**Лабораторна робота №10. Користувацькі функції**

Користувацькі функції дуже схожі з збереженими процедурами. Однак, на відміну від збережених процедур, вони можуть використовуватися тільки в запитах так само, як і системні функції. Як і процедури, вони можуть мати один або кілька параметрів, але повертають скалярні типи даних, наприклад int, decimal, varchar, або тип даних table.

Основні обмеження:

* + Визначені користувачем функції не можуть виконувати дії, що змінюють стан бази даних.
  + Визначені користувачем функції не можуть містити пропозицію OUTPUT INTO, метою якої є таблиця.
  + Визначені користувачем функції не можуть повертати кілька результуючих наборів. Використовуйте збережену процедуру, якщо потрібно повертати кілька результуючих наборів.
  + Обробка помилок у функціях, які визначаються користувачем, обмежена. UDF не підтримує тип TRY ... CATCH, @ERROR і RAISERROR.
  + Визначені користувачем функції не можуть викликати збережену процедуру, але можуть викликати розширену збережену процедуру.
  + Визначені користувачем функції не можуть використовувати динамічний SQL і тимчасові таблиці. Табличні змінні дозволені до використання.
  + Пуста пропозицію FOR XML заборонено.
  + Визначені користувачем функції можуть бути вкладеними, тобто з однієї функції може бути викликана інша.

Як правило, продуктивність скалярних призначених для користувача функцій виявляється невисокою за вказаними нижче причин:

– **ітеративні виклики**. Користувацькі функції викликаються ітеративно - одноразово у відповідному кортежі. Постійні перемикання контексту при виклику функцій вимагають додаткових ресурсів;

– **відсутність оцінки витрат**. Під час оптимізації оцінюються тільки реляційні оператори, але не скалярні. До появи скалярних функцій користувача скалярні оператори, як правило, були невибагливі до ресурсів і не потребували оцінки. Досить було враховувати невелике збільшення завантаження ЦП;

– **виконання, орієнтоване на інтерпретацію**. Користувацькі функції оцінюються як пакети інструкцій, але інструкції виконуються по черзі. Кожна інструкція компілюється окремо, а потім скомпільований план кешується. Хоча така стратегія кешування дозволяє уникнути повторної компіляції і трохи заощадити час, кожна інструкція виконується ізольовано. Перехресна оптимізація інструкцій не проводиться.

– **послідовне виконання**. У SQL Server не допускається паралелізм всередині запитів, що викликають призначені для користувача функції.

**Проста скалярна функція**

Визначення скалярної функції наступне:

CREAТЕ ЕіЕСТ10Е [імя\_схеми.] Імя\_функциі

[{@ Ім’я\_параметра тип\_параметра}] [, ... n]

RETURNS

[WITH]

