ]
    [ WITH <procedure_option> [ ,...n ] ]
    [ FOR REPLICATION ]
    AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
    [;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

Для виконання збереженої процедури використовується команда:

```
[[EXEC [UTE] ім'я_процедури [; номер]
[[@ ім'я_параметра =] {значення | @ ім'я_змінної}
[OUTPUT] | [DEFAULT]] [, ... n]
```

Приклад 8.11.

Процедура без параметрів:

```
USE [check];
GO

CREATE PROCEDURE Count_Goods
AS
Select count (Name)
from goods
where Price > 5000
Go

GO
EXECUTE Count_Goods
```

Результат:

1	3
---	---

Приклад 8.12.

Процедура з параметром:

```

USE [check];
GO

CREATE PROCEDURE Count_Goods1
@Sum_price as Int
AS
Select count (Name)
from goods
where Price > @Sum_price
Go

GO

EXECUTE Count_Goods1 10000 |

```

Результат:

	(Отсутствует имя столбца)
1	2

8.7 Подання

Подання (View) для користувачів баз даних виглядає як таблиця, проте воно не містить даних, а лише надає дані, розташовані в одній або декількох таблицях. **Таким чином, подання – це віртуальні таблиці, які визначаються запитом на мові Transact-SQL.** Подібно реальним таблицям, подання містять іменовані стовпці і рядки з даними, які динамічно вибираються з таблиць і надаються користувачу для перегляду. Подання часто застосовуються для обмеження доступу до конфіденційних даних в таблицях баз даних.

Коли в подання не включається стовпець вихідної таблиці, то вважають, що на таблицю накладений вертикальний фільтр. Якщо в SQL-запиті встановлено одну або декілька умов для вибірки рядків, то вважають, що на таблицю накладений горизонтальний фільтр.

Подання може вибирати дані з іншого подання, які, в свою чергу, можуть також ґрунтуватися на поданнях або таблицях. Вкладеність подання (глибина) не повинна перевищувати 32 рівень. Подання можна створювати, використовуючи бази даних одного сервера (поточного). Максимальна кількість стовпців в одному поданні 1024. Подання не можуть посилатися на тимчасові таблиці. Крім того, не можна створювати тимчасове подання.

Для подання не можна визначати обмеження цілісності, тригери, правила, або замовчування, а також створювати звичайний або повнотекстовий індекс. В основному подання використовуються для вибірки даних.

Однак за допомогою подання можна змінювати дані в таблицях, на основі яких побудовано ці подання, при цьому необхідно дотримуватися ряду правил: подання має містити, як мінімум, одну таблицю в параметрі FROM команди SELECT, не дозволяється використання функцій агрегування та інших.

Як і для таблиць, для подання можна визначити такі права доступу:

- SELECT – перегляд даних;
- INSERT – додавання даних через подання;
- UPDATE – зміна даних у вихідних таблицях;
- DELETE – видалення даних у вихідних таблицях.

Щоб мати можливість створювати подання, треба мати права власника баз даних і мати відповідні дозволи для будь-яких таблиць або подання, згаданих "в запиті на створення цього подання.

Для створення подання використовується така команда Transact-SQL:

```
CREATE VIEW [Ім'я_бази_даних] [Ім'я_власника]  
Ім'я_подання  
[(Ім'я_колонки [... n])]  
[WITH{ENCRYPTION|SCHEMABINDING|  
VIEW_METADATA}  
AS Команда SELECT  
[WITH CHECK OPTION]
```

Якщо в команді не задані імена колонок подання, то вони визначаються по іменах вибраних колонок в команді SELECT. Параметр ENCRYPTION приховує код створення цього подання, а параметр SCHEMABINDING забезпечує контроль структури вихідних об'єктів, до яких звертається оператор SELECT. Опція WITH CHECKOPTION не дає змогу змінювати рядки таким чином, щоб вони зникли під час вибирання командою SELECT.

Для того, щоб створити подання засобами MS SQL Server Management Studio потрібно у провіднику об'єктів розкрити вузол бази даних, натиснути на

його підвузол Подання (View) правою кнопкою миші і в контексті меню вибрати пункт «Створити подання» (рис. 8.1).

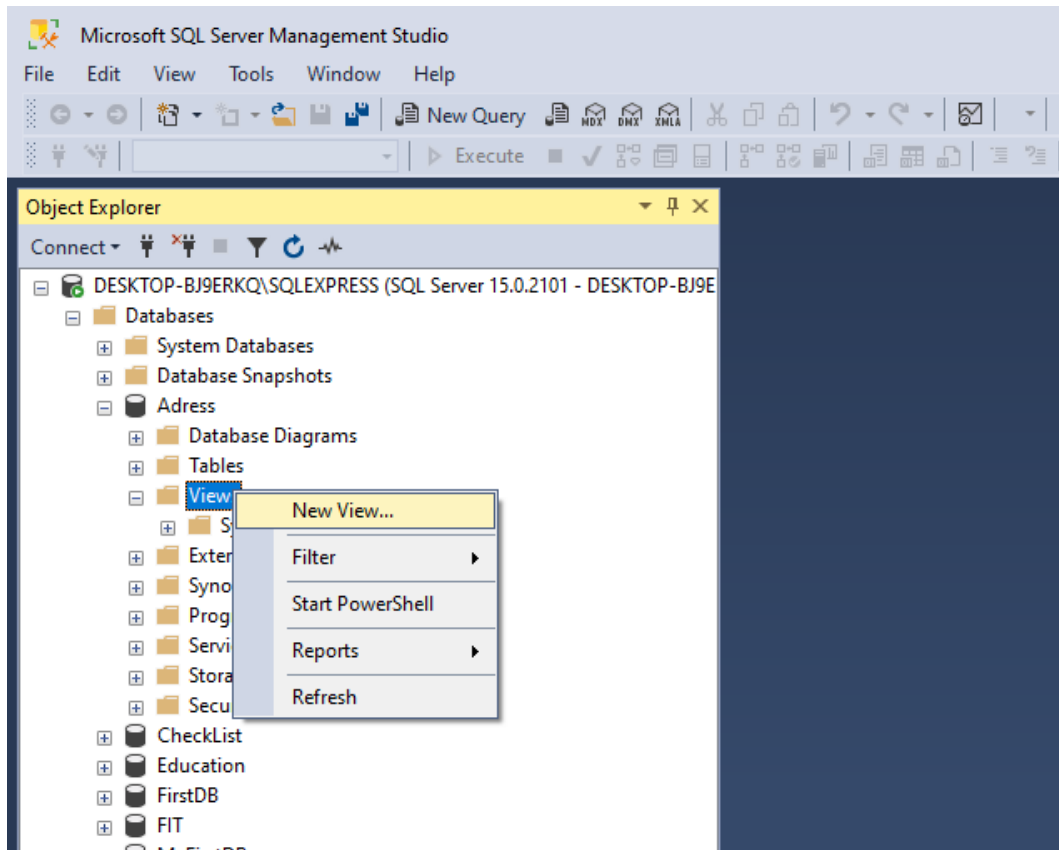


Рис. 8.1 Пункт контекстного меню «Створення подання»
в SQL Server Management Studio

Відкриється вікно додавання таблиці, в якому виберемо зі списку таблиць ті, які потрібні для створення подання – Employee і Person (рис. 8.2).

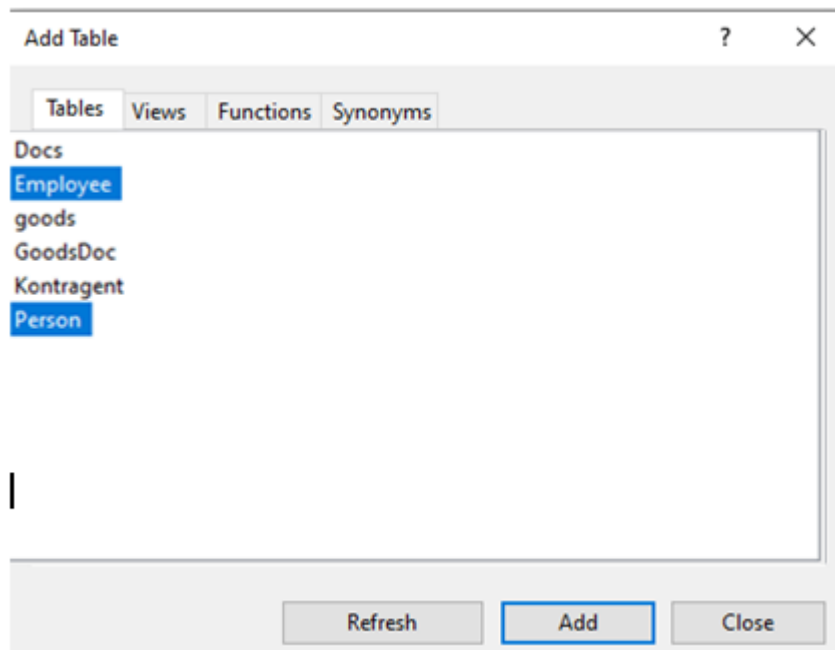


Рис. 8.2 Вибір таблиць БД для створення подання

Відкриється вікно, в якому виберемо ті стовпці таблиця, які потрібні для створення подання. Виберемо в таблиці Employee стовпці Tab та Interest, а в таблиці Person – стовпці Name, email і telefon (рис. 8.3). Задамо псевдонім Expr1 стовпцю Tab.

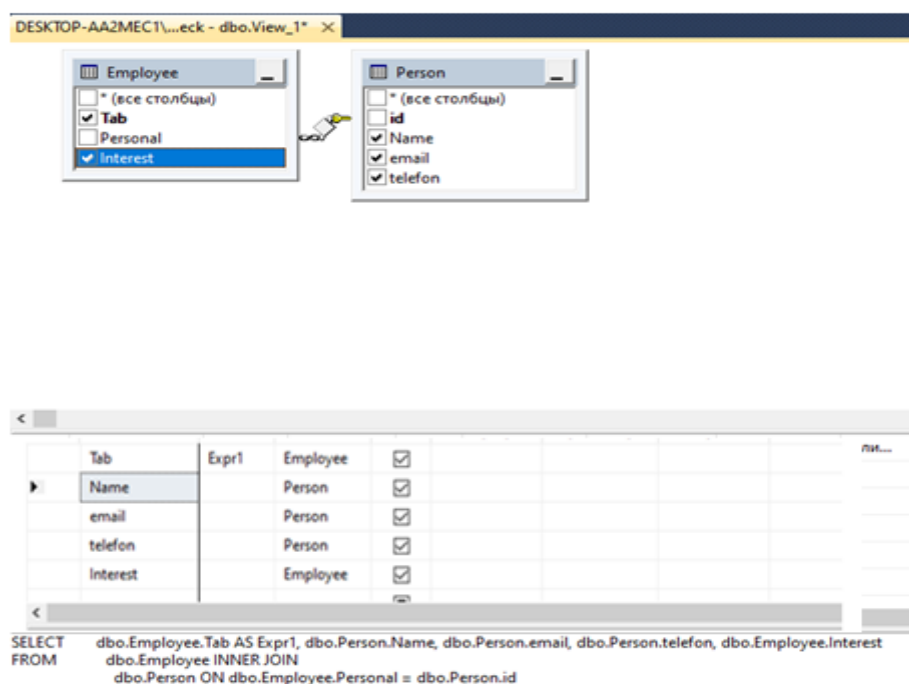


Рис. 8.3 Вибір стовпців таблиць для створення подання

Зберігаємо подання (Файл→Зберегти) з іменем View_1 (рис. 8.4).

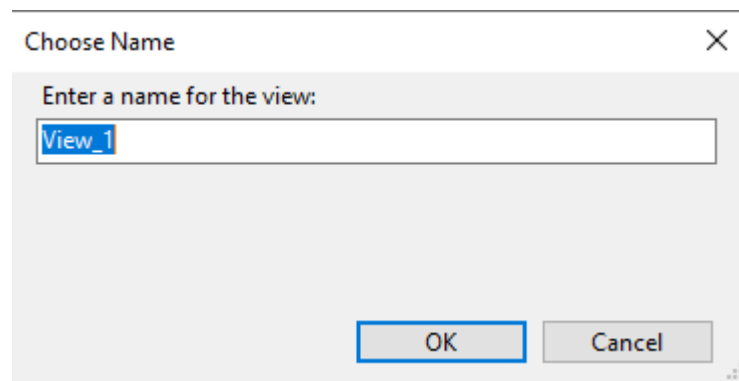


Рис. 8.4 Збереження подання

В результаті у теці Подання в провіднику об'єктів SSMS з'являється наше подання (рис. 8.5).

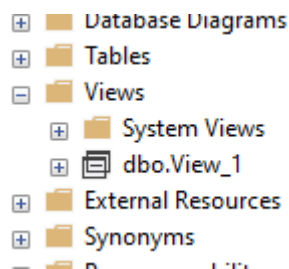


Рис. 8.5 Створене подання View_1 в провіднику об'єктів SSMS

Для перегляду створеного подання натисніть на ньому правою клавішею миші та виберіть в контекстному меню пункт «Вибрати перші 1000 рядків» або «Змінити перші 200 рядків». Подання має табличний вигляд та містить усі вибрані стовпці з обох таблиць Employee і Person. Зверніть увагу, що ім'я першого стовпця – заданий псевдонім Expr1 (рис. 8.6).

	Expr1	Name	email	telefon	Interest
1	1	Іванов Іван	ivanov@ukr.net	0671947186	5
2	2	Петров Петро	petrov@ukr.net	0733211466	10
3	3	Висоцький М.В.	ratusha.ukr.net	0962141566	5

Рис. 8.6 Перегляд подання View_1

Для того щоб створити подання SQL-запитом, створимо і виконаємо такий скрипт:

```

CREATE VIEW EmployView AS
SELECT Employee.Tab AS Tab,
       Person.Name AS Persona,
       Person.telefon AS Telefon
FROM Employee INNER JOIN Person ON Employee.Tab = Person.id;

```

В результаті у теці Подання в провіднику об'єктів SSMS з'являється нове подання – EmployView (рис. 8.7).

