

## Лабораторна робота

Тема: Початок роботи з Microsoft SQL Server. Основи T-SQL. DML.

Ціль: Додавання даних. Команда INSERT. Вибір даних. Команда SELECT  
Сортування. ORDER BY. Вилучення діапазону рядків. Фільтрування. WHERE  
Оператори фільтрації. Оновлення даних. Команда UPDATE Видалення даних.  
Команда DELETE.

### Хід роботи

#### Додавання даних. Команда INSERT

Для додавання даних застосовується команда INSERT, яка має такий формальний синтаксис:

```
1  INSERT [INTO] table_name [(columns_list)] VALUES (value1, value2, ... valueN)
```

Спочатку йде вираз INSERT INTO, потім у дужках можна вказати список стовпців через кому, в які треба додавати дані, і в кінці після слова VALUES дужках перераховують значення, що додаються для стовпців.

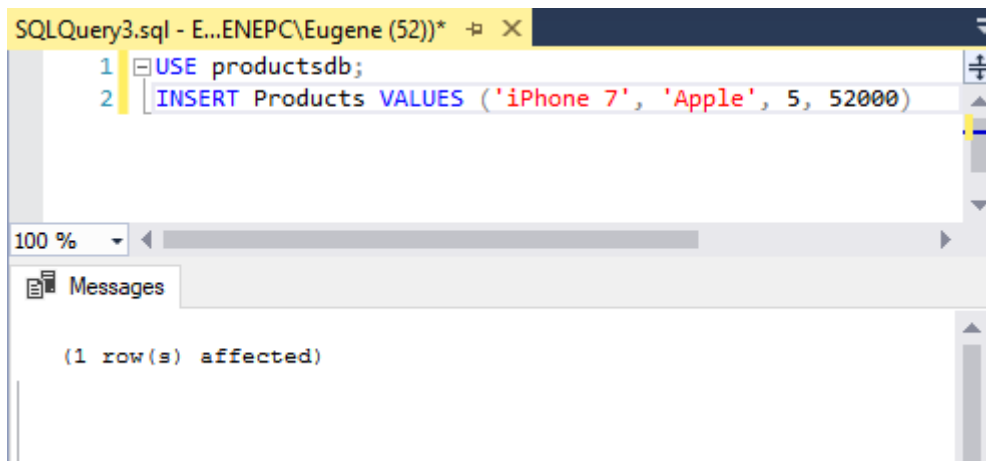
Наприклад, нехай раніше було створено таку базу даних:

```
1  CREATE DATABASE productsdb;  
2  GO  
3  USE productsdb;  
4  CREATE TABLE Products  
5  (  
6  ID INT IDENTITY PRIMARY KEY,  
7  ProductName NVARCHAR(30) NOT NULL,  
8  Manufacturer NVARCHAR(20) NOT NULL,  
9  ProductCount INT DEFAULT 0,  
10 Price MONEY NOT NULL  
11 )
```

Додамо до неї один рядок за допомогою команди INSERT:

```
1  INSERT Products VALUES ('iPhone 7', 'Apple', 5, 52000)
```

Після успішного виконання в SQL Server Management Studio у полі повідомлень має з'явитися повідомлення "1 row(s) affected":



Варто враховувати, що значення стовпців у дужках після ключового слова VALUES передаються по порядку їх оголошення. Наприклад, у виразі CREATE TABLE вище можна побачити, що першим стовпцем йде Id. Але оскільки для нього заданий атрибут IDENTITY, значення цього стовпця автоматично генерується, і його можна не вказувати. Другий стовпець представляє ProductName, тому перше значення - рядок iPhone 7 буде передано саме цьому стовпцю. Друге значення - рядок "Apple" буде передано третьому стовпцю Manufacturer і такі інші. Тобто значення передаються стовпцям в такий спосіб:

- ProductName: 'iPhone 7'
- Manufacturer: 'Apple'
- ProductCount: 5
- Price: 52000

Також при введенні значень можна вказати безпосередні стовпці, які будуть додаватися значення:

```
1 INSERT INTO Products (ProductName, Price, Manufacturer)
2 VALUES ('iPhone 6S', 41000, 'Apple')
```

Тут значення вказується лише трьох стовпців. Причому тепер значення передаються в порядку прямування стовпців:

- ProductName: 'iPhone 6S'
- Manufacturer: 'Apple'
- Price: 41000

Для вказаних стовпців (в даному випадку ProductCount) буде додаватися значення за замовчуванням, якщо заданий атрибут DEFAULT, або значення NULL. При цьому невказані стовпці повинні допускати значення NULL або мати атрибут DEFAULT.

Також ми можемо додати відразу кілька рядків:

```
1  INSERT INTO Products
2  VALUES
3  ('iPhone 6', 'Apple', 3, 36000),
4  ('Galaxy S8', 'Samsung', 2, 46000),
5  ('Galaxy S8 Plus', 'Samsung', 1, 56000)
```

В даному випадку до таблиці буде додано три рядки.

Також при додаванні ми можемо вказати, щоб для стовпця використовувалося значення за промовчанням за допомогою ключового слова DEFAULT або NULL:

```
1  INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)
2  VALUES ('Mi6', 'Xiaomi', DEFAULT, 28000)
```

У цьому випадку для стовпця ProductCount буде використано значення за замовчуванням (якщо воно встановлено, якщо його немає – то NULL).

Якщо всі стовпці мають атрибут DEFAULT, який визначає значення за замовчуванням, або допускають значення NULL, то для всіх стовпців можна вставити значення за замовчуванням:

```
1  INSERT INTO Products
2  DEFAULT VALUES
```

Але якщо брати таблицю Products, то подібна команда завершиться помилкою, оскільки кілька полів немає атрибуту DEFAULT і навіть допускають значення NULL.

## **Вибір даних. Команда SELECT**

Для отримання даних використовується команда SELECT. У спрощеному вигляді вона має наступний синтаксис:

```
1  SELECT columns_list FROM table_name
```