AS

BEGIN

<Оператори Transact-SQL, що становлять тіло процедури>

RETURN <скалярне значення>

END

*Приклад:*

*CREATE FUNCTION testF(@n1 int, @n2 as int)*

*RETURNS int*

*AS*

*BEGIN*

*Return (@n1 \* @n2)*

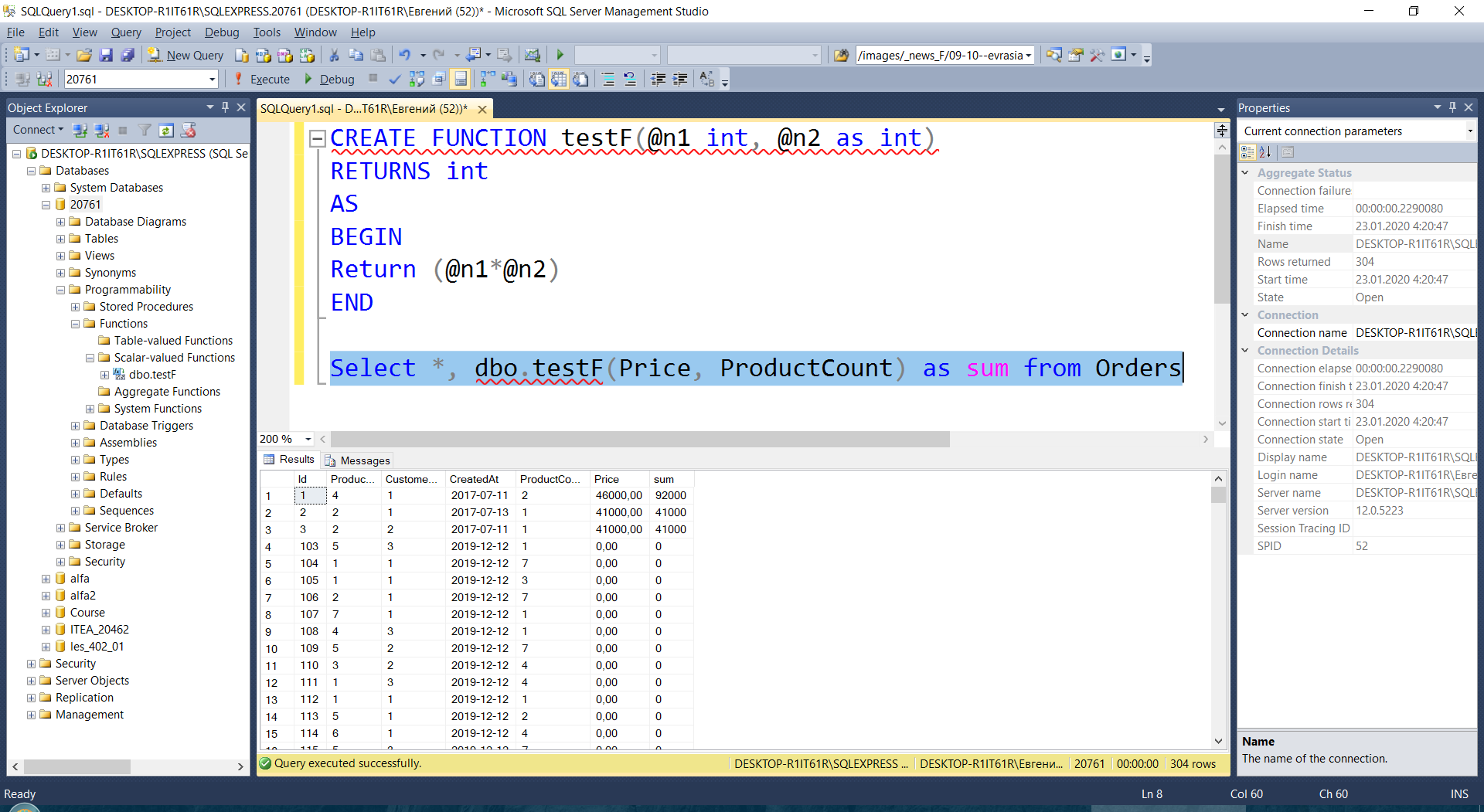
*END*

викликати функцію на виконання можна, можливо так:

*Select dbo.testF(5, 8)*

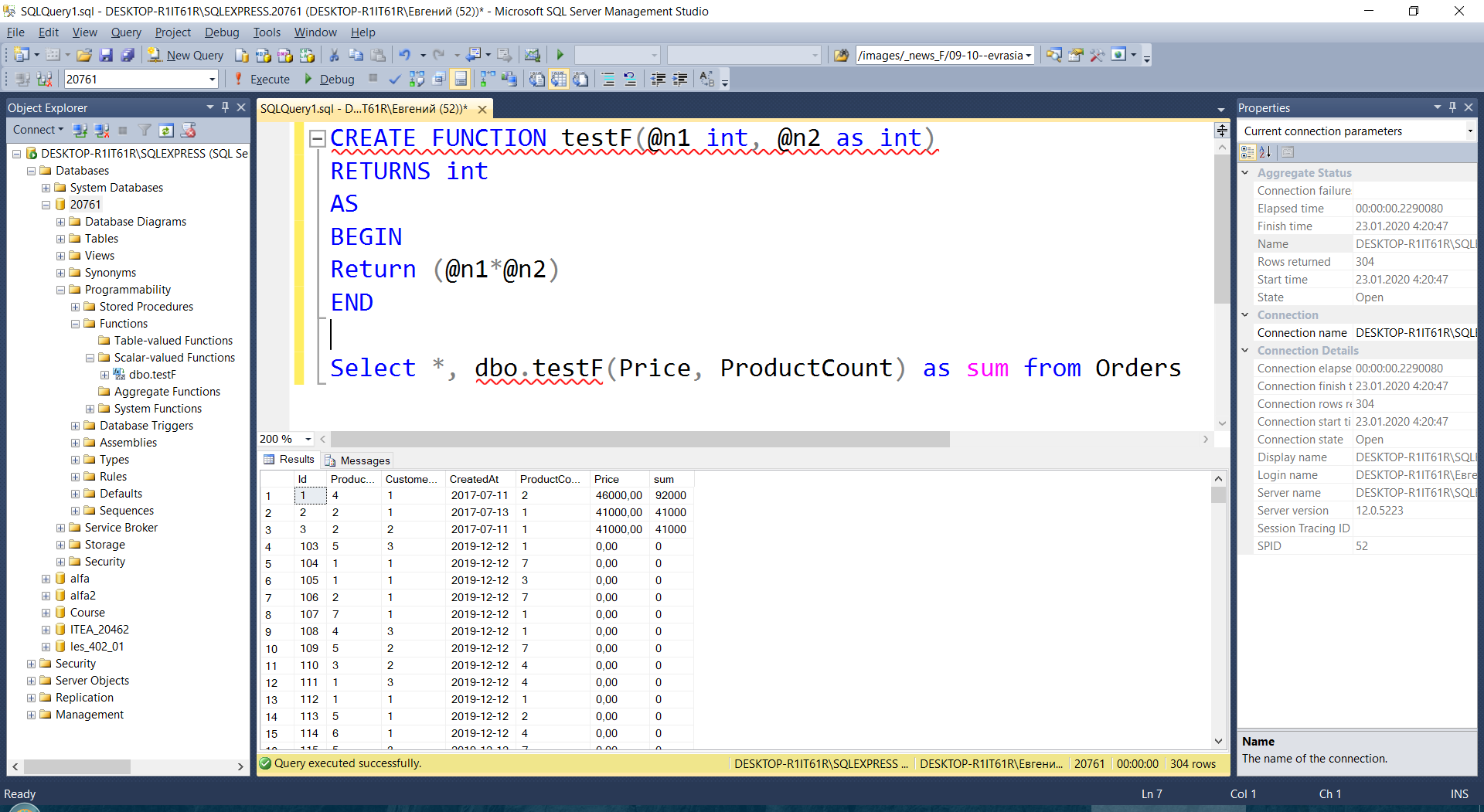


Місцезнаходження функції:



*Використання в запитах:*

*Select \*, dbo.testF(Price, ProductCount) As sum from Orders*



Функції з запитами:

CREATE FUNCTION dbo.ufnGetCountProduct(@Product int)