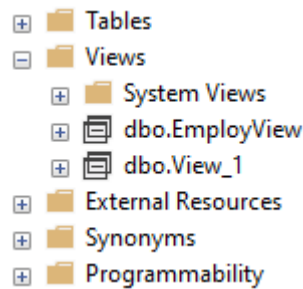


Рис. 8.7 Створене подання EmployView в провіднику об'єктів SSMS

Використаємо створене подання EmployView для отримання даних:

```

USE [check]
SELECT * FROM EmployView;

```

Результат:

	Tab	Persona	Telefon
1	1	Іванов Іван	0671947186
2	2	Петров Петро	0733211466
3	3	Вагнер Дмитро	NULL

Для зміни подання використовується команда ALTER VIEW. Ця команда має практично той же синтаксис, що і CREATE VIEW:

```

ALTER VIEW EmployView AS
SELECT Employee.Tab AS Tab,
       Person.Name AS Persona,
       Person.telefon AS Telefon,
       Person.email AS Email
FROM Employee INNER JOIN Person ON Employee.Tab = Person.id;

```

В результаті виконання цього запиту, отримуємо змінене подання. Переглядаємо його:

	Tab	Persona	Telefon	Email
1	1	Іванов Іван	0671947186	ivanov@ukr.net
2	2	Петров Петро	0733211466	petrov@ukr.net
3	3	Вагнер Дмитро	NULL	NULL

Для видалення подання викликається команда DROP VIEW:

ЗАВДАННЯ

Всі завдання виконувати в MS SQL Server Management Studio.

Варіант	Опис завдання
1	1. Створити подання, в якому для студентів певної групи будуть виводиться оцінки (як в журналі). 2. Написати скалярну функцію, яка повертає загальну кількість викладачів всіх факультетів. 3. Написати функцію, яка повертає таблицю всіх кафедр університету, згруповані по факультетам. 4. Написати процедуру, яка отримує назву факультету і визначає кількість кафедр на ньому.
2	1. Створити подання, в якому за датою продажу виводиться вся інформація про продані товари. 2. Написати скалярну функцію, яка за датою продажу розраховує суму продажу товарів за день. 3. Написати функцію, яка повертає таблицю з інформацією про товари, які продані за певний день. 4. Написати процедуру, яка розраховує суму продажу товарів за місяць.
3	1. Створити подання, в якому за видом військ буде подаватися інформація про роти та їх особистий склад. 2. Написати скалярну функцію, яка за поданими в неї А, В, С буде знаходити рішення квадратного рівняння. 3. Написати функцію, яка повертає таблицю, що за роком буде повертати всю інформацію про особистий склад. 4. Написати процедуру, яка знаходить суму всіх площ місць дислокації частин.
4	1. Створити подання, в якому виводиться інформація про

	співробітників певної бібліотеки. 2. Написати скалярну функцію, яка за заданою датою визначає назву місяця. 3. Написати функцію, яка повертає таблицю, в якій знаходиться інформація про всі фонди для певної бібліотеки. 4. Написати процедуру, яка переводить всю інформацію, подану в таблиці Співробітники, у верхній регістр.
5	1. Створити подання, в якому виводиться інформація про пансіонати з певним видом житла. 2. Написати скалярну функцію <i>arithmetic</i>, приймаючи 3 аргументи: перші 2 – це числа, третій – операція, яка повинна бути проведена над ними. Якщо третій аргумент +, скласти їх; відповідно - відняти; * помножити; / розділити (перше на друге). В інших випадках повернути рядок "Невідома операція". 3. Написати функцію, яка поверне інформацію про клієнтів, які купили путівки на 10 днів. 4. Написати процедуру, яка переводить всю інформацію, подану в таблиці Клієнти, у нижній регістр.
6	1. Створити подання, яке виводить всю інформацію про водіїв певного маршруту. 2. Обрахувати значення функції: $f(x) = \sin(2x) + \cos(3x) + 4 \cdot 85^3$ 3. Написати функцію, яка повертає всю інформацію про маршрути, де працював певний водій. 4. Написати процедуру, яка за поданою датою виводить місяць цієї дати на трьох мовах.
7	1. Створити подання, яке виводить інформацію про пацієнтів з певним діагнозом. 2. Написати функцію розрахування суми арифметичної прогресії. 3. Написати функцію, яка повертає назви діагнозів і призначення лікаря для певного пацієнта.

	4. Написати процедуру, яка знаходить різницю між двома датами.
8	1. Створити подання, в якому відображаються всі дати операцій для певного хірурга. 2. Написати функцію, яка повертає суму геометричної прогресії. 3. Написати функцію, яка повертає види лікування для певного діагнозу. 4. Написати процедуру, яка вираховує дату, що відмінна від заданої на 31 день.
9	1. Створити подання, в якому виводиться інформація, яку літературу брали у певний період. 2. Написати функцію, яка обраховує $f(x) = \cos 103^0 * \cos 43^0 + \sin 103^0 * \cos 43^0 + x - 85^x$ при $x_0 = \frac{\pi}{4}$ 3. Написати функцію, яка повертає літературу певної бібліотеки. 4. Написати процедуру, яка розраховує суму ряду: $u_n = \frac{3n - 2}{3^{n-1}}$.
10	1. Створити подання, яке відображає всі відомості про продані автомобілі. 2. Написати функцію <i>season</i>, яка приймає 1 аргумент – номер місяця (від 1 до 12), а повертає пору року, до якої цей місяць належить (зима, весна, літо або осінь). 3. Написати функцію, що повертає таблицю з інформацією про співробітників, які цього місяця мали продажі. 4. Написати процедуру, яка переводить всю інформацію, подану в таблиці Автомобілі, у нижній регістр.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Що таке вбудовані функції SQL?
2. Що таке збережені функції SQL?

3. Де зберігаються збережені функції на SQL Server?
4. Де зберігаються збережені процедури на SQL Server?
5. Що таке збережена процедура?
6. Різниця між функцією і процедурою?
7. Для чого призначені подання?

Лабораторна робота №9

Тема: Тригери. Транзакції

План

1. Створення, програмування і управління тригерами;
2. Транзакції;
3. Завдання;
4. Питання для самоперевірки.

9.1 Створення, програмування і управління тригерами

Тригер – це спеціальний тип збережених процедур, які запускаються сервером автоматично при виконанні тих чи інших дій з даними таблиці.

Тригери поділяють по типу команд, на які вони реагують:

- INSERT TRIGGER – запускаються у разі спроби вставити дані за допомогою команди INSERT;
- UPDATE TRIGGER – запускаються у разі спроби змінити дані за допомогою команди UPDATE;
- AUTHSMALL TRIGGER – запускаються у разі спроби видалити дані за допомогою команди DELETE.

Параметри FOR, AFTER і INSTEAD OF, які вказуються під час створення тригера, визначають його поведінку таким чином:

- FOR – запуск тригера під час виконання заданого списку команд;
- AFTER – запуск тригера після успішного виконання команд списку;
- INSTEAD OF – тригер викликається замість виконання команд списку.

Можна визначити кілька AFTER-тригерів для кожної операції INSERT, UPDATE і DELETE. За замовчуванням всі тригери є AFTER-тригерами.

Тригери можна створювати для тимчасових або системних таблиць. Команда створення тригера повинна бути першою в пакеті і застосовуватися тільки до однієї таблиці.

Формат створення тригера:

```
CREATE TRIGGER Ім'я тригера
ON {Ім'я таблиці \ Ім'я подання}
[WITH ENCRYPTION] -- шифрування коду тригера;
{{{FOR \ AFTER \ INSTEAD OF}
{[DELETE] \ [ \ ] [INSERT] \ [ \ ] [UPDATE]}}
[WITE APPEND] -- тільки для версій 6.5 і нижче;
[NOT FOR REPLICATION] -- не для реплікації;
AS sql_statement [... n] -- тіло тригера;
}}
|
|{(FOR \ AFTER \ INSTEAD OF}
|[INSERT] \ [ \ ] [UPDATE]}
[WITE APPEND] -- тільки для версій 6.5 і нижче;
[NOT FOR REPLICATION] -- НЕ для реплікації;
AS {IF UPDATE (column) -- при зміні стовпчика;
|[AND \ OR] UPDATE (column) [... n]] -- теж;
}
|
IF (COLUMNS_UPDATED() {bitwise_operator}
Update_bitmask)
{Comparison_operator} column_bitmask [... n]
}
sql_statement [... n] -- тіло
}
|
{{{FOR \ AFTER \ INSTEAD OF}
|[INSERT] \ [ \ ] [UPDATE]}
[WITE APPEND] -- тільки для версій 6.5 і нижче;
[NOT FOR REPLICATION] -- НЕ для реплікації;
AS {IF UPDATE (column) -- при зміні стовпчика;
|[AND \ OR] UPDATE (column) [... n]] -- теж;
IF (COLUMNS_UPDATED() {bitwise_operator}
Update_bitmask)
{Comparison_operator} column_bitmask [... n]
}
sql_statement [... n] -- тіло тригера
}
}
```

Приклад 9.1.

Створимо тригер, який у разі спроби додавання або редагування товарів у таблицю goods, збільшував ціну товару на 20%.

Запишемо і виконаємо такий запит:

```

USE [check];
GO
CREATE TRIGGER Goods_INSERT_UPDATE
ON goods
AFTER INSERT, UPDATE
AS
UPDATE goods
SET Price = Price + Price * 0.20
WHERE Id = (SELECT Id FROM inserted)
Go

USE [check];
GO

INSERT INTO goods
VALUES ('plate', 50)

```

Результат:

	Id	Name	Price
▶	1	Пальто жін. си...	3000
	2	Пальто жін. зи...	4500
	3	Пальто жін. чо...	12400
	4	Пальто унісекс...	5000
	5	Пальто чол. кл...	12500
	6	Пальто чол. ч...	6500
	10	Ball	100
	11	Bag	1000
	13	Ball	50
	1013	Galaxy S7	5000
	2013	plate	60
*	NULL	NULL	NULL

Зверніть увагу на нижній рядок результату – ціна товару «plate» записано 60, а не 50 як у запиті, оскільки спрацював тригер.

Для видалення тригера використовується команда DROP TRIGGER. Буває, що потрібно лише призупинити дію тригера, а не видаляти. У цьому випадку його можна тимчасово вимкнути за допомогою команди DISABLE TRIGGER. А коли тригер знадобиться, його можна увімкнути за допомогою команди ENABLE TRIGGER.

У разі додавання даних (під час команди INSERT) в тригері можливо отримати додані дані з віртуальної таблиці INSERTED. У разі видалення всі видалені дані розміщуються у віртуальній таблиці DELETED. Тригер

оновлення даних спрацьовує під час операції UPDATE. І в такому тригері можливо використати дві віртуальні таблиці. Таблиця INSERTED зберігає значення рядків після оновлення, а таблиця DELETED зберігає самі рядки, але до оновлення.

9.2. Транзакції

Мова запитів Transact-SQL взяла свою назву від слова транзакція. Адже транзакції справді є дуже потужним засобом управління базою даних.

Розглянемо класичне завдання – банківське проведення. Припустимо, що у нас є таблиця із двох полів – номер рахунку у банку та сума грошей на цьому рахунку. Нам необхідно переказати гроші з одного рахунку на інший. Для цього потрібно виконати запит на UPDATE, щоб зменшити суму першого рахунку на потрібну суму. Після цього потрібно виконати запит на UPDATE, щоб збільшити значення другого рахунку. Все начебто нормально. Але може так статися, що після зменшення першого рахунку різко вимкнули світло і сервер не встиг поповнити інший рахунок? Або сталась ще якась помилка, яка не дала завершити дію. Гроші вже зняті, але нікуди не записані, отже – вони зникли.

Якщо спочатку поповнювати рахунок, а потім знімати гроші, то може статись аналогічна ситуація, але вже навпаки: поповнення відбулось, а зняття з якихось причин ні. Тут вже банк ризикує збанкрутувати, адже з'явилась зайва сума на рахунку одного клієнта, а зняття з рахунку іншого не відбулося.

Так, раптове знеструмлення або збій у роботі сервера відбуваються рідко, але також бувають блокування записів. Один користувач може заблокувати запис для зміни і це не дасть запиту змінити його та зняття або збільшення рахунку не відбудеться.

Проблему вирішує транзакція. Перед виконанням операцій оновлення необхідно явно розпочати транзакцію. Після цього виконуємо дві операції UPDATE і після закінчення завершуємо транзакцію. Якщо на момент виконання одного із запитів відбувається збій, то всі зміни, що відбулися після початку транзакції, скасовуються.

Існує два різновиди транзакцій у SQL Server:

- Приховані транзакції – кожен оператор, такий як INSERT, UPDATE або DELETE, виконується в транзакції;
- Явні транзакції оголошені користувачем – оператори, згруповані у BEGIN TRANSACTION та COMMIT TRANSACTION.

Дуже важливо розуміти, що транзакції необхідні лише за модифікації даних, тобто під час використання операторів INSERT, UPDATE чи DELETE. Проста вибірка SELECT не змінює даних, і запам'ятовувати чи відкочувати нічого. Ні, виконувати операції вибірки в транзакції можна, але якщо транзакція не змінює дані, то нема чого її взагалі починати.

У транзакції можна вмикати далеко не всі оператори. Усередині транзакції не може бути таких операторів:

- ALTER DATABASE
- BACKUP LOG
- CREATE DATABASE
- DROP DATABASE
- RECONFIGURE
- RESTORE DATABASE
- RESTORE LOG
- UPDATE STATISTICS

Робота транзакцій забезпечується за допомогою журналу транзакцій бази даних. Оскільки журнал записує всю активність і зміни, які відбуваються, SQL Server може відновлювати дані автоматично, які були втрачені в момент знеструмлення, через системні помилки, у зв'язку з проблемами з боку клієнта, через запит скасування журналу тощо.

Сервер SQL автоматично гарантує, що всі завершені транзакції відображаються в базі даних у момент та після системної помилки. Для цього використовується журнал транзакцій, за допомогою якого завершуються успішно закінчені транзакції та скасовуються не завершені.

За допомогою прихованих транзакцій сервер гарантує, що дані будуть збережені в таблиці, якщо оператор додавання, зміни або видалення даних

виконано вдало. Якщо під час змін сталася помилка, всі зміни відкочуються. Припустимо, що оператор UPDATE змінює 1000 рядків. Якщо на 500-му рядку сталася помилка, то сервер відкочує всі вже зроблені зміни, як вони відбувалися в явній транзакції.