Наприклад, нехай раніше була створена таблиця Products, і до неї додані деякі початкові дані:

```

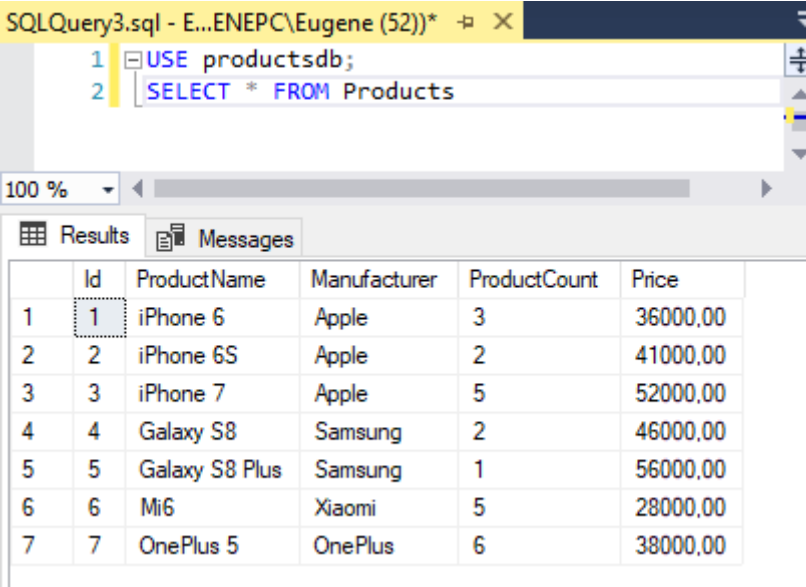
1  CREATE TABLE Products
2  (
3  ID INT IDENTITY PRIMARY KEY,
4  ProductName NVARCHAR(30) NOT NULL,
5  Manufacturer NVARCHAR(20) NOT NULL,
6  ProductCount INT DEFAULT 0,
7  Price MONEY NOT NULL
8  );
9
10 INSERT INTO Products
11 VALUES
12 ('iPhone 6', 'Apple', 3, 36000),
13 ('iPhone 6S', 'Apple', 2, 41000),
14 ('iPhone 7', 'Apple', 5, 52000),
15 ('Galaxy S8', 'Samsung', 2, 46000),
16 ('Galaxy S8 Plus', 'Samsung', 1, 56000),
17 ('Mi6', 'Xiaomi', 5, 28000),
18 ('OnePlus 5', 'OnePlus', 6, 38000)

```

Отримаємо всі об'єкти з цієї таблиці:

```
1  SELECT * FROM Products
```

Символ зірочка\* вказує, що нам треба отримати усі стовпці.



SQLQuery3.sql - E...ENEPC\Eugene (52)\*

```

1  USE productsdb;
2  SELECT * FROM Products

```

100 %

Results Messages

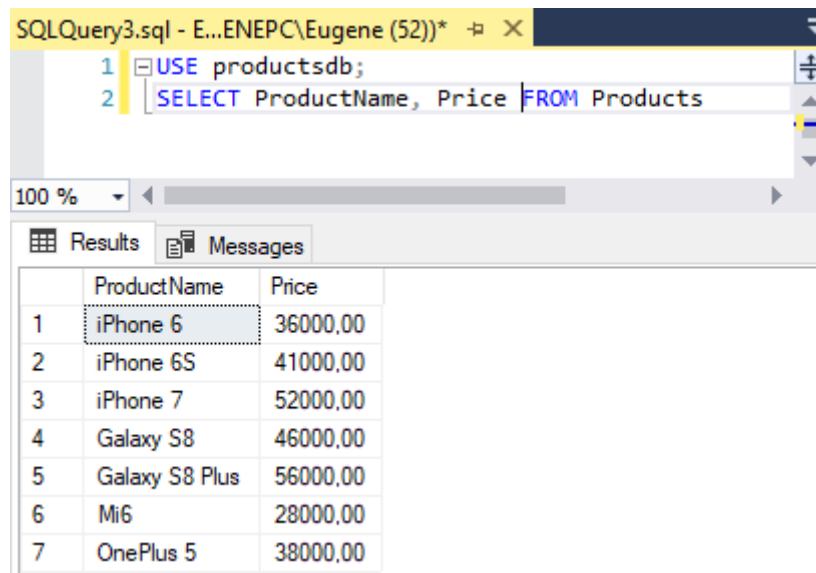
	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

Отримання всіх стовпців за допомогою символу зірочки вважається не дуже гарною практикою, так як, як правило, не всі стовпці бувають потрібні. І більш оптимальний підхід полягає у вказівці всіх необхідних імпальт після слова SELECT. Виняток становить той випадок, коли треба отримати дані по всіх

стовпцях таблиці. Також використання символу \* може бути переважно в таких ситуаціях, коли точно не відомі назви стовпців.

Якщо нам треба отримати дані не по всіх, а по якихось конкретних стовпцях, то всі ці специфікації стовпців перераховуються через кому після SELECT:

```
1 SELECT ProductName, Price FROM Products
```



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

```
1 USE productsdb;  
2 SELECT ProductName, Price FROM Products
```

Below the query window, the 'Results' tab is active, displaying a table with 7 rows and 2 columns: ProductName and Price.

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

Специфікація стовпця необов'язково має представляти його назву. Це може бути будь-який вираз, наприклад, результат арифметичної операції. Так, виконаємо наступний запит:

```
1 SELECT ProductName + '(' + Manufacturer + ')', Price, Price * ProductCount  
2 FROM Products
```

Тут при вибірці створюватимуться три стовпці. Перший стовпець є результатом об'єднання двох стовпців ProductName і Manufacturer. Другий стовпець – стандартний стовпець Price. А третій стовпець представляє значення стовпця Price, помножене значення стовпця ProductCount.

SQLQuery3.sql - E...ENEPC\Eugene (52))\* X SQLQuery2.sql - E...ENEPC\Eugene (51))\*

```

1 USE productsdb;
2
3 SELECT ProductName + ' (' + Manufacturer + ')', Price, Price * ProductCount
4 FROM Products

```

100 %

Results Messages

	(No column name)	Price	(No column name)
1	iPhone 6 (Apple)	36000,00	108000,00
2	iPhone 6S (Apple)	41000,00	82000,00
3	iPhone 7 (Apple)	52000,00	260000,00
4	Galaxy S8 (Samsung)	46000,00	92000,00
5	Galaxy S8 Plus (Samsung)	56000,00	56000,00
6	Mi6 (Xiaomi)	28000,00	140000,00
7	OnePlus 5 (OnePlus)	38000,00	228000,00

За допомогою оператора AS можна змінити назву вихідного стовпця або визначити його псевдонім:

```

1 SELECT
2   ProductName + '(' + Manufacturer + ')' AS ModelName,
3   Price,
4   Price * ProductCount AS TotalSum
5 FROM Products

```

У разі результатом вибірки є дані з 3-м стовпцем. Перший стовпець ModelName поєднує стовпці ProductName і Manufacturere, другий представляє стандартний стовпець Price. Третій стовпець TotalSum зберігає твір стовпців ProductCount та Price. При цьому, як у випадку зі стовпцем Price, необов'язково визначати назву результуючого стовпця за допомогою AS.