RETURNS int

AS

BEGIN

DECLARE @ret int

SELECT @ret = SUM (orders.ProductCount) FROM orders

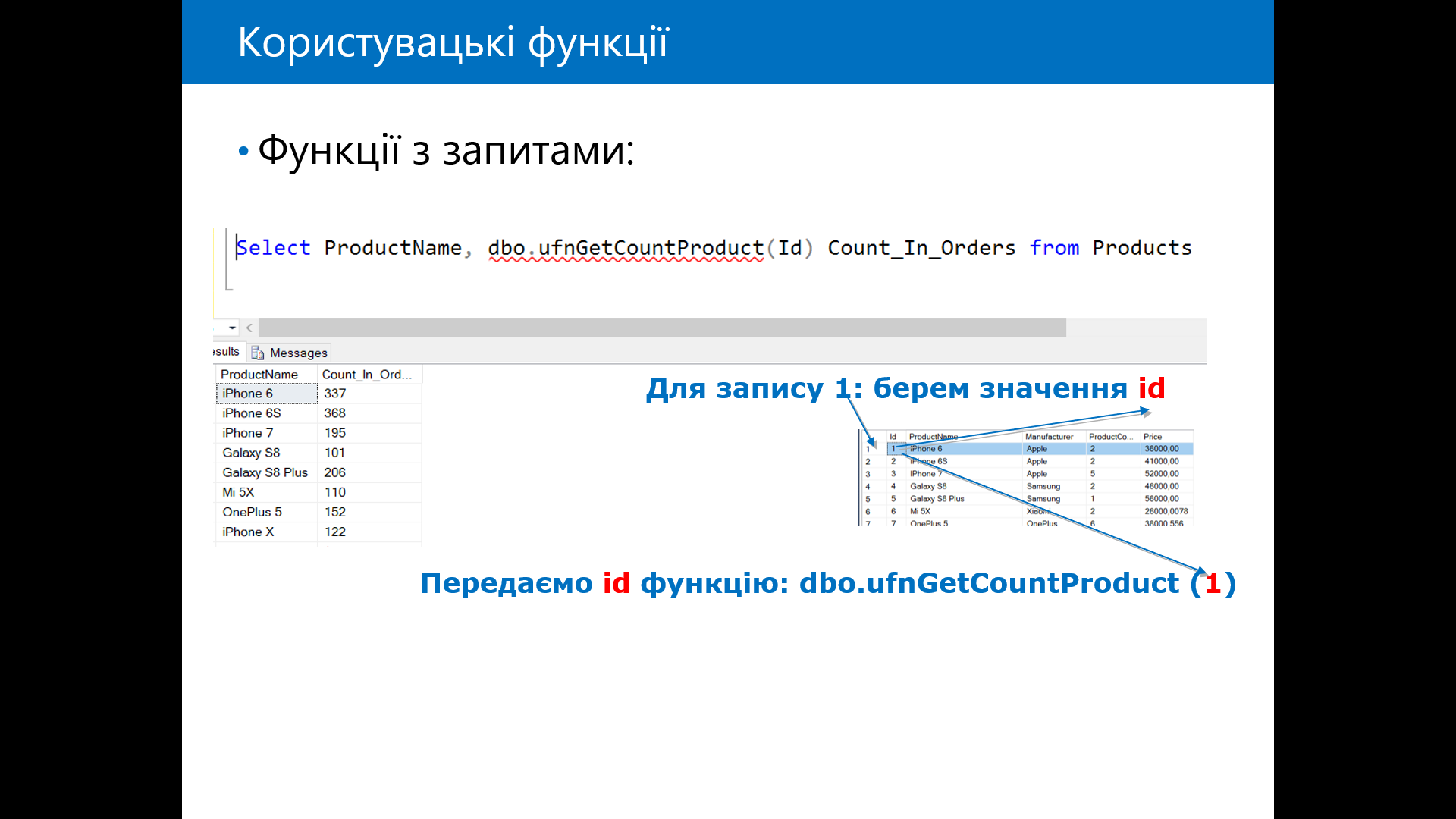
WHERE ProductID = @Product

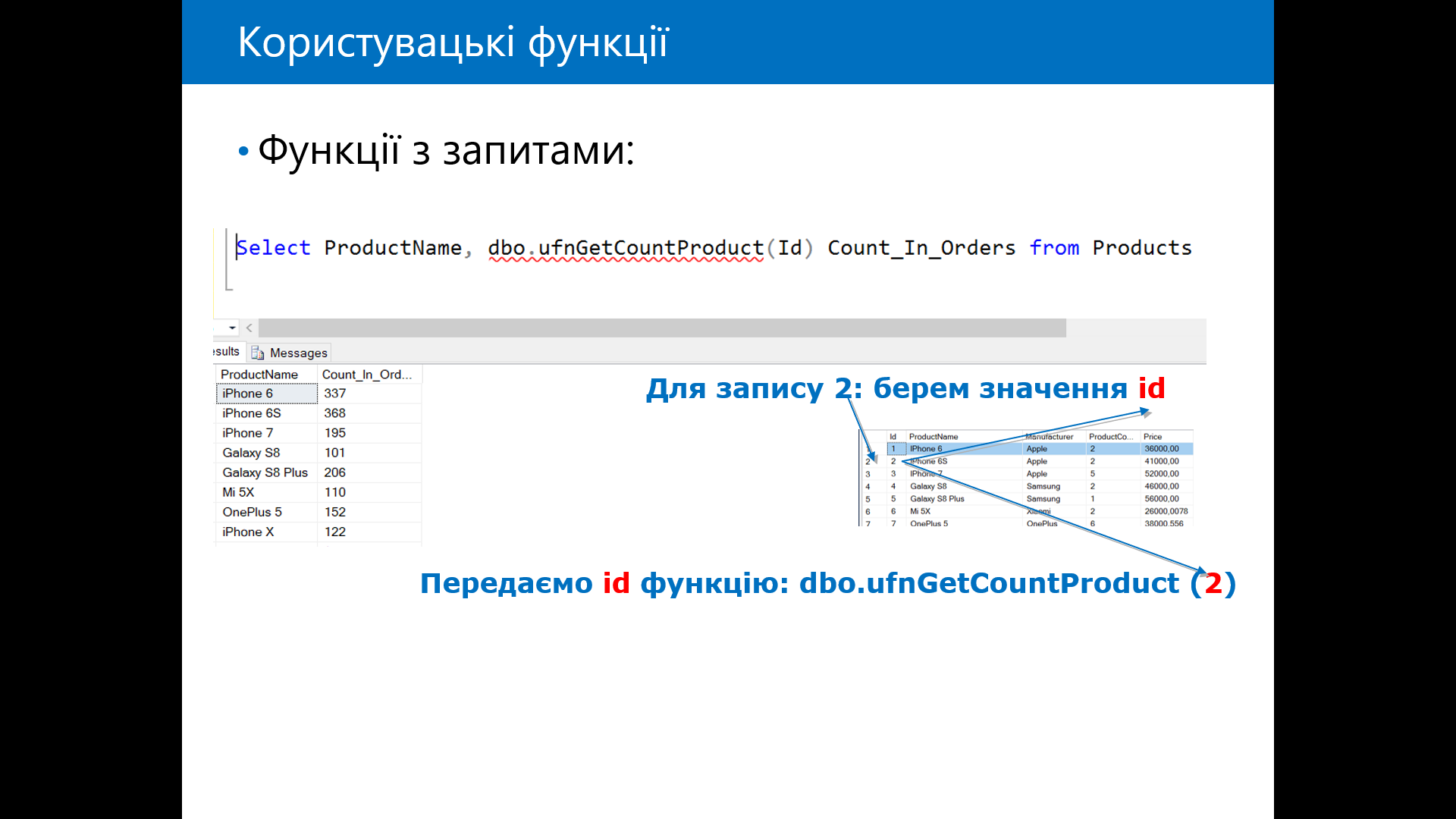
IF (@ret IS NULL) SET @ret = 0;