Користувач може встановлювати точку збереження або маркер усередині транзакції. Точка збереження визначає розташування, до якого може бути повернена транзакція. Транзакція має або продовжити виконання, або бути повністю відкоченою.

Початок транзакції в MS SQL Server має такий синтаксис:

```
BEGIN TRANSACTION  
[ { transaction_name | @transaction_name_variable }  
[WITH MARK['description']]]
```

Опція transaction_name – ім'я транзакції, визначене користувачем. Опція WITH MARK вказує, що транзакція маркована у журналі транзакцій.

Після завершення транзакції зміни необхідно запам'ятати в базі даних. Для цього використовується команда COMMIT

```
COMMIT [ TRANSACTION  
[ transaction_name | @tran_name_variable ] ]
```

Скасування транзакції та всіх змін здійснюватиметься командою ROLLBACK, яка у загальному вигляді виглядає таким чином:

```
ROLLBACK [ TRANSACTION  
[ transaction_name | @tran_name_variable  
| savepoint_name | @savepoint_variable ] ]
```

Дуже важливо розуміти, якщо розпочато транзакцію та змінено якісь записи, то ці записи блокуються, доки транзакція не буде завершена.

Приклад 9.2.

Розробимо транзакцію, яка очищує одну із таблиць і виводить дані з таблиці до відкату та після відкату:

```
USE [check]
-- Починаємо транзакцію
BEGIN TRANSACTION

-- Очищаємо таблицю
DELETE GoodsDoc

SELECT * FROM GoodsDoc

ROLLBACK TRANSACTION

SELECT * FROM GoodsDoc
```

Результат:

	Num	GoodsId	Q
--	-----	---------	---

	Num	GoodsId	Q
1	1	1	5
2	2	1	2
3	2	2	5
4	2	3	4
5	3	2	3
6	3	3	4
7	3	4	2

Як видно з результату, до відкату таблицю було дійсно очищено (вона порожня), а після відкату дані відновились з транзакції.

Приклад 9.3.

Візьмемо таблицю goods бази даних check (рис. 9.1).

DESKTOP-AA2MEC1\...heck - dbo.goods × SQLQuery1.sql - (l			
	Id	Name	Price
	1	Пальто жін. си...	3000
	2	Пальто жін. зи...	4500
	3	Пальто жін. чо...	12400
	4	Пальто унісекс...	5000
	5	Пальто чол. кл...	12500
	6	Пальто чол. ч...	6500
	10	Ball	100
	11	Bag	1000
	13	Ball	50
	1013	Galaxy S7	5000
	2013	plate	60
	2014	DRESS	20,4
	2016	DRESS1	18
	2017	DRESS2	15
	2018	DRESS2	12
*	NULL	NULL	NULL

Рис. 9.1 Таблиця goods бази даних check

Створимо дві послідовні транзакції. Перша транзакція додає товар DRESS3 з ціною 12.5 в таблицю goods, потім змінює його ціну на 15 і зберігає зміни. Друга – змінює ціну товару на 17 і робить відкат до попереднього стану. Виконаємо запит:

```
USE [check]
-- Починаємо транзакцію
BEGIN TRANSACTION

-- Вставляємо рядок даних у
-- таблицю товарів
INSERT INTO goods( Name, Price)
VALUES ( 'DRESS3', 12.50)

-- Оновити дані в останньому рядку
UPDATE goods
SET Price = 15
WHERE Name LIKE 'DRESS3'

COMMIT TRANSACTION

BEGIN TRANSACTION
-- Оновити дані в останньому рядку
UPDATE goods
SET Price = 17
WHERE Name LIKE 'DRESS3'

-- Відкотити транзакцію
ROLLBACK TRANSACTION
|
-- Вибрати всі дані з таблиці
SELECT * FROM goods
```

Результат:

	Id	Name	Price
9	13	Ball	50
10	1...	Galaxy S7	5000
11	2...	plate	60
12	2...	DRESS	20,4
13	2...	DRESS1	18
14	2...	DRESS2	15
15	2...	DRESS2	12
16	2...	DRESS3	18

Приклад 9.4.

Розглянемо вкладені транзакції:

```

BEGIN TRANSACTION T1
-- Вставляємо рядок даних у таблицю товарів
INSERT INTO goods
  (Name, Price)
VALUES ('T-shirt', 11.30)

-- Починаємо транзакцію 2
BEGIN TRANSACTION T2

-- Оновити дані в останньому рядку
UPDATE goods
SET Price = 14
WHERE Name LIKE 'T-shirt'

COMMIT TRANSACTION T2

-- Запам'ятовуємо зміни
COMMIT TRANSACTION T1

-- Вибрати всі дані з таблиці
SELECT * FROM goods

```

Результат:

	Id	Name	Price
10	1...	Galaxy S7	5000
11	2...	plate	60
12	2...	DRESS	20,4
13	2...	DRESS1	18
14	2...	DRESS2	15
15	2...	DRESS2	12
16	2...	DRESS3	18
17	2...	T-shirt	16,8

!!! Зверніть увагу, що ціна в результаті на T-shirt не 14, а 16,8, оскільки спрацьовує тригер на вставку (додає 20% від ціни).

Приклад 9.5.

Виконаємо відкат до визначеної точки:

```

USE [check]
-- Починаємо транзакцію

BEGIN TRANSACTION T1

-- Вставляємо рядок даних у таблицю товарів
INSERT INTO goods
(Name, Price)
VALUES ('T-shirt1', 14)

-- Зберігаємо транзакцію
SAVE TRAN ins_complete

-- Оновити дані в останньому рядку
UPDATE goods
SET Price = 20
WHERE Name LIKE 'T-shirt1'

-- Відкочуємо транзакцію
ROLLBACK TRANSACTION ins_complete

-- Запам'ятовуємо зміни
COMMIT TRANSACTION T1

-- Вибрати всі дані з таблиці
SELECT * FROM goods

```

Результат:

	Id	Name	Price
11	2...	plate	60
12	2...	DRESS	20,4
13	2...	DRESS1	18
14	2...	DRESS2	15
15	2...	DRESS2	12
16	2...	DRESS3	18
17	2...	T-shirt	16,8
18	2...	T-shirt1	16,8

!!! Зверніть увагу, що ціна в результаті на T-shirt1 не 14, а 16,8, оскільки спрацював тригер на вставку (додає 20% від ціни).

ЗАВДАННЯ

Створити для своєї БД тригери для додавання записів, редагування та видалення. Виконати приклади 9.2-9.5 для своїх таблиць. Всі запити виконати мають бути виконані в Azure Data Studio.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Що таке тригер?
2. Які тригери ви знаєте?
3. Звідки бере інформацію тригер для додавання?
4. Звідки бере інформацію тригер для видалення?
5. Звідки бере інформацію тригер для зміни?
6. Що таке транзакція?
7. Які операції дозволені в транзакції?
8. Що таке точка відкату?

Лабораторна робота №10

Тема: Розроблення програмних застосунків з реляційними БД

План

1. Введення в ADO.NET
2. C# и MS SQL Server
3. Визначення контексту даних і моделей
4. Фільтрація і сортування
5. Зміна об'єктів в LINQ to SQL
6. Додавання об'єктів в LINQ to SQL
7. Видалення об'єктів в LINQ to SQL
8. Платформа Entity framework
9. Встановлення пакету EF6
10. Підходи Code First до існуючої бази даних
11. Додавання об'єктів через DbSet
12. Редагування об'єктів через DbSet
13. Видалення об'єктів через DbSet
14. Завдання;
15. Питання для самоперевірки.

10.1 Введення в ADO.NET

ADO.NET – технологія роботи з даними, яка заснована на платформі .NET Framework. Ця технологія надає набір класів для відправлення запитів до баз даних, встановлення підключення, отримання відповіді від бази даних і ряду інших операцій.

Причому важливо зазначити, що систем управління базами даних (СУБД) може бути безліч. У своїй сутності вони можуть відрізнятися. MS SQL Server, наприклад, для створення запитів використовує мову T-SQL, а MySQL і Oracle застосовують мову PL-SQL. Різні СУБД можуть мати різні типи даних. Однак

функціонал ADO.NET побудований таким чином, щоб надати розробникам уніфікований інтерфейс для роботи з найрізноманітнішими СУБД.

Оснoву інтерфейсу взаємодії з базами даних в ADO.NET складає обмежене коло об'єктів: Connection, Command, DataReader, DataSet і DataAdapter. За допомогою об'єкта Connection встановлюється підключення до джерела даних. Об'єкт Command дає змогу виконувати операції з даними з БД. Об'єкт DataReader зчитує отримані у результаті запиту дані. Об'єкт DataSet призначений для зберігання даних з БД і надає можливість працювати з ними незалежно від БД. Об'єкт DataAdapter є посередником між DataSet і джерелом даних. Загалом через ці об'єкти і буде здійснюватися робота з базою даних.

Однак, щоб використовувати один і той же набір об'єктів для різних джерел даних, потрібен відповідний провайдер даних. Власне через провайдера 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.

За замовчуванням в ADO.NET є такі вбудовані провайдери:

- Провайдер для MS SQL Server;
- Провайдер для OLE DB – надає доступ до деяких старих версій MS SQL Server, а також до БД Access, DB2, MySQL і Oracle;
- Провайдер для ODBC – використовується для тих джерел даних, для яких немає своїх провайдерів;
- Провайдер для Oracle;
- Провайдер EntityClient – використовується для технології ORM Entity Framework;
- Провайдер для сервера SQL Server Compact 4.0.

Крім цих провайдерів, які є вбудованими, існує також безліч інших, призначених для різних СУБД, наприклад, для MySQL.

Основні простори імен, які використовуються в ADO.NET:

- System.Data визначає класи, інтерфейси, делегати, які реалізують архітектуру ADO.NET;

- System.Data.Common містить класи, спільні для всіх провайдерів ADO.NET;
- System.Data.Design визначає класи, які використовуються для створення своїх власних наборів даних;
- System.Data.Odbc визначає функціональність провайдера даних для ODBC;
- System.Data.OleDb визначає функціональність провайдера даних для OLE DB;
- System.Data.Sql зберігає класи, які підтримують специфічну для SQL Server функціональність;
- System.Data.OracleClient визначає функціональність провайдера для баз даних Oracle;
- System.Data.SqlClient визначає функціональність провайдера для баз даних MS SQL Server;
- System.Data.SqlServerCe визначає функціональність провайдера для SQL Server Compact 4.0;
- System.Data.SqlTypes містить класи для типів даних MS SQL Server
- Microsoft.SqlServer.Server: зберігає компоненти для взаємодії SQL Server і середовища CLR.

Функціонально класи ADO.NET можна розбити на два рівня: підключень і відключень. Кожен провайдер даних .NET реалізує свої версії об'єктів Connection, Command, DataReader, DataAdapter і ряду інших, які складають рівень підключень. Тобто за допомогою них встановлюється підключення до БД і виконується взаємодія з нею. Як правило, реалізації цих об'єктів для кожного конкретного провайдера в своїй назві мають префікс, який вказує на назву провайдера.

Інші класи, такі як DataSet, DataTable, DataRow, DataColumn і ряд інших, складають рівень відключень, оскільки після отримання даних в DataSet можливо працювати з цими даними незалежно від того, чи встановлено підключення, чи ні. Тобто після отримання даних з БД застосунок може бути відключено від джерела даних.

10.2 C# і MS SQL Server

Приклад 10.1. Рядок підключення

Створимо проєкт простого консольного застосунку.

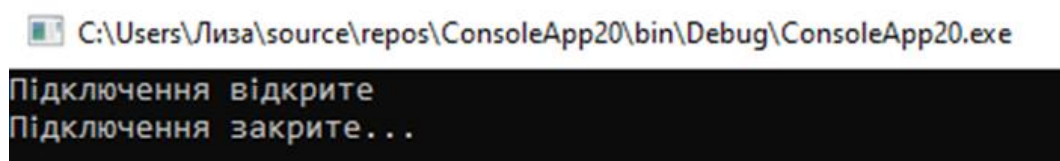
Насамперед визначимо рядок підключення, що надає інформацію про базу даних та сервер, до яких належить встановити підключення:

```
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317";
            // Створюємо підключення
            SqlConnection connection = new SqlConnection(connectionString);
            try
            {
                // Відкриваємо підключення
                connection.Open();
                Console.WriteLine("Підключення відкрите");
            }
            catch (SqlException ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                // закриваємо підключення
                connection.Close();
                Console.WriteLine("Підключення закрито...");
            }

            Console.Read();
        }
    }
}
```

Результат:



```
C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe
Підключення відкрите
Підключення закрито...
```

У разі використання різних систем управління базами даних, різних провайдерів даних .NET рядок підключення може відрізнятися. Навіть для підключення однієї і тієї ж бази даних рядок підключення може змінюватися залежно від обставин.

Рядок підключення — це набір параметрів у вигляді пар «ключ = значення». В цьому разі для підключення до раніше створеної бази даних check треба визначити рядок підключення з чотирьох параметрів:

- Data Source – назва 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 – назва бази даних на сервері.
- User Id – Id користувача SQL Server.
- Password – пароль цього користувача.

Жорстке кодування рядка підключення (тобто його визначення в коді програми), як правило, рідко використовується. Набагато більш гнучкий шлях – визначення його в спеціальних конфігураційних файлах програми. У проєктах десктопних застосунків – це файл App.config, а в веб-застосунках – це в основному файл Web.config.

Оскільки у прикладі створено проєкт консольного застосування, то він містить файл App.config (рис. 10.1).