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT
4 ProductName + ' (' + Manufacturer + ')' AS ModelName,
5 Price,
6 Price * ProductCount AS TotalSum
7 FROM Products

```

100 %

Results Messages

	ModelName	Price	TotalSum
1	iPhone 6 (Apple)	36000,00	108000,00
2	iPhone 6S (Apple)	41000,00	82000,00
3	iPhone 7 (Apple)	52000,00	260000,00
4	Galaxy S8 (Samsung)	46000,00	92000,00
5	Galaxy S8 Plus (Samsung)	56000,00	56000,00
6	Mi6 (Xiaomi)	28000,00	140000,00
7	OnePlus 5 (OnePlus)	38000,00	228000,00

## DISTINCT

Оператор DISTINCT дозволяє вибрати унікальні рядки. Наприклад, у нашому випадку в таблиці може бути по кілька товарів від тих самих виробників.

Виберемо всіх виробників:

```

1 SELECT DISTINCT Manufacturer
2 FROM Products

```

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT DISTINCT Manufacturer
4 FROM Products

```

100 %

Results Messages

	Manufacturer
1	Apple
2	OnePlus
3	Samsung
4	Xiaomi

У разі критерієм розмежування рядків є стовпець Manufacturer. Тому в результуючій вибірці будуть лише унікальні значення Manufacturer. І якщо, наприклад, у базі даних є два товари з виробником Apple, то ця назва зустрічатиметься в результуючій вибірці лише один раз.

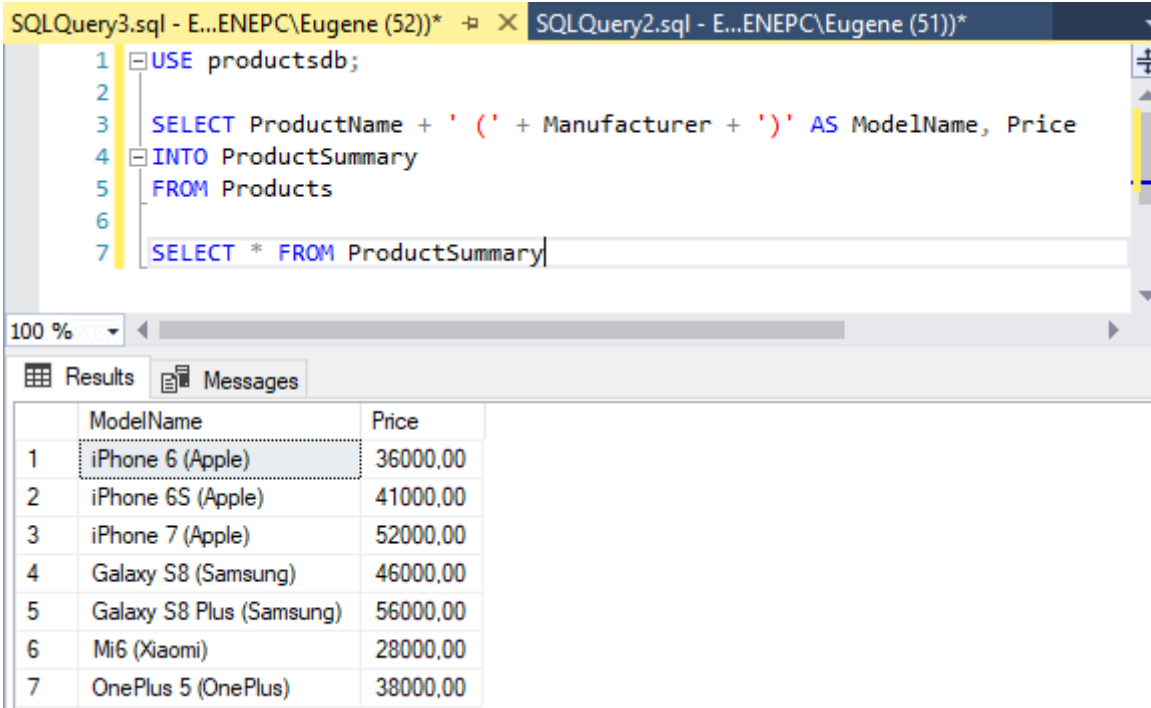
Вибірка з додаванням

### *SELECT INTO*

Вираз `SELECT INTO` дозволяє вибрати з однієї таблиці деякі дані до іншої таблиці, при цьому друга таблиця створюється автоматично. Наприклад:

```
1  SELECT ProductName + '(' + Manufacturer + ')' AS ModelName, Price
2  INTO ProductSummary
3  FROM Products
4
5  SELECT * FROM ProductSummary
```

Після виконання цієї команди у базі даних буде створено ще одну таблицю `ProductSummary`, яка матиме два стовпці `ModelName` і `Price`, а дані для цих стовпців будуть взяті з таблиці `Products`:



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following SQL code:

```
1 USE productsdb;
2
3 SELECT ProductName + '(' + Manufacturer + ')' AS ModelName, Price
4 INTO ProductSummary
5 FROM Products
6
7 SELECT * FROM ProductSummary
```

The bottom pane shows the results of the query in a table with two columns: `ModelName` and `Price`. The table contains seven rows of data:

	ModelName	Price
1	iPhone 6 (Apple)	36000,00
2	iPhone 6S (Apple)	41000,00
3	iPhone 7 (Apple)	52000,00
4	Galaxy S8 (Samsung)	46000,00
5	Galaxy S8 Plus (Samsung)	56000,00
6	Mi6 (Xiaomi)	28000,00
7	OnePlus 5 (OnePlus)	38000,00

При виконанні цієї команди таблиця, до якої йде вибірка (у разі `ProductSummary`), має існувати у базі даних.

Але, припустимо, ми потім вирішили додати всі дані з таблиці `Products` до вже існуючої таблиці `ProductSummary`. У цьому випадку можна знову ж таки використовувати команду `INSERT`:

```
1  INSERT INTO ProductSummary
2  SELECT ProductName + '(' + Manufacturer + ')' AS ModelName, Price
3  FROM Products
```



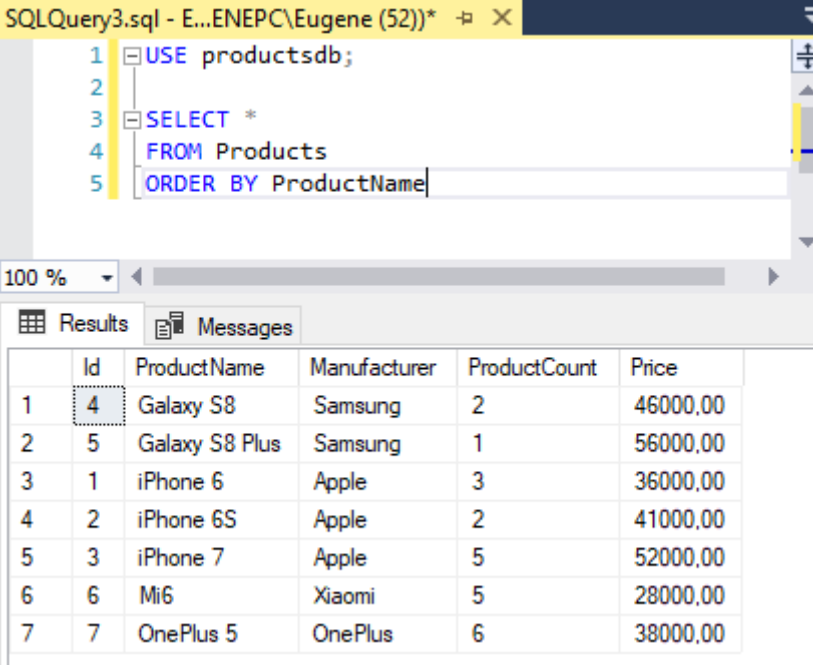
Тут значення, що додаються, фактично представляють результат вибірки з таблиці Products.

