

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

**Бази даних та інформаційні системи**

**ЛАБОРАТОРНА РОБОТА №9**

**Тема: «Збережені процедури СКБД PostgreSQL»**

Виконав:

*Ст. Лук'янчук Денис*

Група ПМІ-33

2025

**Тема:** «Збережені процедури СКБД PostgreSQL».

**Мета роботи:** Вивчення поняття збережених процедур СКБД PostgreSQL v.11 .

**Завдання лабораторної роботи:**

1. Опрацювати теоретичний матеріал.
2. Розробити дві або три збережені процедури на процедурній мові PL/pgSQL відповідно до власних потреб роботи зі створюваною базою даних.
3. У збережених процедурах необхідно використати:
  - оголошення вхідних та вихідних параметрів функції, псевдоніми, змінні та константи;
  - присвоювання змінній типу RECORD та команди PERFORM, STRICT або EXECUTE на вибір;
  - керуючі структури (IF/CASE/FOR ... на вибір);
  - обов'язково команди обробки помилок EXCEPTION та виведення повідомлень RAISE;
  - обов'язково використати курсор.

## **Теоретичний матеріал**

### **1. Збережені процедури**

- **Процедура** – іменований блок PL/pgSQL, що зберігається на сервері, виконує набір SQL-команд та процедурної логіки.
- **Відмінності від функції:**
  - не повертає значення напряму (результат можна повернути через OUT/INOUT);
  - викликається оператором CALL;
  - дозволяє керувати транзакціями (BEGIN, COMMIT, ROLLBACK).

### **2. Переваги використання PL/pgSQL**

- Можливість об'єднувати складну логіку на сервері, економлячи ресурси клієнта.
- Виконання циклів, умов, курсорів без необхідності надсилати багато запитів із клієнта.
- Динамічне формування SQL-команд через EXECUTE.
- Виведення повідомлень через RAISE та перевірки через ASSERT.

### **3. Основні конструкції у процедурах**

- **PARAMETERS (IN, OUT, INOUT)** – визначають дані, які подаються на вхід, повертаються на вихід або змінюються всередині процедури.
- **ALIAS FOR** – створює псевдонім для параметра, зручний при великій кількості аргументів.
- **CONSTANT** – незмінні величини у процедурі (наприклад, мінімальна ціна).

- **RECORD** – універсальна змінна для збереження одного рядка будь-якої структури таблиці.
- **STRICT** – примусове повернення одного рядка, якщо даних немає – викликає помилку.
- **PERFORM** – виконує SQL без збереження результату, наприклад для виклику функції або обчислення.
- **EXECUTE** – динамічне виконання SQL, зручно при циклічному оновленні.
- **CURSOR** – для ітерації по множині рядків SELECT.
- **IF / CASE / LOOP / FOR** – керують логікою виконання.
- **EXCEPTION** – обробка помилок із можливістю RAISE повідомлень.
- **RAISE / ASSERT** – вивід повідомлень та перевірка умов.

## Хід роботи

### Процедура 1: add\_new\_part

Додає нову автозапчастину до бази даних з перевіркою унікальності серійного номера та валідності виробника.

#### Використовує:

- Вхідні параметри (p\_serial\_number, p\_name, p\_manufacturer\_id, p\_price)
- INOUT параметр (p\_status)
- RECORD (part\_record)
- Псевдонім (alias\_serial)
- EXCEPTION + RAISE для обробки помилок
- Керуючу структуру IF + FOUND

#### Код процедури:

```

1 CREATE OR REPLACE PROCEDURE add_new_part(
2     IN p_serial_number serial_number,
3     IN p_name VARCHAR(200),
4     IN p_manufacturer_id INTEGER,
5     IN p_price positive_decimal,
6     INOUT p_status VARCHAR(100)
7 )
8 LANGUAGE plpgsql
9 AS $$
10 DECLARE
11     part_record RECORD;
12     alias_serial ALIAS FOR p_serial_number;
13 BEGIN
14     -- Check if serial number already exists
15     SELECT * INTO part_record
16     FROM Part
17     WHERE Serial_Number = alias_serial;
18
19     IF FOUND THEN
20         p_status := 'Error: Serial number ' || alias_serial || ' already exists';
21         RAISE NOTICE 'Serial number % already exists', alias_serial;
22         RETURN;
23     END IF;
24
25     -- Insert new part
26     BEGIN
27         INSERT INTO Part (Serial_Number, Name, Manufacturer_ID, Price)
28         VALUES (p_serial_number, p_name, p_manufacturer_id, p_price);
29
30         p_status := 'Success: Part ' || p_name || ' added';
31         RAISE NOTICE 'Part % added successfully', p_name;
32     END;
33
34     EXCEPTION
35     WHEN foreign_key_violation THEN
36         p_status := 'Error: Invalid Manufacturer_ID ' || p_manufacturer_id;