RETURN @ret;

END;

Select ProductName, dbo.ufnGetCountProduct(Id) Count\_In\_Orders from Products





Принципи створення функцій

* 1. Коли треба робити?
  + Тоді, коли постійно робимо одну і ту саме дію на сталій структурі таблиць
  + Коли базою користуються багато користувачів, які не вміють будувати підзапити, але можуть оперувати вбудованими функціями
* 2. З чого починати написання функції?
  + Проаналізувати задачу та розробити алгоритм
  + Оптимізувати код за швидкодією та кількістю звертань до БД
* 3. Як організувати отримання, передачу та обробку всередині функції значень з БД?
  + Використовувати змінні для обміну даними та розрахунків

**Параметр за замовчунням**

Особливості використання параметрів за замовчунням полягає в тому, що необхідно їх обов’язково відзначати параметром default, якщо вони не передбачені для визначення.

Особливість виклику:

CREATE FUNCTION SelectSumOrdersByProdIdDateN (@Product int, @NDays int = 10) RETURNS int

AS

BEGIN

DECLARE @ret int

SELECT @ret = SUM (o.ProductCount\*o.price) FROM orders AS o

WHERE ProductID = @Product

and o.CreatedAt BETWEEN CAST(GETDATE()-@NDays AS DATE) AND CAST(GETDATE() AS DATE)

IF (@ret IS NULL) SET @ret = 0;

RETURN @ret;

END;

---

select o.\*, dbo.SelectSumOrdersByProdIdDateN(o.ProductId, 1)

from orders as o

select o.\*, dbo.SelectSumOrdersByProdIdDateN(o.ProductId, default)

from orders as o

Завдання

1. Проробити всі приклади з роботи
2. Створити функцію для отримання суми закупок за кодом замовника.
3. Створити функцію для отримання імені замовника за його id. Використати дану функцію для таблиці Orders.
4. Створити функцію для отримання максимальної вартості замовлення за id замовника (CustomerId). Використати дану функцію для таблиці Customers.
5. \*Створити користувацькі функції для отримання з повного імені окремо значення Імені та окремо По батькові (dbo.want\_name2 для імені, dbo.want\_name3 – для По батькові).
6. \*Модифікувати таблицю Customers і додати поля Name2, Name3 (відповідно для Імені і для По батькові), куди через UPDATE з використанням нових функцій занести данні.
7. \*Створити наступні функції та перевірити їх роботу
   1. підрахунок сум продажів по id продукту
   2. підрахунок числа замовлень по id продукту
   3. підрахунок сум продажів по id продукту за останніх 10 днів
   4. підрахунок числа замовлень по id продукту за останніх 10 днів
   5. підрахунок сум продажів по id продукту за останніх N днів
   6. підрахунок числа замовлень по id продукту за останніх N днів
8. \*Об'єднати всі 6 функцій, вибираючи виконання по переданому параметру