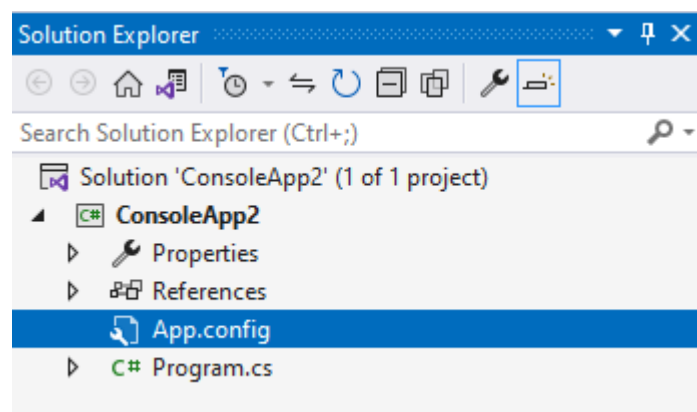
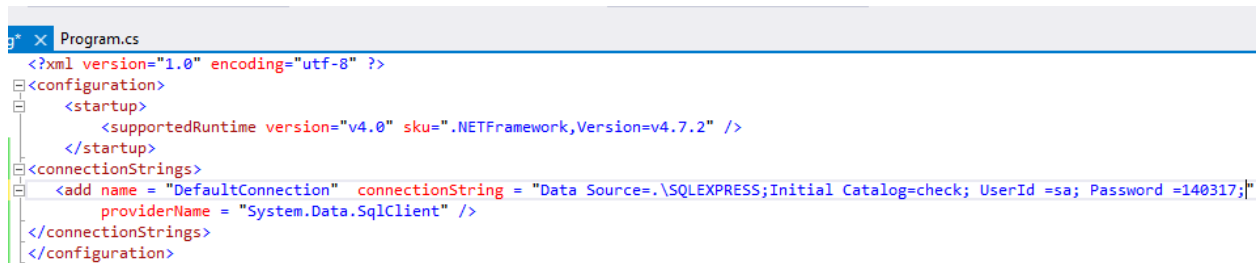


Рис. 10.1 Файл App.config в оглядачі рішень

На цьому етапі файл App.config має такий зміст:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
</configuration>
```

Додамо у цей файл визначення рядка підключення:



```

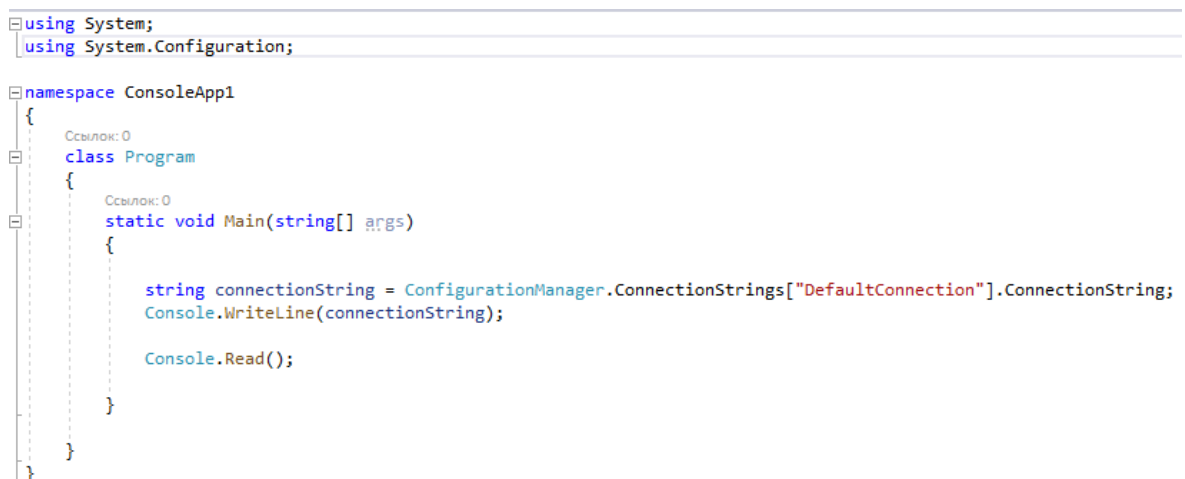
Program.cs
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=check; UserId =sa; Password =140317;"
        providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>

```

Для визначення всіх підключень в програмі в межах вузла `<configuration>` додається новий вузол `<connectionStrings>`. У цьому вузлі визначаються рядки підключення за допомогою елемента `<add>`. У застосунку можливо використовувати безліч рядків підключення. У файлі також можна визначити безліч елементів `<add>`.

Кожен рядок підключення має назву, яка визначається за допомогою атрибута `name`. В цьому разі рядок підключення називається "DefaultConnection". Назва може бути довільною. Атрибут `connectionString` зберігає рядок підключення, тобто весь той текст, який було визначено вище в методі `Main`. Третій атрибут – `providerName` – задає простір імен провайдера даних. Для підключення до бази даних MS SQL Server варто використовувати провайдер для SQL Server, функціональність якого полягає в просторі імен `System.Data.SqlClient`.

Тепер отримаємо цей рядок підключення у застосунку:



```

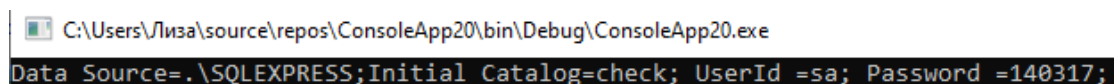
using System;
using System.Configuration;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
            Console.WriteLine(connectionString);

            Console.Read();
        }
    }
}

```

Результат:



```

C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe
Data Source=.\SQLEXPRESS;Initial Catalog=check; UserId =sa; Password =140317;

```

Для роботи з конфігурацією застосунку потрібно додати в проєкт бібліотеку `System.Configuration.dll`. Для цього в оглядачі рішень натискаємо

правою кнопкою миші на вузлі Посилання і вибираємо у контекстному меню Додати посилання (рис. 10.2).

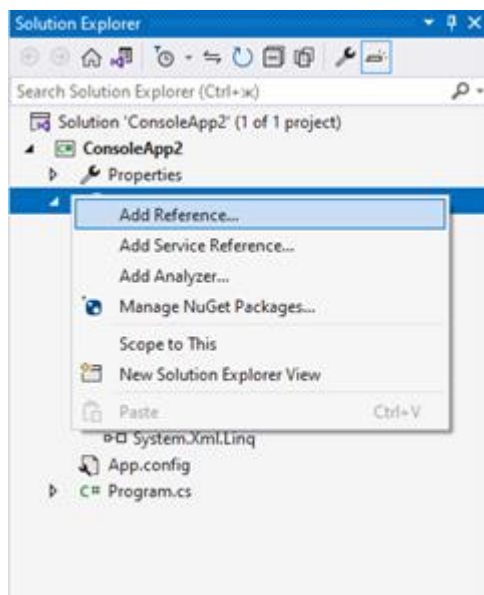


Рис. 10.2 Пункт «Додати посилання» контекстного меню в оглядачі рішень

Відкриється вікно управління посиланнями, ставимо галочку напроти System.Configuration.

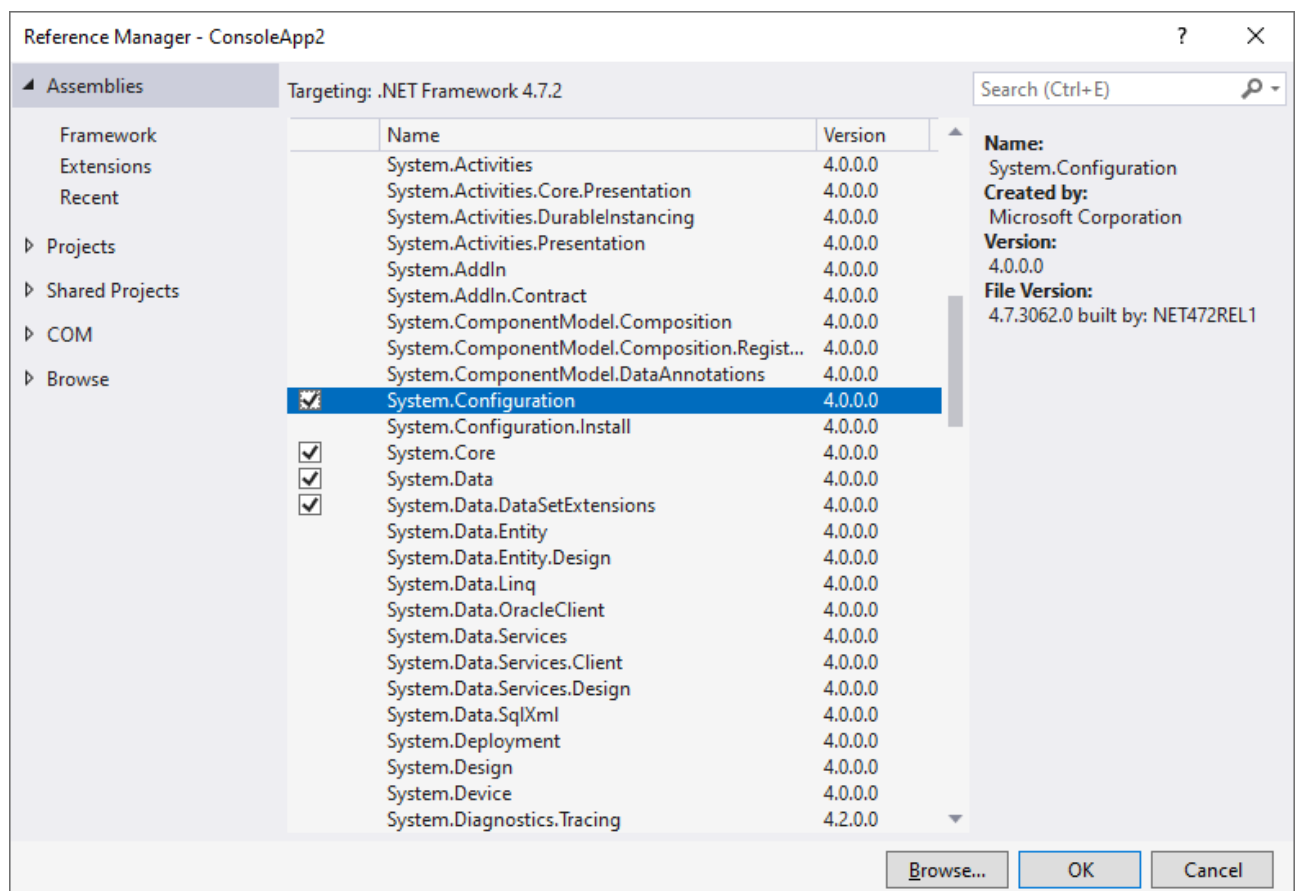


Рис. 10.3 Вікно управління посиланнями

За допомогою об'єкта `ConfigurationManager.ConnectionStrings` ["назва_рядка_підключення"] можливо отримати рядок підключення і використовувати його в застосунку.

Параметри рядка підключення:

- `Application Name` – назва програми. Може приймати в якості значення будь-який рядок. Значення за замовчуванням: `".Net SqlClient Data Provide"`;
- `AttachDBFileName` – зберігає повний шлях до бази даних;
- `Connect Timeout` – часовий період в секундах, через який очікується установка підключення. Приймає одне зі значень з інтервалу 0-32767. За замовчуванням дорівнює 15. В якості альтернативної назви параметра може використовуватися `Connection Timeout`;
- `Data Source` – назва примірника SQL Server, з яким буде йти взаємодія;
- `Integrated Security` – встановлює перевірку пароля.

Приклад 10.2. Створення підключення

Для підключення до бази даних необхідно створити і використовувати об'єкт `SqlConnection`:

```
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317";
            // Створюємо підключення
            SqlConnection connection = new SqlConnection(connectionString);
            try
            {
                // Відкриваємо підключення
                connection.Open();
                Console.WriteLine("Підключення відкрите");
            }
            catch (SqlException ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                // Закриваємо підключення
                connection.Close();
                Console.WriteLine("Підключення закрито...");
            }

            Console.Read();
        }
    }
}
```

У конструктор об'єкту `SqlConnection` передається рядок підключення, який ініціалізує об'єкт. Щоб використовувати цей об'єкт і підключатися до бази даних, потрібно виконати його метод `Open()`, а після завершення роботи з базою даних треба викликати метод `Close()` для закриття підключення. У разі виникнення помилок закриття підключення відбувається в блоці `finally`.

В якості альтернативного методу можна використовувати конструкцію `using`, яка автоматично закриває підключення:

```
using System;
using System.Data.SqlClient;

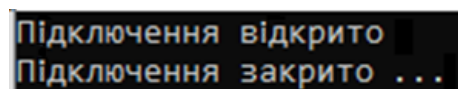
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";

            // Створення підключення
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                // Відкриваємо підключення
                connection.Open();
                Console.WriteLine("Підключення відкрито");
            }

            Console.WriteLine("Підключення закрито ...");

            Console.Read();
        }
    }
}
```

Результат:



```
Підключення відкрито
Підключення закрито ...
```

Приклад 10.3. Отримання інформації про підключення

Об'єкт `SqlConnection` має низку властивостей, які дають змогу отримати інформацію про підключення:

```

using System;
using System.Data.SqlClient;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";

            // Створення підключення
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                // Відкриваємо підключення
                connection.Open();
                Console.WriteLine("Підключення відкрито");
                // Виведення інформації про підключення
                Console.WriteLine("Властивості підключення:");
                Console.WriteLine("\tРядок підключення: {0}", connection.ConnectionString);
                Console.WriteLine("\tБаза даних: {0}", connection.Database);
                Console.WriteLine("\tСервер: {0}", connection.DataSource);
                Console.WriteLine("\tВерсія сервера: {0}", connection.ServerVersion);
                Console.WriteLine("\tСтояние: {0}", connection.State);
                Console.WriteLine("\tWorkstationId: {0}", connection.WorkstationId);
            }

            Console.WriteLine("Підключення закрито ...");
        }
    }
}

```

Результат:

```

C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe
Підключення відкрито
Властивості підключення:
Рядок підключення: Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa;
База даних: check
Сервер: .\SQLEXPRESS
Версія сервера: 12.00.2269
Стояние: Open
WorkstationId: DESKTOP-AA2MEC1
Підключення закрито ...

```

Приклад 10.4. Можливі помилки та їх вирішення

При підключенні до бази даних можуть виникнути помилки. Найбільш поширеною помилкою є така:

Unhandled Exception: System.ArgumentException: Keyword not supported ...

Така помилка викликана через неправильне задання параметрів рядка підключення. До того ж неважливо, як було визначено рядок підключення – в коді або в файлі конфігурації. У повідомленні про помилку вказується, який параметр задано неправильно, і в цьому випадку його треба виправити.

Ще одна поширена помилка така:

Can not open database "назва бази даних" requested by the login. The login failed.

Login failed for user 'користувач'

Вона означає, що даної бази немає або принаймні немає подібної бази даних. В цьому випадку треба переконатися, що на сервері є база даних з такою назвою, а якщо є, то перевірити, чи є доступ для даного користувача до цієї БД.

Також часто зустрічається помилка:

A network-related or instance-specific error occurred while establishing a connection to SQL Server.

The server was not found or was not accessible ...

У цьому випадку, як правило, SQL Server не запущений. І його треба запустити або перезапустити через панель служб. Також може бути, що в рядку підключення вказано неправильне ім'я сервера, або MS SQL Server взагалі не встановлено.

Приклад 10.5. Виконання команд і SqlCommand

Після встановлення підключення можливо виконувати команди з базою даних, наприклад, додати в базу даних об'єкт, видалити, змінити його або просто витягти. Команди подані об'єктом інтерфейсу System.Data.IDbCommand. Провайдер для MS SQL надає його реалізацію у вигляді класу SqlCommand. Цей клас інкапсулює sql-вираз, який потрібно виконати.

Для виконання команди нам буде потрібно sql-вираз і об'єкт підключення:

```

class Program
{
    Ссылка: 0
    static void Main(string[] args)
    {
        string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";

        // Створення підключення
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            // Відкриваємо підключення
            connection.Open();
            Console.WriteLine("Підключення відкрито");
            SqlCommand command = new SqlCommand();
            command.CommandText = "SELECT * FROM Users";
            command.Connection = connection;

        }

        Console.WriteLine("Підключення закрито ...");

        Console.Read();
    }
}

```

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

В якості альтернативи можна було б використовувати одну з версій конструктора класу:

```

class Program
{
    Ссылка: 0
    static void Main(string[] args)
    {
        string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";

        // Створення підключення
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            // Відкриваємо підключення
            connection.Open();
            Console.WriteLine("Підключення відкрито");
            SqlCommand command = new SqlCommand();
            command.CommandText = "SELECT * FROM Employee";
            command.Connection = connection;

        }

        Console.WriteLine("Підключення закрито ...");

        Console.Read();
    }
}

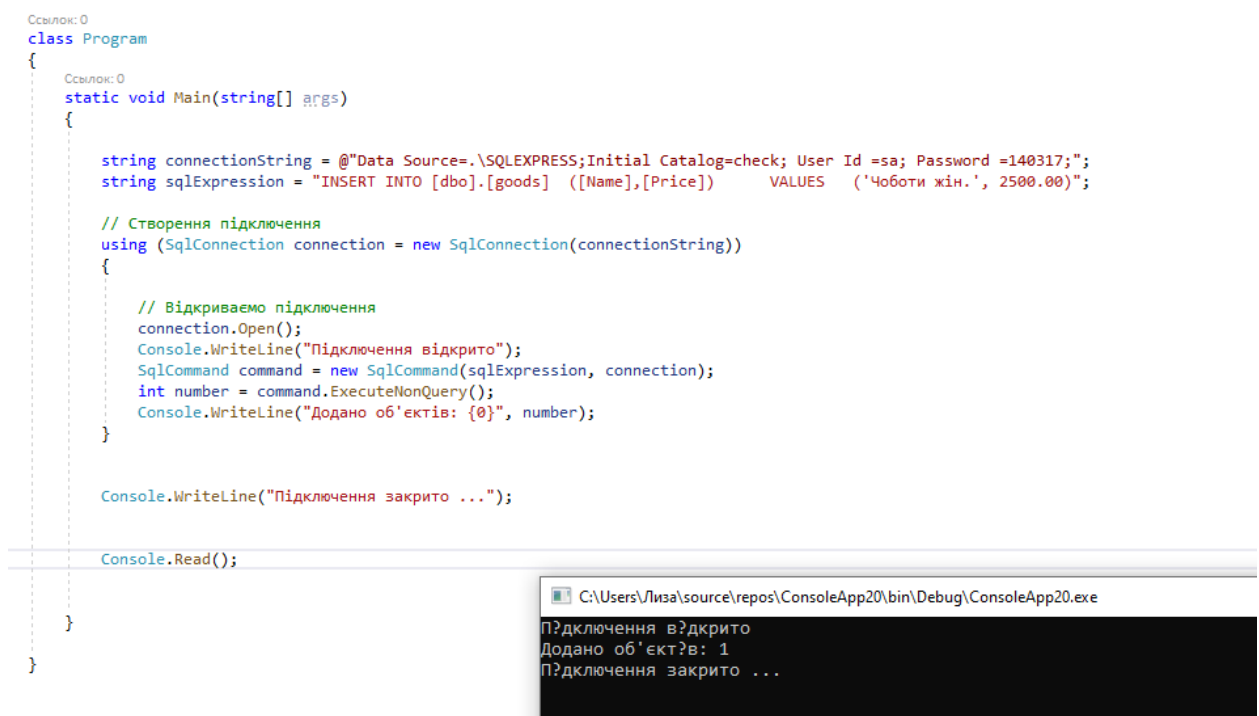
```

Щоб виконати команду, необхідно застосувати один з методів SqlCommand:

- ExecuteNonQuery – просто виконує sql-вираз і повертає кількість змінених записів. Підходить для sql-виразів INSERT, UPDATE, DELETE.
- ExecuteReader – виконує sql-вираз і повертає рядки з таблиці. Підходить для sql-вирази SELECT.
- ExecuteScalar – виконує sql-вираз і повертає одне скалярне значення, наприклад, число. Підходить для sql-виразу SELECT в парі з однією із вбудованих функцій SQL, як наприклад, Min, Max, Sum, Count.

Приклад 10.6. Додавання об'єктів

Виконаємо команду додавання одного об'єкта в таблицю goods бази даних check (рис. 10.4).



```
Ссылка: 0
class Program
{
    Ссылка: 0
    static void Main(string[] args)
    {
        string connectionString = @"Data Source=.\\SQLEXPRESS;Initial Catalog=check; User Id=sa; Password =140317;";
        string sqlExpression = "INSERT INTO [dbo].[goods] ([Name],[Price]) VALUES ('Чоботи жін.', 2500.00)";

        // Створення підключення
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            // Відкриваємо підключення
            connection.Open();
            Console.WriteLine("Підключення відкрито");
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            int number = command.ExecuteNonQuery();
            Console.WriteLine("Додано об'єктів: {0}", number);
        }

        Console.WriteLine("Підключення закрито ...");

        Console.Read();
    }
}
```

C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

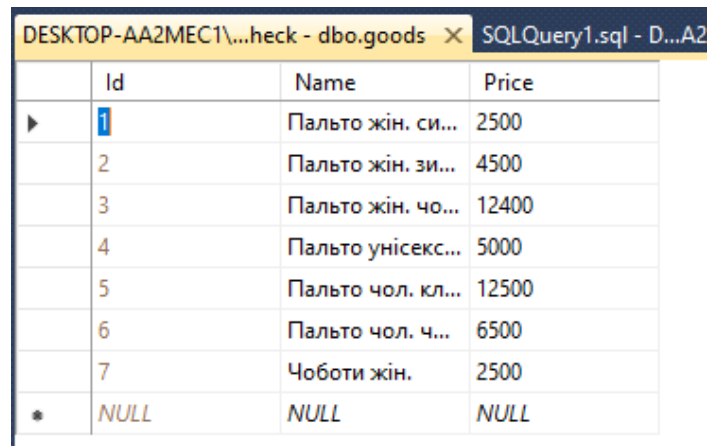
Підключення відкрито
Додано об'єктів: 1
Підключення закрито ...

Рис. 10.4 Додавання одного об'єкта в таблицю goods бази даних check

Для вставки об'єкта використовується sql-вираз INSERT, що має такий синтаксис:

```
INSERT INTO назва_таблиці (стовпчик1, стовпчик2, стовпчикN) VALUES  
(значення1, значення2, значенняN)
```

В таблиці goods бази даних check є три стовпці – Id і Price, які зберігають цілі числа і дійсні числа, та Name, який зберігає рядок. Тому відповідно ми додаємо для стовпця Name значення ‘Чоботи жін.’, а для стовпця Price число 2500.00 (рис. 10.5).



	Id	Name	Price
▶	1	Пальто жін. си...	2500
	2	Пальто жін. зи...	4500
	3	Пальто жін. чо...	12400
	4	Пальто унісекс...	5000
	5	Пальто чол. кл...	12500
	6	Пальто чол. ч...	6500
	7	Чоботи жін.	2500
*	NULL	NULL	NULL

Рис. 10.5 Результат додавання запису в таблицю goods бази даних check

Метод ExecuteNonQuery() повертає число змінених рядків (в цьому випадку доданих в таблицю об’єктів). Не обов’язково повертати результат методу, але він може використовуватися задля перевірки, що операція, зокрема, додавання, пройшла успішно.

Приклад 10.7. Оновлення об’єктів

Оновлення буде відбуватися аналогічно, тільки за допомогою sql-виразу UPDATE, який має такий синтаксис:

```
UPDATE назва_таблиці  
SET стовпчик1 = значення1, стовпчик 2 = значення2, стовпчик N = значенняN  
WHERE деякий_стовпчик = деяке_значення
```

Застосуємо цей вираз для зміни ціни «Чоботи жін.» з 2500.00 на 6500.50 (рис. 10.6)



Рис. 10.6 Оновлення рядків в таблиці goods

Тут оновлюється рядок, в якому Name = 'Чоботи жін.', тобто вище доданий об'єкт (рис. 10.7). Якщо в таблиці буде кілька рядків, у яких Name = 'Чоботи жін.', то оновляться всі ці рядки.

DESKTOP-AA2MEC1\...heck - dbo.goods X SQLQuery1.sql - D...			
	Id	Name	Price
5	1	Пальто жін. си...	2500
	2	Пальто жін. зи...	4500
	3	Пальто жін. чо...	12400
	4	Пальто унісекс...	5000
	5	Пальто чол. кл...	12500
	6	Пальто чол. ч...	6500
	7	Чоботи жін.	6500,5
*	NULL	NULL	NULL

Рис. 10.7 Результат оновлення рядків «Чоботи жін.» в таблиці goods

Приклад 10.8. Видалення об'єктів

Видалення проводиться за допомогою sql-виразу DELETE, який має такий синтаксис:

DELETE FROM таблиця

WHERE стовпець = значення

Видалимо, наприклад, всі товари з назвою ‘Чоботи жіні.’ (рис. 10.8).



Рис. 10.8 Видалення рядків в таблиці goods

У всіх трьох випадках фактично змінюється тільки sql-вираз, а інша логіка залишається незмінною. І ми також можемо виконувати відразу декілька операцій:

Ссылка 0

```
static void Main(string[] args)
{
    string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";

    // Створення підключення
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        // Відкриваємо підключення
        connection.Open();
        Console.WriteLine("Підключення відкрито");
        Console.WriteLine("Введіть назву товару, який потрібно додати:");
        string name1 = Console.ReadLine();

        Console.WriteLine("Введіть його ціну:");
        double price = Convert.ToDouble(Console.ReadLine());

        string sqlExpression = String.Format("INSERT INTO [dbo].[goods](Name, Price) VALUES ({0}, {1})", "" + name1 + "", price );

        SqlCommand command = new SqlCommand(sqlExpression, connection);
        int number = command.ExecuteNonQuery();
        Console.WriteLine("Додано об'єктів: {0}", number);
        Console.WriteLine("Введіть назву товару, ціну якого потрібно змінити:");
        name1 = Console.ReadLine();

        Console.WriteLine("Введіть його нову ціну:");
        price = Convert.ToDouble(Console.ReadLine());

        sqlExpression = String.Format("UPDATE [dbo].[goods] SET Price = {0} WHERE Name = {1}", price, "" + name1 + "");

        command = new SqlCommand(sqlExpression, connection);
        number = command.ExecuteNonQuery();
        Console.WriteLine("Змінено об'єктів: {0}", number);

        Console.WriteLine("Введіть назву товару, який потрібно видалити:");
        name1 = Console.ReadLine();

        sqlExpression = String.Format("DELETE FROM [dbo].[goods] WHERE Name = {0}", "" + name1 + "");

        command = new SqlCommand(sqlExpression, connection);
        number = command.ExecuteNonQuery();
        Console.WriteLine("Видалено об'єктів: {0}", number);
    }

    Console.WriteLine("Підключення закрито ...");

    Console.Read();
}
```

Результат:

```
Підключення відкрито
Введіть назву товару, який потрібно додати:
ball
Введіть його ціну:
100
Додано об'єктів: 1
Введіть назву товару, ціну якого потрібно змінити:
ball
Введіть його нову ціну:
150
Змінено об'єктів: 2
Введіть назву товару, який потрібно видалити:
ball
Видалено об'єктів: 2
Підключення закрито ...
```

Приклад 10.9. Читання результатів запиту і SqlDataReader

У попередньому прикладі було розглянуто, як виконувати команди за допомогою методу `ExecuteNonQuery()`. Якщо ж потрібно зчитувати дані, які зберігаються в таблиці, то нам буде потрібно інший метод – `ExecuteReader()`. Цей метод повертає об'єкт `SqlDataReader`, який використовується для читання даних. Отримаємо всі дані з таблиці `goods` і виведемо їх на консоль (рис. 10.9).

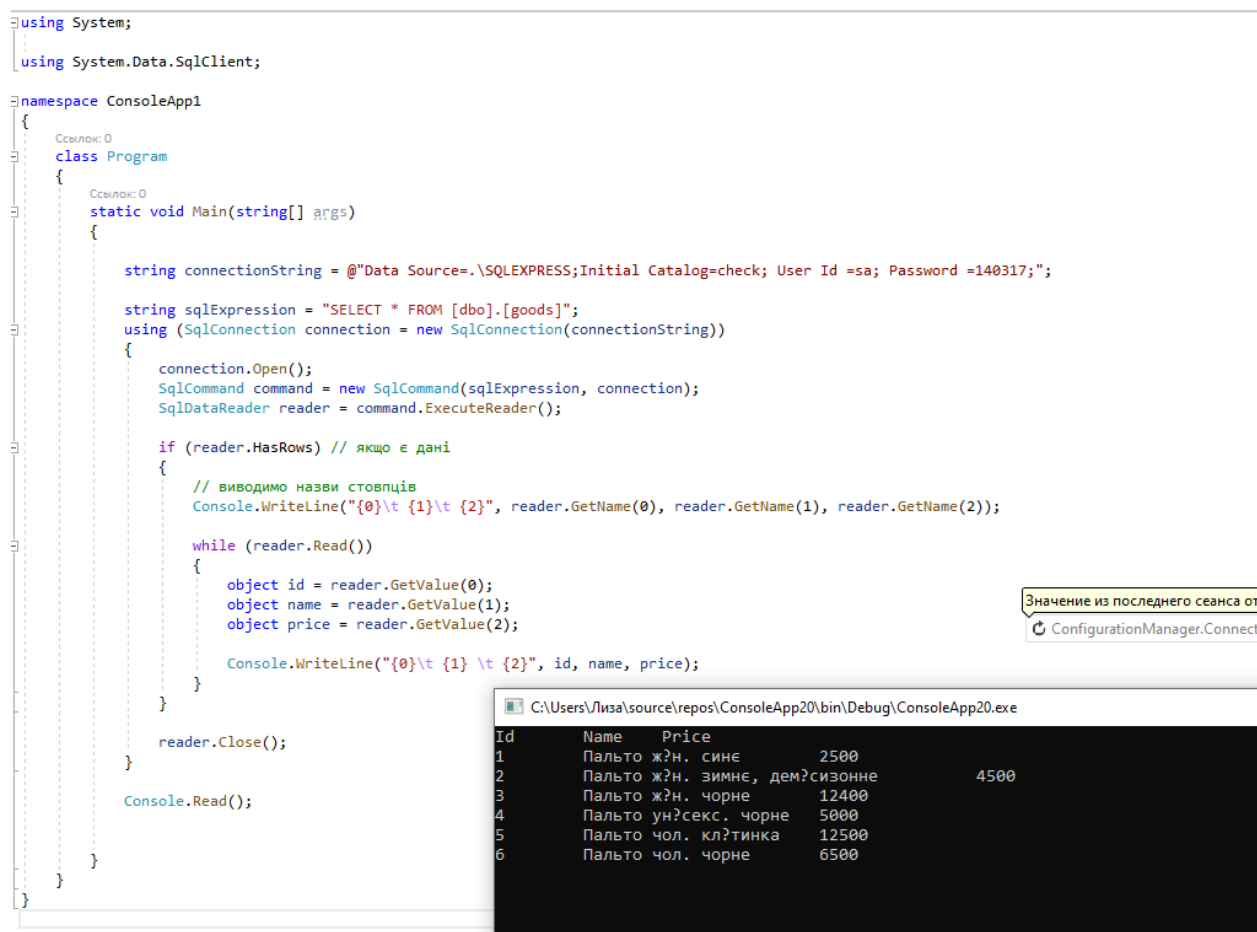


Рис. 10.9 Виведення усіх даних з таблиці `goods` в консоль

Для вибірки даних з БД використовується `sql`-вираз `SELECT`. У цьому прикладі вибираються всі стовпці усіх рядків таблиці. Результатом виконання запиту є об'єкт `SqlDataReader`, який надає потрібні дані.

Але спочатку виконується перевірка за допомогою властивості `HasRows`, а чи є взагалі дані. Якщо дані є, то виводяться заголовки таблиці за допомогою методів `reader.GetName()`. Варто зазначити, що отримані стовпці розташовані в вибірці саме в тому порядку, в якому вони визначені в таблиці. Тобто, якщо другим в таблиці йде стовпець `"Name"`, то, щоб отримати його, застосовується

метод `GetName(1)`. Тут індекс у дужках 1, оскільки нумерація стовпців починається з нуля.

Далі зчитуємо самі дані. За допомогою методу `reader.Read()` відбувається перехід до наступного рядка і повертається логічне значення, яке вказує, чи є дані для зчитування.

У циклі `while (reader.Read ())` в порядку проходження стовпців отримуємо дані за допомогою методу `GetValue()`, який повертає дані у вигляді об'єкти типу `object`. Наприклад, стовпець `Id` є першим, тому для його отримання застосовується метод `reader.GetValue(0)`. А стовпець `Name` є другим, тому його значення отримуємо за допомогою `reader.GetValue (1)`.

Після завершення роботи з `SqlDataReader` його потрібно закрити методом `Close()`. Поки один `SqlDataReader` не закритий, інший об'єкт `SqlDataReader` для одного і того ж підключення використовувати не вдасться.

В якості альтернативи можливо звертатися до даних через назву параметра:

```
while (reader.Read())
{
    object id = reader["Id"];
    object name = reader["Name"];
    object age = reader["Price"];
}
```

В цьому випадку результат буде аналогічним.

10.3 Визначення контексту даних і моделей

LINQ to SQL – технологія доступу і управління реляційними даними. Дана технологія дає змогу складати запити до БД у зручній формі у за допомогою операторів LINQ, які потім трансформуються в sql-вирази. Ключовими об'єктами тут є сутності, які зберігаються в базі даних, контекст даних і запит LINQ.

Щоб задіяти LINQ to SQL в проєкті, нам треба додати бібліотеку `System.Data.Linq.dll`. Для цього нам потрібно в оглядачі рішень натиснути

правою кнопкою миші на гілці Посилання і вибрати з контекстного меню Додати посилання на System.Data.Linq (рис. 10.10).

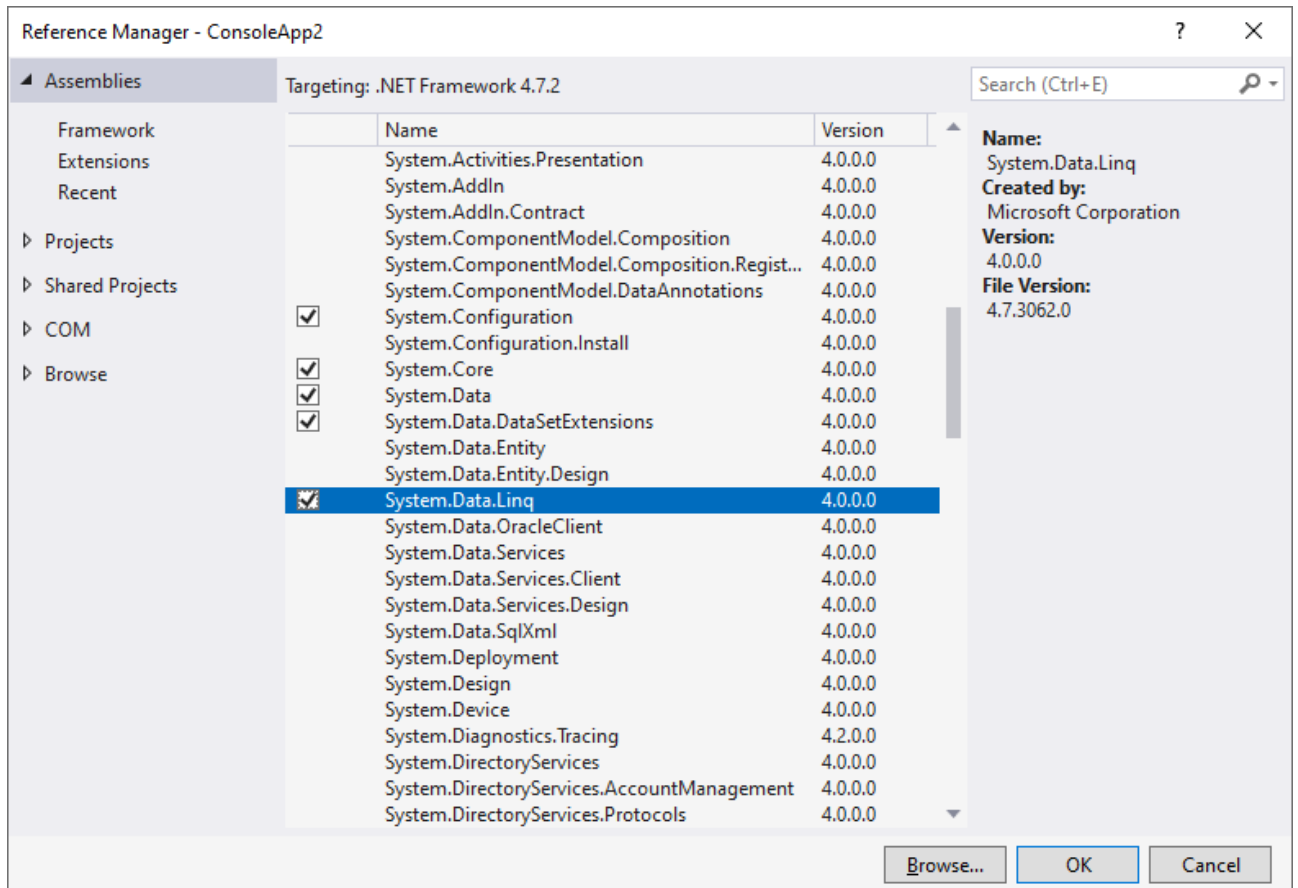


Рис. 10.10 Посилання на System.Data.Linq

Для взаємодії з базою даних в LINQ to SQL використовуються моделі і контекст даних. Моделі – це звичайні класи, які зіставляються з однією з таблиць в базі даних.

Нехай ми маємо базу check, в якій визначена таблиця goods (Рис. 10.11)

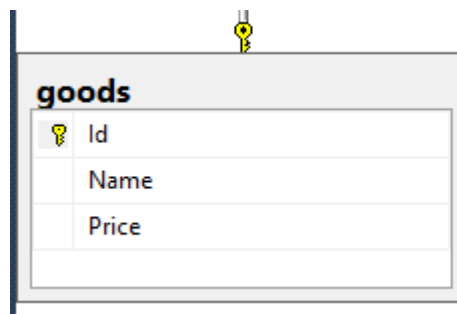


Рис. 10.11 Таблиця goods на діаграмі бази даних check

Спочатку визначимо сутність для роботи з даними цієї таблиці:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Linq.Mapping;

namespace ConsoleApp20
{
    [Table(Name = "goods")]
    Ссылка: 2
    public class goods
    {
        [Column(IsPrimaryKey = true, IsDbGenerated = true)]
        ссылка: 1
        public int Id { get; set; }
        [Column(Name = "Name")]
        ссылка: 1
        public string Name { get; set; }

        [Column]
        ссылка: 1
        public float Price { get; set; }
    }
}

```

Атрибут [Table] дає змогу виконати зіставлення таблиці з БД з цією моделлю. Однак за замовчуванням модель повинна називатися так само, як і таблиця, тобто goods. Тому, щоб зіставлення пройшло успішно, в атрибуті Table вказується властивість Name з ім'ям таблиці.

Кожна властивість моделі відповідає стовпцю за типом даних. Щоб відбувалося автоматичне співставлення, над властивостями застосовується атрибут [Column]. Без цього атрибута система не зможе зв'язати стовпчик з таблиці з властивістю.

Даний атрибут має ряд властивостей, за допомогою якого можна налаштувати зіставлення стовпців:

- AutoSync – вказує, як треба витягувати значення стовпця після вставки або поновлення;
- CanBeNull – вказує, чи може стовпець приймати значення null.
- DbType – визначає тип стовпчика. Вказується, якщо треба створити нову базу даних;
- Expression – зберігає вираз, який буде використовуватися для обчислення значення властивості;
- IsPrimaryKey – зберігає логічне значення і вказує, чи виконує стовпець роль первинного ключа (як в даному випадку Id);

- **IsDbGenerated** – зберігає логічне значення, яке вказує, чи буде значення стовпця генеруватися самою БД;
- **IsDiscriminator** – вказує, чи буде стовпець розмежувачем в системі успадкування класів;
- **IsVersion** – вказує, чи буде стовпець зберігати номер версії рядка або значення timestamp, яке вказує на час останньої зміни рядка;
- **Name** – задає ім'я стовпця, з яким буде зіставлятися ця властивість;
- **Storage** – вказує на ім'я приватної змінної, яка буде зберігати значення даного стовпця;
- **UpdateCheck** – визначає, як LINQ to SQL буде вирішувати проблему паралелізму. Якщо в моделі немає властивостей зі значення **IsVersion = true**, то операції з даними БД буде порівнювати значення рядків з таблиці з новими значеннями.

У класі програми визначимо такий код:

```

using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data.Linq;

namespace ConsoleApp20
{
    class Program
    {
        static string connectionString = @"Data Source=. \SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";
        static void Main(string[] args)
        {
            DataContext db = new DataContext(connectionString);

            // Отримуємо таблицю товари
            Table<goods> goods = db.GetTable<goods>();

            foreach (var good in goods)
            {
                Console.WriteLine("{0} \t{1} \t{2}", good.Id, good.Name, good.Price);
            }

            Console.Read();
        }
    }
}

```

Результат:

C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

```

1      Пальто ж?н. синє          2500
2      Пальто ж?н. зимнє, дем?сизонне  4500
3      Пальто ж?н. чорне        12400
4      Пальто ун?секс. чорне     5000
5      Пальто чол. кл?тинка     12500
6      Пальто чол. чорне        6500
10     Ball      100

```

10.4 Фільтрація і сортування

Щоб застосувати фільтрацію і сортування в LINQ to SQL, скористаємося стандартними операторами і виразами LINQ. Наприклад, виведемо всі товари, у яких ціна більша ніж 100 грн. і впорядкуємо їх за назвою:

```
using System;
using System.Collections.Generic;

using System.Data.SqlClient;
using System.Data.Linq;
using System.Linq;

namespace ConsoleApp20
{
    class Program
    {
        static string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";
        static void Main(string[] args)
        {
            DataContext db = new DataContext(connectionString);

            var query = db.GetTable<goods>().Where(u => u.Price > 100).OrderBy(u => u.Name);
            foreach (var good in query)
            {
                Console.WriteLine("{0} \t{1} \t{2}", good.Id, good.Name, good.Price);
            }

            Console.Read();
        }
    }
}
```

Результат:

C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

2	Пальто ж?н. зимнє, дем?сизонне	4500
1	Пальто ж?н. синє	2500
3	Пальто ж?н. чорне	12400
4	Пальто ун?секс. чорне	5000
5	Пальто чол. кл?тинка	12500
6	Пальто чол. чорне	6500

10.5. Зміна об'єктів в LINQ to SQL

Під час отримання об'єктів з бази даних в контекст DataContext ці об'єкти кешуються і у них встановлюється стан Unchanged. Якщо змінимо значення властивостей якого-небудь об'єкта з отриманого набору, то DataContext для цього об'єкта створює копію зі зміненими значеннями і встановлює у неї статус ToBeUpdated.