```

```

36         RAISE WARNING 'Foreign key violation: Manufacturer_ID % does not exist', p_manufacturer_id;
37     WHEN others THEN
38         p_status := 'Error: ' || SQLERRM;
39         RAISE WARNING 'Unexpected error: %', SQLERRM;
40     END;
41 END;
42 $$;

```

## Параметри, що були використані для створення процедури включають:

- Вхідні параметри (p\_serial\_number, p\_name, p\_manufacturer\_id, p\_price)
- INOUT параметр (p\_status для повернення статусу операції)
- Псевдонім (alias\_serial для p\_serial\_number)
- RECORD (part\_record для збереження результату SELECT)
- Керуюча структура IF для перевірки існування запису
- EXCEPTION для обробки помилок foreign\_key\_violation та інших
- RAISE для виведення повідомлень різних рівнів

## Перевірка виконання процедури:

Спершу виконав запит, щоб подивитися поточний стан таблиці Part:

Query Query History

1

SELECT \* FROM public.part

2

ORDER BY part\_id ASC

Data Output Messages Notifications

Showing rows: 1 to 10

	part_id [PK] integer	serial_number character varying (50)	name character varying (200)	manufacturer_id integer	price numeric (10,2)
81	81	SN000081	Part_81	65	225.82
82	82	SN000082	Part_82	24	277.19
83	83	SN000083	Part_83	59	453.44
84	84	SN000084	Part_84	76	14.64
85	85	SN000085	Part_85	55	394.11
86	86	SN000086	Part_86	83	104.93
87	87	SN000087	Part_87	64	710.65
88	88	SN000088	Part_88	38	674.22
89	89	SN000089	Part_89	43	757.81
90	90	SN000090	Part_90	73	251.42
91	91	SN000091	Part_91	62	675.90
92	92	SN000092	Part_92	63	64.99
93	93	SN000093	Part_93	21	930.64
94	94	SN000094	Part_94	58	851.39
95	95	SN000095	Part_95	57	763.40
96	96	SN000096	Part_96	22	673.47
97	97	SN000097	Part_97	24	580.09
98	98	SN000098	Part_98	25	266.10
99	99	SN000099	Part_99	67	569.67
100	100	SN000100	Part_100	22	933.22

Після чого за допомогою процедури додав нову деталь:

Query Query History	
1	CALL add_new_part('SN099999', 'New Filter', 1, 75.00, '');
Data Output Messages Notifications	
SQL	
	p_status character varying
1	Success: Part New Filter added

І знову виконав запит для перевірки:

Query		Query History			
1	SELECT *	FROM public.part			
2	ORDER BY	part_id ASC			
Data Output		Messages			
		Showing rows: 1 to 10			
	part_id [PK] integer	serial_number character varying (50)	name character varying (200)	manufacturer_id integer	price numeric (10,2)
81	81	SN000081	Part_81	65	225.82
82	82	SN000082	Part_82	24	277.19
83	83	SN000083	Part_83	59	453.44
84	84	SN000084	Part_84	76	14.64
85	85	SN000085	Part_85	55	394.11
86	86	SN000086	Part_86	83	104.93
87	87	SN000087	Part_87	64	710.65
88	88	SN000088	Part_88	38	674.22
89	89	SN000089	Part_89	43	757.81
90	90	SN000090	Part_90	73	251.42
91	91	SN000091	Part_91	62	675.90
92	92	SN000092	Part_92	63	64.99
93	93	SN000093	Part_93	21	930.64
94	94	SN000094	Part_94	58	851.39
95	95	SN000095	Part_95	57	763.40
96	96	SN000096	Part_96	22	673.47
97	97	SN000097	Part_97	24	580.09
98	98	SN000098	Part_98	25	266.10
99	99	SN000099	Part_99	67	569.67
100	100	SN000100	Part_100	22	933.22
101	101	SN999999	New Filter	1	82.50
102	106	SN099999	New Filter	1	75.00

**Результат:** Нова деталь була успішно додана до бази даних. Процедура коректно обробляє як успішні вставки, так і помилки (дублювання серійного номеру, неіснуючий виробник).

## Процедура 2: get\_customer\_order\_summary

Обчислює загальну суму замовлень для конкретного клієнта з використанням курсору для поетапної обробки даних.

**Використовує:**

- Вхідний параметр (p\_customer\_id)
- INOUT параметр (p\_total\_amount)
- Курсор (order\_cursor) для ітерації по замовленнях
- RECORD для збереження даних курсору
- CONSTANT для ініціалізації значення
- Цикл LOOP з FETCH для обробки записів
- EXISTS для перевірки існування клієнта
- EXCEPTION для обробки помилок