## Сортування. ORDER BY

Оператор ORDER BY дозволяє відсортувати видобуті значення за певним стовпцем:

```
1  SELECT *
2  FROM Products
3  ORDER BY ProductName
```

У цьому випадку рядки сортуються за зростанням значення стовпця ProductName:



SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```
1  USE productsdb;
2
3  SELECT *
4  FROM Products
5  ORDER BY ProductName
```

100 %

Results Messages

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

Сортування також можна проводити за псевдонімом стовпця, який визначається за допомогою оператора AS:

```
1  SELECT ProductName, ProductCount * Price AS TotalSum
2  FROM Products
3  ORDER BY TotalSum
```

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT ProductName, ProductCount * Price AS TotalSum
4 FROM Products
5 ORDER BY TotalSum

```

100 %

Results Messages

	ProductName	TotalSum
1	Galaxy S8 Plus	56000,00
2	iPhone 6S	82000,00
3	Galaxy S8	92000,00
4	iPhone 6	108000,00
5	Mi6	140000,00
6	OnePlus 5	228000,00
7	iPhone 7	260000,00

За умовчанням застосовується сортування за зростанням. За допомогою додаткового оператора DESC можна задати сортування за спаданням.

```

1 SELECT ProductName
2 FROM Products
3 ORDER BY ProductName DESC

```

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT ProductName
4 FROM Products
5 ORDER BY ProductName DESC

```

100 %

Results Messages

	ProductName
1	OnePlus 5
2	Mi6
3	iPhone 7
4	iPhone 6S
5	iPhone 6
6	Galaxy S8 Plus
7	Galaxy S8

За замовчуванням замість DESC використовується оператор ASC:

```

1 SELECT ProductName
2 FROM Products
3 ORDER BY ProductName ASC

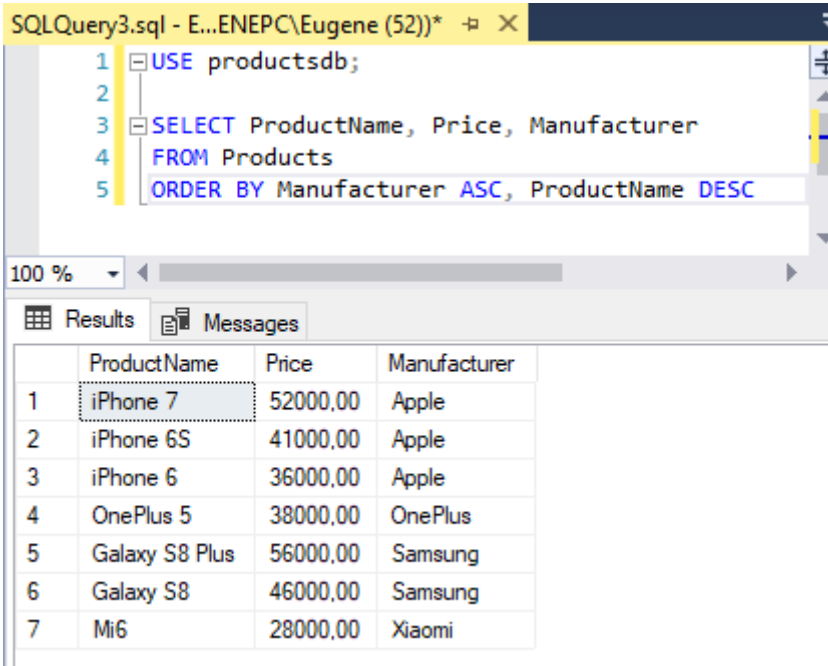
```

Якщо необхідно відсортувати відразу по кількох стовпцях, всі вони перераховуються після оператора ORDER BY:

```
1 SELECT ProductName, Price, Manufacturer
2 FROM Products
3 ORDER BY Manufacturer, ProductName
```

У цьому випадку спочатку рядки сортуються за стовпцем Manufacturer за зростанням. Потім, якщо є два рядки, в яких стовпець Manufacturer має однакове значення, то вони сортуються за стовпцем ProductName також за зростанням. Але знову ж таки за допомогою ASC і DESC можна окремо для різних стовпців визначити сортування за зростанням та зменшенням:

```
1 SELECT ProductName, Price, Manufacturer
2 FROM Products
3 ORDER BY Manufacturer ASC, ProductName DESC
```



SQLQuery3.sql - E...\ENEPC\Eugene (52))

```
1 USE productsdb;
2
3 SELECT ProductName, Price, Manufacturer
4 FROM Products
5 ORDER BY Manufacturer ASC, ProductName DESC
```

100 %

Results Messages

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

Як критерій сортування також можна використовувати складно вираз на основі стовпців:

```
1 SELECT ProductName, Price, ProductCount
2 FROM Products
3 ORDER BY ProductCount * Price
```

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT ProductName, Price, ProductCount
4 FROM Products
5 ORDER BY ProductCount * Price

```

100 %

Results Messages

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

## Вилучення діапазону рядків

### Оператор TOP

Оператор TOP дозволяє вибрати певну кількість рядків із таблиці:

```

1 SELECT TOP 4 ProductName
2 FROM Products

```

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT TOP 4 ProductName
4 FROM Products

```

100 %

Results Messages

	ProductName
1	iPhone 6
2	iPhone 6S
3	iPhone 7
4	Galaxy S8

Додатковий оператор PERCENT дозволяє вибрати відсоткову кількість рядків із таблиці. Наприклад, виберемо 75% рядків:

```

1 SELECT TOP 75 PERCENT ProductName
2 FROM Products

```

## OFFSET та FETCH

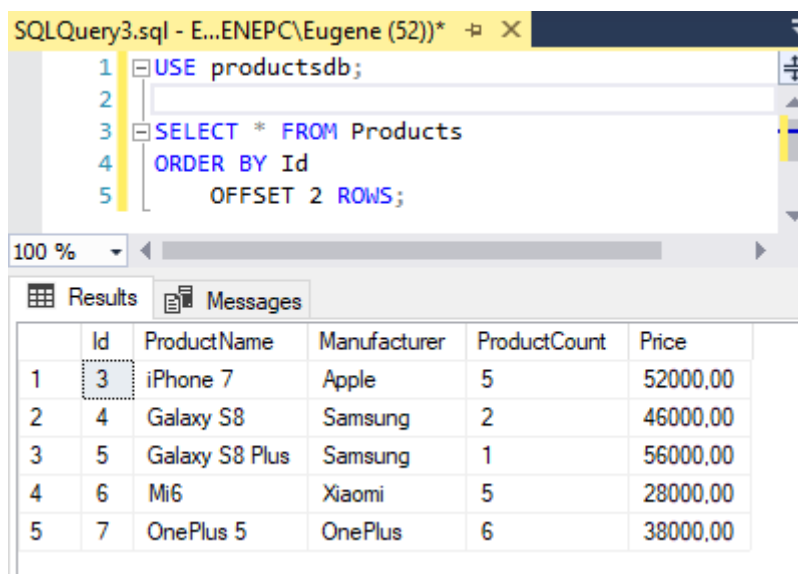
Оператор TOP дозволяє витягти певну кількість рядків, починаючи з початку таблиці. Для вилучення набору рядків з будь-якого місця застосовуються оператори OFFSET і FETCH. Важливо, що ці оператори застосовуються лише у відсортованому наборі даних після виразу ORDER BY.

```
1 ORDER BY statement
2 OFFSET shift {ROW|ROWS}
3 [FETCH {FIRST|NEXT} numner_of_records {ROW|ROWS} ONLY]
```

Наприклад, виберемо всі рядки, починаючи з третього:

```
1 SELECT * FROM Products
2 ORDER BY Id
3 OFFSET 2 ROWS;
```

Число після ключового слова OFFSET вказує, скільки рядків потрібно пропустити.



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

Тепер виберемо лише три рядки, починаючи з третього:

```
1 SELECT * FROM Products
2 ORDER BY Id
3 OFFSET 2 ROWS
4 FETCH NEXT 3 ROWS ONLY;
```

Після оператора FETCH вказується ключове слово FIRST або NEXT (яке саме в даному випадку не має значення) і потім вказується кількість рядків, які треба отримати.

SQLQuery3.sql - E...\ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT * FROM Products
4 ORDER BY Id
5     OFFSET 2 ROWS
6     FETCH NEXT 3 ROWS ONLY;

```

100 %

Results Messages

	Id	ProductName	Manufacturer	ProductCount	Price
1	3	iPhone 7	Apple	5	52000,00
2	4	Galaxy S8	Samsung	2	46000,00
3	5	Galaxy S8 Plus	Samsung	1	56000,00

Ця комбінація операторів зазвичай використовується для посторінкової навігації, коли необхідно отримати певну сторінку з даними.

## Фільтрування. WHERE

Для фільтрації у команді SELECT застосовується оператор WHERE. Після цього оператора ставиться умова, якій має відповідати рядок:

```
1 WHERE condition
```

Якщо умова є істинною, то рядок потрапляє в результуючу вибірку. Як можна використовувати операції порівняння. Ці операції порівнюють два вирази. У T-SQL можна застосовувати такі операції порівняння:

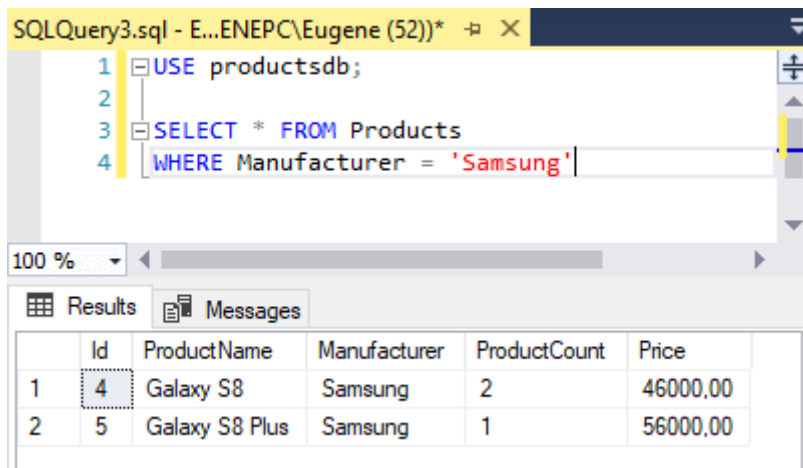
- =: порівняння на рівність (на відміну від сі-подібних мов у T-SQL для порівняння на рівність використовується один знак одно)
- <>: порівняння на нерівність
- <: менше ніж
- >: більше ніж
- !<: не менше як
- !>: не більш ніж
- <=: менше ніж або одно
- >=: більше ніж або одно

Наприклад, знайдемо всі товари, виробником яких є компанія Samsung:

```

1 SELECT * FROM Products
2 WHERE Manufacturer = 'Samsung'

```



Варто зазначити, що в даному випадку регістр не має значення, і ми могли б використовувати для пошуку рядок "Samsung", і "SAMSUNG", і "samsung". Усі ці варіанти давали б еквівалентний результат вибірки.

Інший приклад - знайдемо всі товари, у яких ціна більша за 45000:

```

1 SELECT * FROM Products
2 WHERE Price > 45000

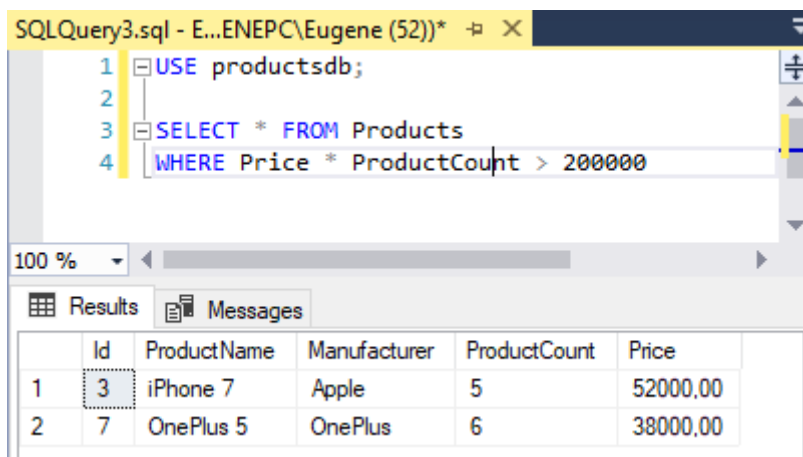
```

Як умови можуть використовуватися і складніші висловлювання. Наприклад, знайдемо всі товари, у яких сукупна вартість більша за 200 000:

```

1 SELECT * FROM Products
2 WHERE Price * ProductCount > 200000

```



## Логічні оператори

Для поєднання кількох умов в одне можуть використовуватися логічні оператори. У T-SQL є такі логічні оператори:

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

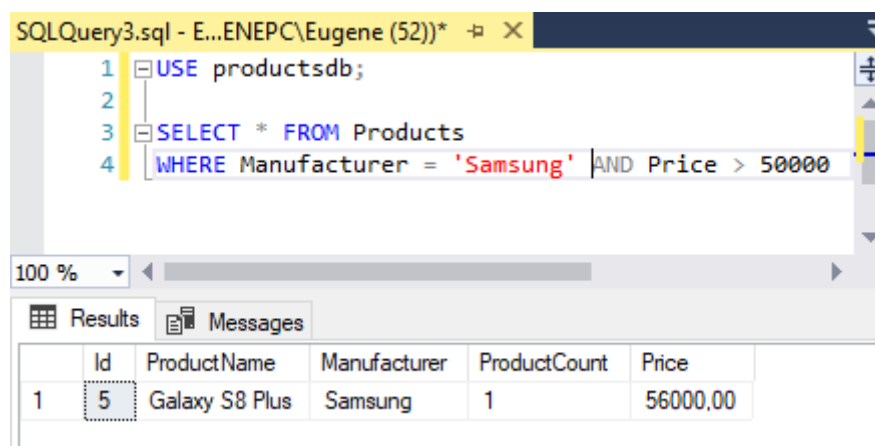
```
1 вираз1 AND вираз2
```

- Тільки якщо обидва ці висловлювання одночасно є істинними, то й загальна умова оператора AND також буде істинною. Тобто, якщо і перша умова істинна, і друга.
- **OR**: операція логічного АБО. Вона також поєднує два вирази:  
1    вираз1 OR вираз2
- Якщо хоча б один із цих виразів істинний, то загальна умова оператора OR також буде істинною. Тобто якщо або перша умова істинна, або друга.
- **NOT**: операція логічного заперечення. Якщо вираз у цій операції помилковий, то загальна умова є істинною.  
1    NOT вираз

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

Наприклад, виберемо всі товари, у яких виробник Samsung і водночас ціна більша за 50000:

```
1  SELECT * FROM Products
2  WHERE Manufacturer = 'Samsung' AND Price > 50000
```



Тепер змінимо оператор на OR. Тобто виберемо всі товари, у яких або виробник Samsung, або ціна більша за 50000:

```
1  SELECT * FROM Products
2  WHERE Manufacturer = 'Samsung' OR Price > 50000
```



SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT * FROM Products
4 WHERE Manufacturer = 'Samsung' OR Price > 50000

```

100 %

Results Messages

	Id	ProductName	Manufacturer	ProductCount	Price
1	3	iPhone 7	Apple	5	52000,00
2	4	Galaxy S8	Samsung	2	46000,00
3	5	Galaxy S8 Plus	Samsung	1	56000,00

Застосування оператора NOT – виберемо всі товари, у яких виробник не Samsung:

```

1 SELECT * FROM Products
2 WHERE NOT Manufacturer = 'Samsung'

```

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

1 USE productsdb;
2
3 SELECT * FROM Products
4 WHERE NOT Manufacturer = 'Samsung'

```

100 %

Results Messages

	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	6	Mi6	Xiaomi	5	28000,00
5	7	OnePlus 5	OnePlus	6	38000,00

Але здебільшого цілком можна обійтися без оператора NOT. Так, у попередній приклад ми можемо переписати так:

```

1 SELECT * FROM Products
2 WHERE Manufacturer <> 'Samsung'

```

Також в одній команді SELECT можна використовувати відразу кілька операторів:

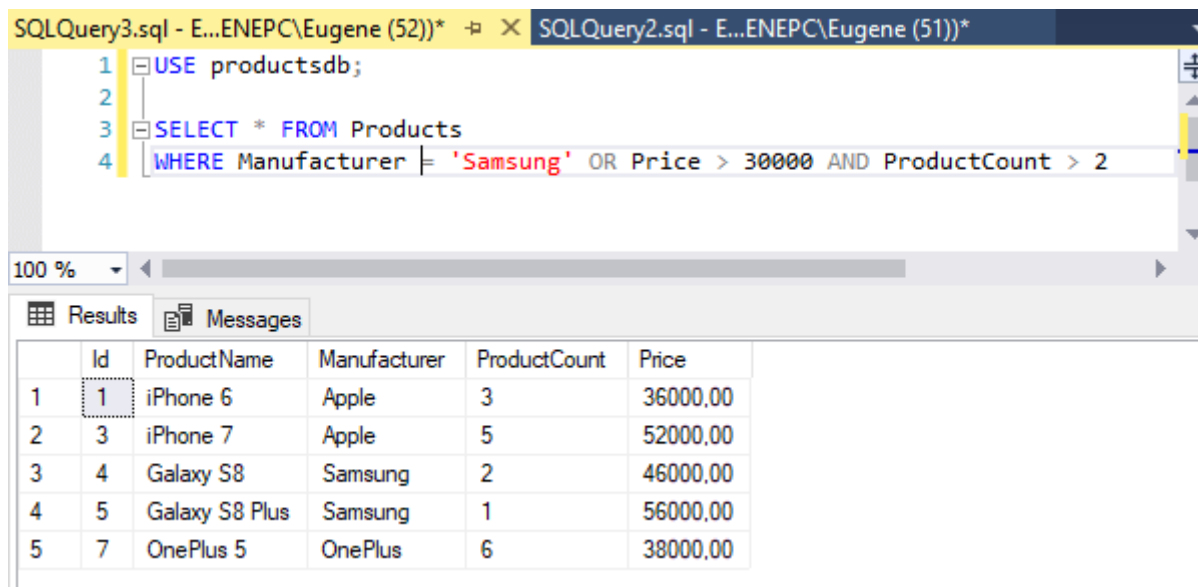
```

1 SELECT * FROM Products
2 WHERE Manufacturer = 'Samsung' OR Price > 30000 AND ProductCount > 2

```

Так як оператор AND має більш високий пріоритет, то спочатку буде виконуватися вираз Price > 30000 AND ProductCount > 2, і тільки потім

оператор OR. Тобто тут вибираються товари, яких на складі більше 2 і у яких одночасно ціна більша за 30000, або ті товари, виробником яких є Samsung.



The screenshot shows a SQL query window with the following text:

```
1 USE productsdb;  
2  
3 SELECT * FROM Products  
4 WHERE Manufacturer = 'Samsung' OR Price > 30000 AND ProductCount > 2
```

Below the query window, the 'Results' tab is active, displaying a table with 5 rows and 6 columns:

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

За допомогою дужок ми також можемо перевизначити порядок операцій:

```
1 SELECT * FROM Products  
2 WHERE (Manufacturer = 'Samsung' OR Price > 30000) AND ProductCount > 2
```

## IS NULL

Ряд стовпців може допускати значення NULL. Це значення не еквівалентно порожньому рядку ". NULL представляє повну відсутність будь-якого значення. І для перевірки на наявність такого значення застосовується оператор IS NULL.