При виклику методу `SubmitChanges()` контекст даних звіряє значення оригінального об'єкта і його зміненої копії. І якщо два об'єкти відрізняються, то створюється sql-вираз `UPDATE`, за допомогою якого відбувається оновлення об'єкта в базі даних.

Для прикладу отримаємо перший об'єкт з бази даних, змінимо його і зробимо збереження:

```
using System;
using System.Collections.Generic;

using System.Data.SqlClient;
using System.Data.Linq;
using System.Linq;

namespace ConsoleApp20
{
    class Program
    {
        static string connectionString = @"Data Source=.\\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";

        static void Main(string[] args)
        {
            DataContext db = new DataContext(connectionString);

            Console.WriteLine("До оновлення");
            foreach (var good in db.GetTable<goods>().Take(5))
            {
                Console.WriteLine("{0} \t{1} \t{2}", good.Id, good.Name, good.Price);
            }
            Console.WriteLine();

            // беремо перший товар
            goods good1 = db.GetTable<goods>().FirstOrDefault();
            // і змінимо його ціну
            good1.Price = 12321;
            // збережемо зміни
            db.SubmitChanges();

            Console.WriteLine();
            Console.WriteLine("Після оновлення");
            foreach (var good in db.GetTable<goods>().Take(5))
            {
                Console.WriteLine("{0} \t{1} \t{2}", good.Id, good.Name, good.Price);
            }

            Console.Read();
        }
    }
}
```

Результат:

C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

```
До оновлення
1   Пальто ж?н. синє           2500
2   Пальто ж?н. зимнє, дем?сизонне  4500
3   Пальто ж?н. чорне         12400
4   Пальто ун?секс. чорне     5000
5   Пальто чол. кл?тинка     12500

Після оновлення
1   Пальто ж?н. синє           12321
2   Пальто ж?н. зимнє, дем?сизонне  4500
3   Пальто ж?н. чорне         12400
4   Пальто ун?секс. чорне     5000
5   Пальто чол. кл?тинка     12500
```

10.6 Додавання об'єктів в LINQ to SQL

Щоб додати новий об'єкт в базу даних, необхідно викликати у таблиці в контексті даних метод `InsertOnSubmit()` або `InsertAllOnSubmit()` і, за потреби, додати список об'єктів:

```
using System;
using System.Collections.Generic;

using System.Data.SqlClient;
using System.Data.Linq;
using System.Linq;

namespace ConsoleApp20
{
    class Program
    {
        static string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";
        static void Main(string[] args)
        {
            DataContext db = new DataContext(connectionString);

            Console.WriteLine("До додавання");
            foreach (var good in db.GetTable<goods>().Take(5))
            {
                Console.WriteLine("{0} \t{1} \t{2}", good.Id, good.Name, good.Price);
            }
            Console.WriteLine();

            // Створюємо новий товар
            goods good1 = new goods { Name = "Bag", Price = 1000 };
            // додаємо його в таблицю товарів
            db.GetTable<goods>().InsertOnSubmit(good1);
            db.SubmitChanges();

            Console.WriteLine();
            Console.WriteLine("Після додавання");
            foreach (var good in db.GetTable<goods>().OrderByDescending(u => u.Id).Take(6))
            {
                Console.WriteLine("{0} \t{1} \t{2}", good.Id, good.Name, good.Price);
            }

            Console.Read();
        }
    }
}
```

Результат:

```
C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

До додавання
1   Пальто ж?н. синє          12321
2   Пальто ж?н. зимнє, дем?сизонне  4500
3   Пальто ж?н. чорне         12400
4   Пальто ун?секс. чорне     5000
5   Пальто чол. кл?тинка     12500

П?сля додавання
12   Bag      1000
11   Bag      1000
10   Ball     100
6    Пальто чол. чорне       6500
5    Пальто чол. кл?тинка    12500
4    Пальто ун?секс. чорне   5000
```

10.7 Видалення об'єктів в LINQ to SQL

Для видалення об'єкта з бази даних застосовується метод `DeleteOnSubmit()` або `DeleteAllOnSubmit()`, якщо видаляється список об'єктів (рис. 10.12)

```
Ссылка: 0
class Program
{
    static string connectionString = @"Data Source=. \SQLEXPRESS;Initial Catalog=check; User Id =sa; Password =140317;";
    Ссылка: 0
    static void Main(string[] args)
    {
        DataContext db = new DataContext(connectionString);

        // отримаємо останній об'єкт для видалення
        var good1 = db.GetTable<goods>().OrderByDescending(u => u.Id).FirstOrDefault();

        if (good1 != null)
        {
            Console.WriteLine("Об'єкт, що видаляється:");
            Console.WriteLine("Id: {0}", good1.Id);
            Console.WriteLine("Назва: {0}", good1.Name);
            Console.WriteLine("Ціна: {0}", good1.Price);

            db.GetTable<goods>().DeleteOnSubmit(good1);
            db.SubmitChanges();
            Console.WriteLine("Об'єкт видалено");
        }

        Console.Read();
    }
}
```

```
C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

Об'єкт, що видаляється:
Id: 12
Назва: Bag
Ціна: 1000
Об'єкт видалено
```

Рис. 10.12 Видалення об'єктів з таблиці goods

10.8 Платформа Entity Framework

Entity Framework – спеціальна об'єктно-орієнтована технологія на базі фреймворка .NET для роботи з даними. Якщо традиційні засоби ADO.NET дають змогу створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework має більш високий рівень абстракції, який дає змогу абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні оперують таблицями, індексами, первинними і зовнішніми ключами, то на концептуальному рівні, який пропонує Entity Framework, вже працюють з об'єктами.

Центральною концепцією Entity Framework є поняття сутності або entity. Сутність – це набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх наборами.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людину, то можливо виділити такі властивості, як ім'я, прізвище, зріст, вік, вага. Властивості необов'язково є простими даними типу `int`, а й можуть представляти більш комплексні структури даних. І у кожної сутності може бути одна або декілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

Сутності можуть бути пов'язані асоціативним зв'язком один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо не тільки отримувати певні рядки, що зберігають об'єкти, з БД, а й отримувати об'єкти, пов'язані різними асоціативними зв'язками.

Іншим ключовим поняттям є Entity Data Model. Ця модель зіставляє класи сутностей з реальними таблицями в БД.

Entity Data Model складається з трьох рівнів: концептуального, рівень сховища і рівень зіставлення (маппінга). На концептуальному рівні

відбувається визначення класів сутностей, які використовуються в застосунку. Рівень сховища визначає таблиці, стовпці, відносини між таблицями і типи даних, з якими порівнюється використовувана база даних. Рівень зіставлення (мапінга) служить посередником між попередніми двома поняттями, визначаючи зіставлення між властивостями класу сутності і стовпцями таблиць.

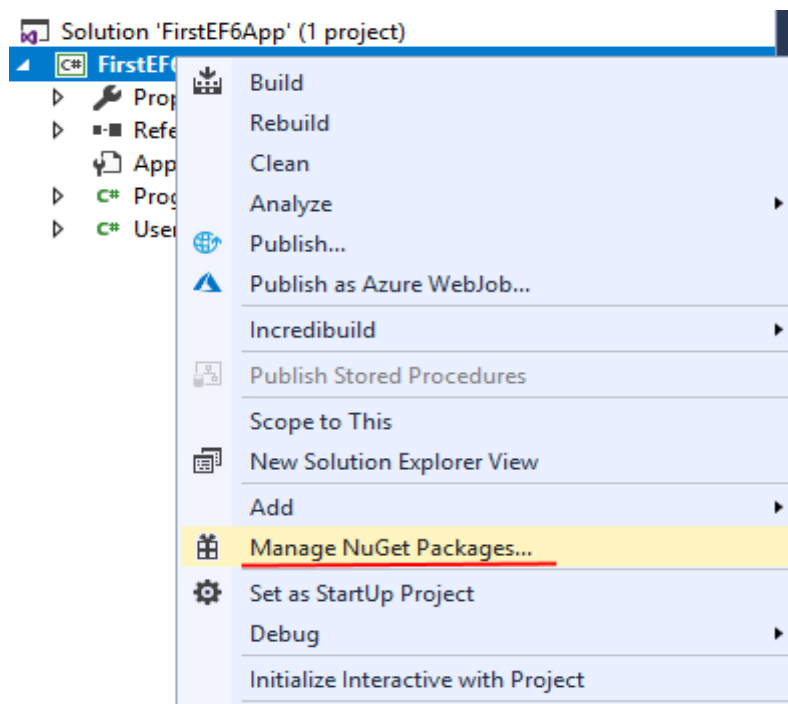
Таким чином, через класи, визначені у застосунку, можливо взаємодіяти з таблицями бази даних.

Entity Framework передбачає три можливі способи взаємодії з БД:

- Database first – Entity Framework створює набір класів, які відображають модель конкретної бази даних;
- Model first – спочатку розробник створює модель бази даних, за якою потім Entity Framework створює саму базу даних на сервері;
- Code first – розробник створює клас моделі даних, які будуть зберігатися в БД, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці.

10.9. Встановлення пакету EF6

Щоб додати пакет EF6, натиснемо на проєкт в Explorer Solution правою кнопкою миші і виберемо в контекстному меню Manage NuGet Packages ...:



Потім у вікні управління NuGet-пакетами в вікні пошуку введемо слово "Entity" і виберемо пакет Entity Framework і встановимо його ().

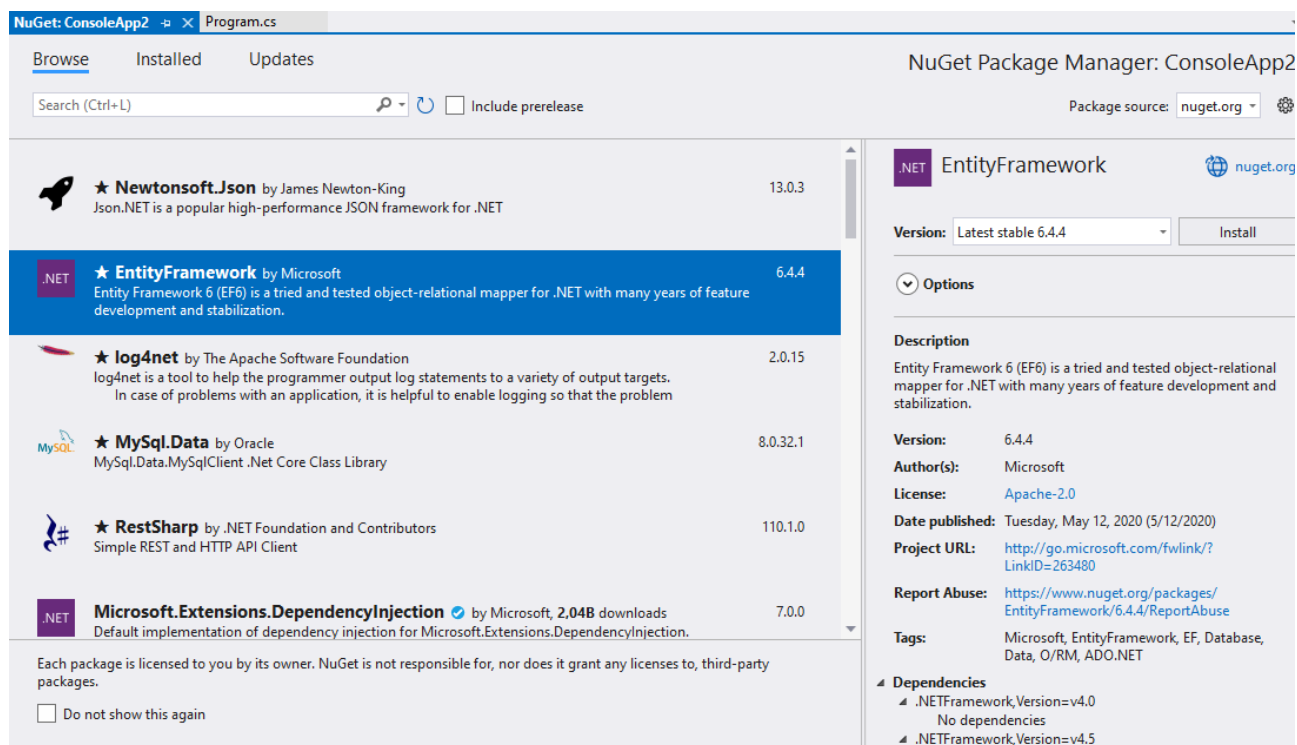


Рис. 10.13 Вікно управління NuGet-пакетами – встановлення Entity Framework

Основу функціональності Entity Framework складають класи, які знаходяться в просторі імен System.Data.Entity. Серед усього набору класів цього простору імен слід виділити такі:

- DbContext – визначає контекст даних, який використовується для взаємодії з базою даних.
- DbModelBuilder – зіставляє класи на мові C # з сутностями в базі даних.
- DbSet / DbSet <TEntity> – представляє набір сутностей, які зберігаються в базі даних

У будь-якому застосунку, який працює з БД через Entity Framework, потрібен контекст (клас похідний від DbContext) і набір даних DbSet, через який можливо взаємодіяти з таблицями БД. В цьому разі таким контекстом буде клас goodsContext.

У конструкторі цього класу викликається конструктор базового класу, в який передається рядок "DbConnection" – це ім'я майбутнього рядка підключення до бази даних.

10.10 Підходи Code First до існуючої бази даних

У Solution Explorer знайдемо файл App.config та відкриємо його. Перед закриттям тегом </configuration> додамо нову секцію connectionStrings:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <entityFramework>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=.\\SQLEXPRESS;Initial Catalog=check; User Id=sa; Password=140317;"
        providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Визначимо класи моделі даних та контексту. Додамо клас моделі goods:

```
Ссылка: 2
public class goods
{
    Ссылка: 2
    public int Id { get; set; }
    Ссылка: 1
    public string Name { get; set; }
    Ссылка: 1
    public float Price { get; set; }
}
```

І також додамо клас контексту даних:

```
Ссылка: 3
public class goodsContext : DbContext
{
    Ссылка: 1
    public goodsContext() :
        base("DefaultConnection")
    { }

    Ссылка: 1
    public DbSet<goods> Goods { get; set; }
}
```

У конструкторі контексту даних передаємо конструктор базового класу ім'я рядка підключення з файлу конфігурації App.config.

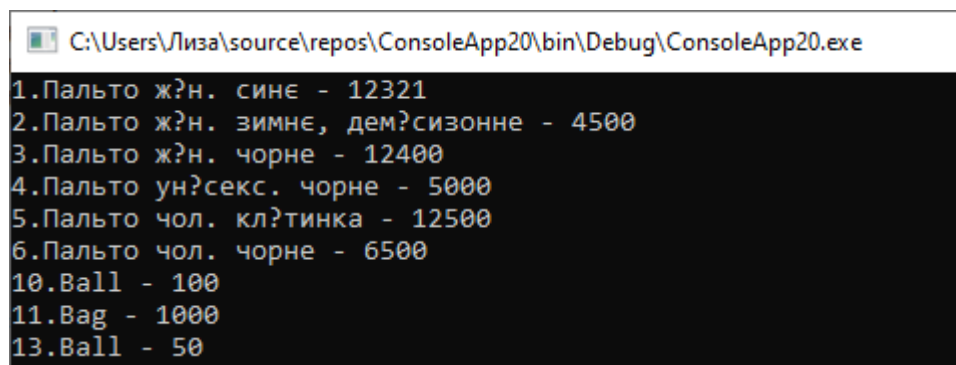
Для отримання даних визначимо такий код:

```
using System;
using System.Linq;
using System.Data.Entity;

namespace ConsoleApp20
{
    class Program
    {
        static void Main(string[] args)
        {
            using (goodsContext db = new goodsContext())
            {
                var Goods1 = db.Goods;
                foreach (goods u in Goods1)
                {
                    Console.WriteLine("{0}.{1} - {2}", u.Id, u.Name, u.Price);
                }
            }

            Console.Read();
        }
    }
}
```

Результат:



C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

```
1.Пальто ж?н. синє - 12321
2.Пальто ж?н. зимнє, дем?сизонне - 4500
3.Пальто ж?н. чорне - 12400
4.Пальто ун?секс. чорне - 5000
5.Пальто чол. кл?тинка - 12500
6.Пальто чол. чорне - 6500
10.Ball - 100
11.Bag - 1000
13.Ball - 50
```

10.11 Додавання об'єктів через DbSet

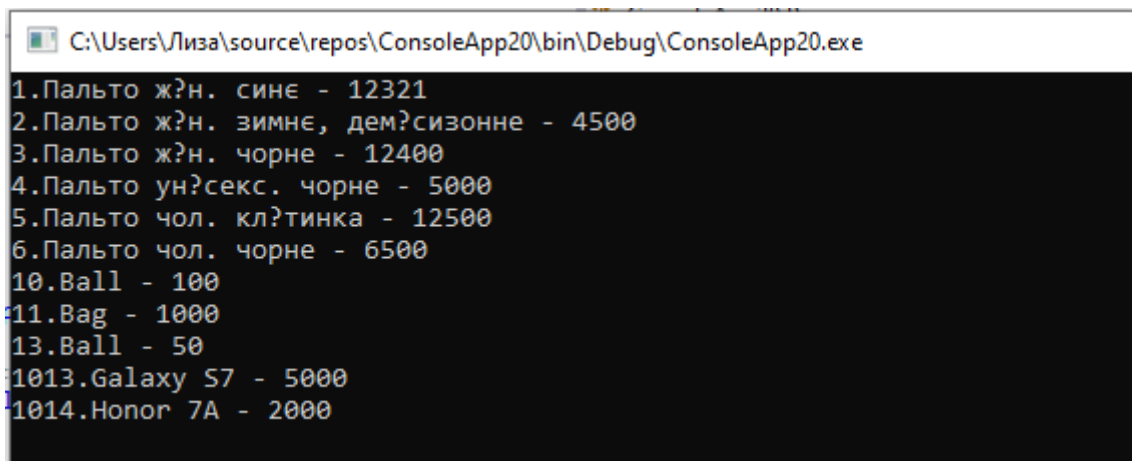
Для додавання застосовується метод Add() об'єкта DbSet:

Ссылка: 0

```
static void Main(string[] args)
{
    using (goodsContext db = new goodsContext())
    {
        goods Good1= new goods { Name = "Galaxy S7", Price = 5000 };
        goods Good2 = new goods { Name = "Honor 7A", Price = 2000 };
        // додавання
        db.Goods.Add(Good1);
        db.Goods.Add(Good2);
        db.SaveChanges(); //Збереження змін
        var Goods1 = db.Goods;
        foreach (goods u in Goods1)
        {
            Console.WriteLine("{0}.{1} - {2}", u.Id, u.Name, u.Price);
        }
    }

    Console.Read();
}
```

Результат:



```
C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe
1.Пальто ж?н. синє - 12321
2.Пальто ж?н. зимнє, дем?сизонне - 4500
3.Пальто ж?н. чорне - 12400
4.Пальто ун?секс. чорне - 5000
5.Пальто чол. кл?тинка - 12500
6.Пальто чол. чорне - 6500
10.Ball - 100
11.Bag - 1000
13.Ball - 50
1013.Galaxy S7 - 5000
1014.Honor 7A - 2000
```

10.12 Редагування об'єктів через DbSet

Контекст даних здатний відстежувати зміни об'єктів, тому для редагування об'єкта достатньо змінити його властивості і після цього викликати метод `SaveChanges()`:

Ссылка: U

```
static void Main(string[] args)
{
    using (goodsContext db = new goodsContext())
    {
        // отримуємо перший об'єкт
        goods F = db.Goods.Find(1);

        F.Price = 3000;
        db.SaveChanges(); // зберігаємо зміни
    }

    var Goods1 = db.Goods;
    foreach (goods u in Goods1)
    {
        Console.WriteLine("{0}.{1} - {2}", u.Id, u.Name, u.Price);
    }

    Console.Read();
}
```

Результат:

 C:\Users\Лиза\source\repos\ConsoleApp20\bin\Debug\ConsoleApp20.exe

```
1.Пальто ж?н. синє - 3000
2.Пальто ж?н. зимнє, дем?сизонне - 4500
3.Пальто ж?н. чорне - 12400
4.Пальто ун?секс. чорне - 5000
5.Пальто чол. кл?тинка - 12500
6.Пальто чол. чорне - 6500
10.Ball - 100
11.Bag - 1000
13.Ball - 50
1013.Galaxy S7 - 5000
1014.Honor 7A - 2000
```

10.13 Видалення об'єктів через DbSet

Для видалення об'єкта застосовується метод `Remove()` об'єкта `DbSet`.
Видалимо запис, який містить значення «1014» з таблиці `goods ()`

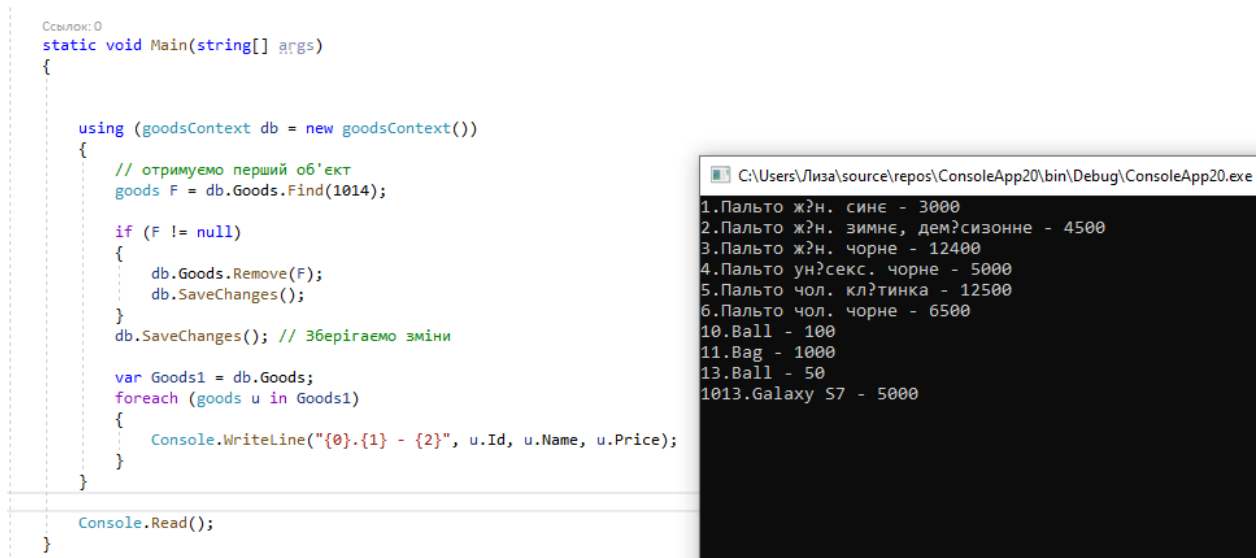


Рис. 10.14 Видалення запису з таблиці goods

ЗАВДАННЯ

Варіант	Опис завдання
1	Розробити програмний застосунок для своєї БД, використавши ADO.NET. Розробити зручний інтерфейс.
2	Розробити програмний застосунок для своєї БД для ОС Windows. Розробити зручний інтерфейс.
3	Розробити програмний застосунок для своєї БД, використавши LINQ. Розробити зручний інтерфейс.
4	Розробити програмний застосунок для своєї БД, використавши EF. Розробити зручний інтерфейс.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Технологія ADO.NET. Недоліки і переваги.
2. Технологія LINQ. Недоліки і переваги.
3. Технологія Entity Framework. Недоліки і переваги.

ЛІТЕРАТУРА

1. Васильєв Д. О., Горлова Т. В. Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Організація баз даних та знань» для студентів денної форми навчання зі спеціальності 122 – «комп'ютерні науки» за освітньо-професійною програмою «комп'ютерні науки» освітнього ступеня «бакалавр». Кременчук: МОН України Кременчуць. нац. ун-т ім. Михайла Острогр., 2020. 58 с.

2. Методичні вказівки до лабораторної роботи «Створення баз даних за допомогою мови Transact SQL» з курсу «Проектування баз даних» для студентів спеціальності 122 Комп'ютерні науки та інформаційні технології, з курсу «Бази даних» спеціальності 186 Видавництво та поліграфічна справа та з курсу «Проектування інформаційних систем» спеціальності 124 Системний аналіз / уклад.: Л. Б. Кащєєв, С. В. Коваленко, О. С. Куценко – Харків : НТУ «ХП», 2018. – 16 с.

3. Погромська Г.С., Махровська Н.А. Бази даних: проектування та реалізація. – Миколаїв: МНУ ім. В.О. Сухомлинського, 2019. – 183 с.

4. Лосєв М. Ю. Бази даних : навчально-практичний посібник для самостійної роботи студентів [Електронний ресурс] / М. Ю. Лосєв, В. В. Федько. – Харків : ХНЕУ ім. С. Кузнеця, 2018. – 233 с. Доступ: http://repository.hneu.edu.ua/bitstream/123456789/21468/1/2018-Лосєв_М_Ю%2C_Федько_В_В.pdf

5. Рябцев В. В. Методичні рекомендації щодо забезпечення самостійної роботи студентів з дисципліни «Бази даних» (для бакалаврів спеціальностей 121 Інженерія програмного забезпечення та 122 Комп'ютерні науки та інформаційні технології). – К. : МАУП, 2018. – 24 с.

6. Програмування баз даних: конспект лекцій [Електронний ресурс] : навч. посіб. Для студ. спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка» / М. В. Добролюбова ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 22,7 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 275 с. Доступ: <https://ela.kpi.ua/handle/123456789/43918>

7. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Принципи проектування баз даних та баз знань" для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання : [Електронне видання] / уклад. О. К. Пандорін. – Х. : ХНЕУ ім. С. Кузнеця, 2015. – 44 с.

8. Гайна Г.А. Г12 Основи проектування баз даних: Навчальний посібник. – К.: КНУБА, 2005. – 204 с.

9. Мулеса О.Ю. Основи мови запитів SQL. – Ужгород, 2015. – 48 с.

10. Буяк Л.М., Мушак А.Я., Хома Н.Г. Працюємо з базами даних в середовищі Microsoft Office: теоретичні аспекти та приклади розв’язування задач Навчальний посібник з курсу „Сучасні інформаційні технології” для студентів денної та заочно-дистанційної форм навчання галузей знань 07 „Управління та адміністрування”, 23 „Соціальна робота” ступеня вищої освіти „бакалавр”. – Тернопіль: ТНЕУ, 2019. – 80 с.

11. Бардус І. О. Базы даних у схемах (на основі фундаменталізованого підходу) : навч. посіб. /І. О. Бардус, М. І. Лазарєв, А. О. Ніценко. – Харків: «Діса плюс», 2017. – 133 с.

12. Мулеса О.Ю. Інформаційні системи та реляційні бази даних. Навч.посібник. – Електронне видання, 2018. – 118 с.

13. Реляційна алгебра. Реляційне числення [Електронний ресурс]: методичні вказівки для підготовки до контрольної роботи з нормативних навчальних дисциплін “Базы даних та розподілені інформаційно-аналітичні системи”, “Організація баз даних та знань”/ В. В. Булатецький, Л. В. Булатецька; ВНУ ім. Лесі Українки. – Електронні текстові данні (1 файл: 992 КБ). – Луцьк : ВНУ ім. Лесі Українки, 2020. – 36 с.

14. SQL Tutorial. URL: https://www.w3schools.com/sql/sql_create_table.asp.

15. SQL Server. URL: <https://www.sqlservertutorial.net/>.

16. Educational SQL resources. URL: <https://learn.microsoft.com/en-us/sql/sql-server/educational-sql-resources?view=sql-server-ver16>.

17. Електронний ресурс. URL:

https://learn.ztu.edu.ua/pluginfile.php/204794/mod_resource/content/0/1r7.%202sem_Adm.pdf

18. Електронний ресурс. URL:

<https://www.sqlservertutorial.net/sql-server-administration/sql-server-create-role/>

19. Електронний ресурс. URL:

<https://learn.microsoft.com/en-us/sql/t-sql/statements/create-role-transact-sql?view=sql-server-ver16>