Наприклад, виберемо всі товари, у яких не встановлено поле ProductCount:

```
1 SELECT * FROM Products  
2 WHERE ProductCount IS NULL
```

Якщо, навпаки, необхідно отримати рядки, у яких поле ProductCount не рівне NULL, то можна використовувати оператор NOT:

```
1 SELECT * FROM Products  
2 WHERE ProductCount IS NOT NULL
```

## Оператори фільтрації

### Оператор IN

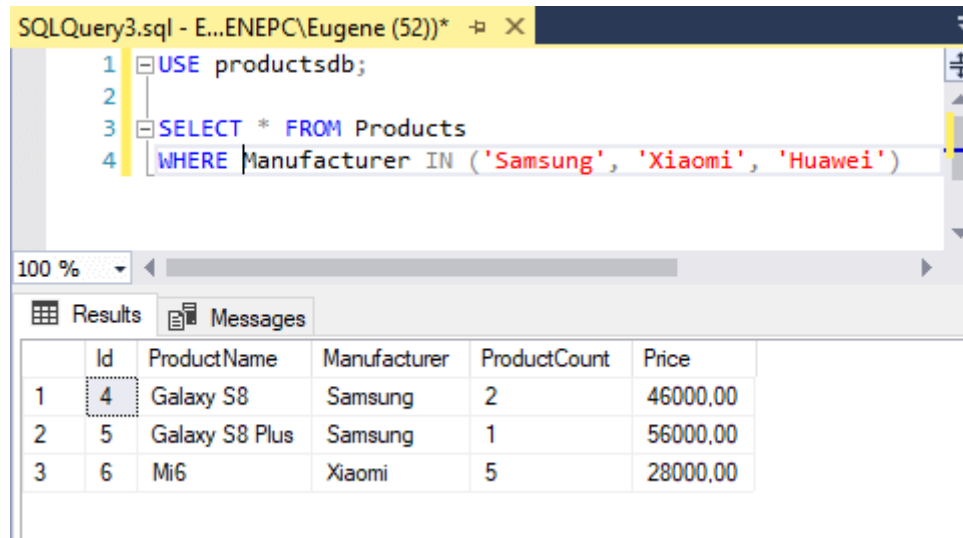
Оператор IN дозволяє визначити набір значень, які мають стовпці:

```
1 WHERE expression [NOT] IN (expression)
```

Вираз у дужках після IN визначає набір значень. Цей набір може обчислюватися динамічно на підставі, наприклад, ще одного запиту або це можуть бути константні значення.

Наприклад, виберемо товари, у яких виробник або Samsung, або Xiaomi, або Huawei:

```
1 SELECT * FROM Products
2 WHERE Manufacturer IN ('Samsung', 'Xiaomi', 'Huawei')
```



The screenshot shows a SQL query window titled 'SQLQuery3.sql - E...ENEPC\Eugene (52)\*'. The query is as follows:

```
1 USE productsdb;
2
3 SELECT * FROM Products
4 WHERE Manufacturer IN ('Samsung', 'Xiaomi', 'Huawei')
```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

	Id	ProductName	Manufacturer	ProductCount	Price
1	4	Galaxy S8	Samsung	2	46000,00
2	5	Galaxy S8 Plus	Samsung	1	56000,00
3	6	Mi6	Xiaomi	5	28000,00

Ми могли б всі ці значення перевірити через оператор OR:

```
1 SELECT * FROM Products
2 WHERE Manufacturer = 'Samsung' OR Manufacturer = 'Xiaomi' OR Manufacturer
   = 'Huawei'
```

Але використання оператора IN набагато зручніше, особливо якщо подібних значень дуже багато.

За допомогою оператора NOT можна знайти всі рядки, які, навпаки, не відповідають набору значень:

```
1 SELECT * FROM Products
2 WHERE Manufacturer NOT IN ('Samsung', 'Xiaomi', 'Huawei')
```

## Оператор BETWEEN

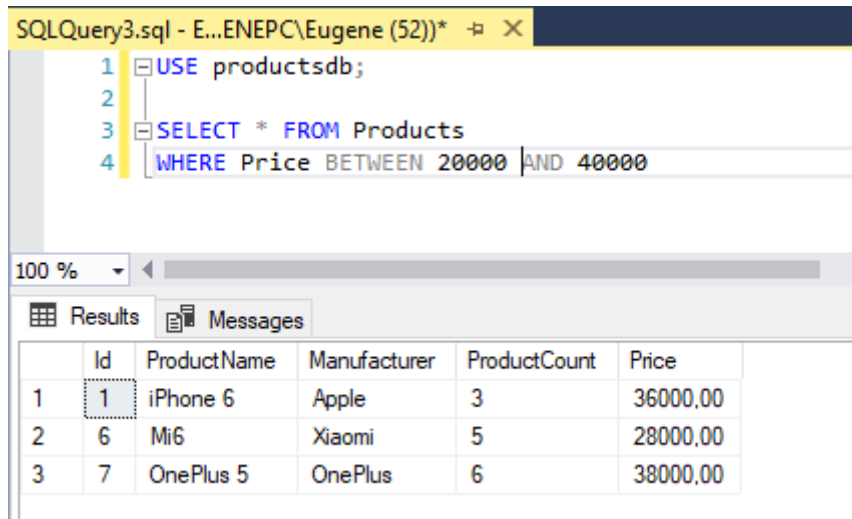
Оператор BETWEEN визначає діапазон значень за допомогою початкового та кінцевого значення, якому має відповідати вираз:

```
1 WHERE statement [NOT] BETWEEN start_value AND end_value
```

Наприклад, отримаємо всі товари, у яких ціна від 20 000 до 40 000 (початкове та кінцеве значення також включаються до діапазону):

```
1 SELECT * FROM Products
```

```
2 WHERE Price BETWEEN 20000 AND 40000
```



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

```
1 USE productsdb;  
2  
3 SELECT * FROM Products  
4 WHERE Price BETWEEN 20000 AND 40000
```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	6	Mi6	Xiaomi	5	28000,00
3	7	OnePlus 5	OnePlus	6	38000,00

Якщо треба, навпаки, вибрати ті рядки, які не потрапляють у цей діапазон, то застосовується оператор NOT:

```
1 SELECT * FROM Products  
2 WHERE Price NOT BETWEEN 20000 AND 40000
```

Також можна використовувати складніші висловлювання. Наприклад, отримаємо товари, запаси яких на певну суму (ціна \* кількість):

```
1 SELECT * FROM Products  
2 WHERE Price * ProductCount BETWEEN 100000 AND 200000
```

## Оператор LIKE

Оператор LIKE приймає шаблон рядка, якому має відповідати вираз.

```
1 WHERE вираз [NOT] LIKE шаблон_рядки
```

Для визначення шаблону може застосовуватися ряд спеціальних символів підстановки:

- %: відповідає будь-якому підрядку, який може мати будь-яку кількість символів, при цьому підрядок може і не містити жодного символу
- \_: відповідає будь-якому одиночному символу
- []: відповідає одному символу, який вказаний у квадратних дужках.
- [ - ]: відповідає одному символу з певного діапазону
- [^]: відповідає одному символу, який не вказано після ^

Деякі приклади використання підстановок:

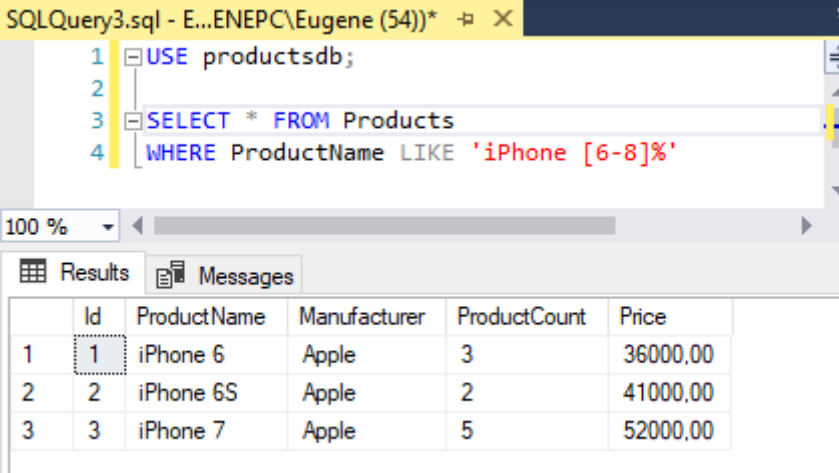
- WHERE ProductName LIKE 'Galaxy%'

Відповідає таким значенням як "Galaxy Ace 2" або "Galaxy S7"

- WHERE ProductName LIKE 'Galaxy S\_'  
Відповідає таким значенням як "Galaxy S7" або "Galaxy S8"
- WHERE ProductName LIKE 'iPhone [78]'  
Відповідає таким значенням як iPhone 7 або iPhone8
- WHERE ProductName LIKE 'iPhone [6-8]'  
Відповідає таким значенням як iPhone 6, iPhone 7 або iPhone8
- WHERE ProductName LIKE 'iPhone [^7]%'  
Відповідає таким значенням як iPhone 6, iPhone 6S або iPhone8. Але не відповідає значенням "iPhone 7" та "iPhone 7S"
- WHERE ProductName LIKE 'iPhone [^1-6]%'  
Відповідає таким значенням як iPhone 7, iPhone 7S і iPhone 8. Але не відповідає значенням "iPhone 5", "iPhone 6" та "iPhone 6S"

Застосуємо оператор LIKE:

```
1 SELECT * FROM Products
2 WHERE ProductName LIKE 'iPhone [6-8]%'
```



SQLQuery3.sql - E...ENEPC\Eugene (54)\*

```

1 USE productsdb;
2
3 SELECT * FROM Products
4 WHERE ProductName LIKE 'iPhone [6-8]%'

```

100 %

	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

## Оновлення даних. Команда UPDATE

Для зміни рядків у таблиці застосовується команда UPDATE. Вона має такий формальний синтаксис:

```

1 UPDATE table_name
2 SET column1 = value1, column2 = value2, ... columnN = valueN
3 [FROM result_set AS alias]
4 [WHERE update_condition]

```

Наприклад, збільшимо у всіх товарів ціну на 5000:

```

1  UPDATE Products
2  SET Price = Price + 5000

```

SQLQuery3.sql - E...ENEPC\Eugene (52))\*

```

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

```

100 %

Results Messages

	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

Використовуємо критерій і змінимо назву виробника з "Samsung" на "Samsung Inc.":

```

1  UPDATE Products
2  SET Manufacturer = 'Samsung Inc.'
3  WHERE Manufacturer = 'Samsung'

```

Більш складний запит - замінімо у поля Manufacturer значення Apple на Apple Inc. у перших 2 рядках:

```

1  UPDATE Products
2  SET Manufacturer = 'Apple Inc.'
3  FROM
4  (SELECT TOP 2 FROM Products WHERE Manufacturer='Apple') AS Selected
5  WHERE Products.Id = Selected.Id

```

За допомогою підзапиту після ключового слова FROM проводиться вибірка перших двох рядків, у яких Manufacturer = Apple. Для цієї вибірки буде визначено псевдонім Selected. Псевдонім зазначається після оператора AS.

Далі йде умова оновлення Products.Id=Selected.Id. Тобто фактично ми маємо справу з двома таблицями – Products та Selected (яка є похідною від Products). У Selected знаходиться два перших рядки, у яких Manufacturer = 'Apple'. У Products взагалі всі рядки. І оновлення проводиться тільки для рядків, які є у вибірці Selected. Тобто якщо в таблиці Products десятки товарів з виробником Apple, то оновлення торкнеться лише двох перших.

## **Видалення даних. Команда DELETE**

Для видалення застосовується команда DELETE:

```
1 DELETE [FROM] table_name
2 WHERE delete_condition
```

Наприклад, видалимо рядки, у яких id дорівнює 9:

```
1 DELETE Products
2 WHERE Id=9
```

Або видалимо всі товари, виробником яких є Xiaomi і які мають ціну меншу за 15000:

```
1 DELETE Products
2 WHERE Manufacturer='Xiaomi' AND Price < 15000
```

Більш складний приклад - видалімо перші два товари, у яких виробник - Apple:

```
1 DELETE Products FROM
2 (SELECT TOP 2 * FROM Products
3 WHERE Manufacturer='Apple') AS Selected
4 WHERE Products.Id = Selected.Id
```

Після першого оператора FROM відбувається вибірка двох рядків з таблиці Products. Цій вибірці призначається псевдонім Selected за допомогою оператора AS. Далі встановлюємо умову, що й Id у таблиці Products має те значення, як і Id у вибірці Selected, то рядок видаляється.

SQLQuery1.sql - E...ENEPC\Eugene (53))\* X

```

1 USE productsdb;
2
3 SELECT * FROM Products
4
5 DELETE Products FROM
6 (SELECT TOP 2 * FROM Products
7  WHERE Manufacturer='Apple') AS Selected
8  WHERE Products.Id = Selected.Id
9
10 SELECT * FROM Products
11

```

100 %

Results Messages

	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	3	iPhone 7	Apple	5	52000,00
2	4	Galaxy S8	Samsung	2	46000,00
3	5	Galaxy S8 Plus	Samsung	1	56000,00
4	6	Mi6	Xiaomi	5	28000,00
5	7	OnePlus 5	OnePlus	6	38000,00

Якщо необхідно видалити всі рядки незалежно від умови, то умову можна не вказувати:

```
1 DELETE Products
```