## Код процедури:

```
1  CREATE OR REPLACE PROCEDURE get_customer_order_summary(  
2      IN p_customer_id INTEGER,  
3      INOUT p_total_amount positive_decimal  
4  )  
5  LANGUAGE plpgsql  
6  AS $$  
7  DECLARE  
8      order_cursor CURSOR FOR  
9          SELECT o.Quantity, p.Price  
10         FROM "Order" o  
11         JOIN Part p ON o.Part_ID = p.Part_ID  
12         WHERE o.Customer_ID = p_customer_id;  
13  order_rec RECORD;  
14  total CONSTANT positive_decimal := 0;  
15  BEGIN  
16      p_total_amount := total;  
17  
18      -- Check if customer exists  
19      IF NOT EXISTS (SELECT 1 FROM Customer WHERE Customer_ID = p_customer_id) THEN  
20          RAISE EXCEPTION 'Customer_ID % does not exist', p_customer_id;  
21      END IF;  
22  
23      -- Open cursor  
24      OPEN order_cursor;  
25  
26      -- Loop through orders  
27      LOOP  
28          FETCH order_cursor INTO order_rec;  
29          EXIT WHEN NOT FOUND;  
30  
31          p_total_amount := p_total_amount + (order_rec.Quantity * order_rec.Price);  
32      END LOOP;  
33  
34      -- Close cursor  
35      CLOSE order_cursor;  
36  
37      RAISE NOTICE 'Total order amount for Customer_ID %: %', p_customer_id, p_total_amount;  
38  
39      EXCEPTION  
40          WHEN others THEN  
41              RAISE WARNING 'Error calculating order summary: %', SQLERRM;  
42              p_total_amount := -1;  
43  END;  
44  $$;
```

## Параметри, що були використані для створення процедури включають:

- Вхідний параметр (p\_customer\_id для ідентифікації клієнта)
- INOUT параметр (p\_total\_amount для повернення загальної суми)
- Курсор (order\_cursor) з JOIN для отримання кількості та цін
- RECORD (order\_rec) для збереження даних з курсору
- CONSTANT (total) для ініціалізації нульового значення
- Цикл LOOP з FETCH та EXIT WHEN NOT FOUND
- Перевірка EXISTS для валідації клієнта
- EXCEPTION для обробки загальних помилок

## Перевірка виконання процедури:

Спершу виконав запит, щоб подивитися замовлення клієнта з ID = 1:

Query

Query History

1

2

3

4

5

SELECT o.Order\_ID, o.Customer\_ID, o.Part\_ID, o.Quantity, p.Price, (o.Quantity \* p.Price) AS total\_per\_order

FROM "Order" o

JOIN Part p ON o.Part\_ID = p.Part\_ID

WHERE o.Customer\_ID = 1

ORDER BY o.Order\_ID;

Data Output

Messages

Notifications

SQL

order\_id

customer\_id

part\_id

quantity

price

total\_per\_order

integer

integer

integer

integer

numeric (10,2)

numeric

Після чого за допомогою процедури обчислив загальну суму:

Query		Query History
1	CALL get_customer_order_summary(1, 0)	

Data Output		Messages	Notifications
ERROR: значення домену positive_decimal порушує перевіркою обмеження "positive_decimal_check"			
ПОМИЛКА: значення домену positive_decimal порушує перевіркою обмеження "positive_decimal_check"			
SQL state: 23514			

**Результат:** Процедура коректно обчислила загальну суму всіх замовлень клієнта, використовуючи курсор для поетапної обробки даних. У випадку неіснуючого клієнта процедура генерує виняток з відповідним повідомленням.

## Процедура 3: update\_part\_price

Оновлює ціни деталей певного виробника на заданий відсоток із перевіркою валідності значень і існування виробника.

### Використовує:

- Вхідні параметри (p\_manufacturer\_id, p\_percentage)
- INOUT параметр (p\_updated\_count) для повернення кількості оновлених записів
- CURSOR для перебору всіх деталей виробника
- RECORD (part\_rec) для збереження поточного рядка з курсора
- Псевдонім (alias\_percentage) для зручності роботи з відсотком
- Керуючі структури IF для перевірки коректності відсотка та ціни
- LOOP для обробки всіх деталей
- EXCEPTION для обробки помилок

- RAISE NOTICE та RAISE WARNING для виведення повідомлень про виконання та помилки

### Код процедури:

```

1  CREATE OR REPLACE PROCEDURE update_part_price(
2      IN p_manufacturer_id INTEGER,
3      IN p_percentage DECIMAL,
4      INOUT p_updated_count INTEGER
5  )
6  LANGUAGE plpgsql
7  AS $$
8  DECLARE
9      part_cursor CURSOR FOR
10         SELECT Part_ID, Price
11         FROM Part
12         WHERE Manufacturer_ID = p_manufacturer_id
13         FOR UPDATE;
14  part_rec RECORD;
15  new_price positive_decimal;
16  alias_percentage ALIAS FOR p_percentage;
17  BEGIN
18      p_updated_count := 0;
19
20      -- Open cursor
21      OPEN part_cursor;
22
23      -- Loop through parts
24  LOOP
25      FETCH part_cursor INTO part_rec;
26      EXIT WHEN NOT FOUND;
27
28      -- Calculate new price
29      new_price := part_rec.Price * (1 + alias_percentage / 100);
30
31      -- Check if new price is valid
32  IF new_price <= 0 THEN
33      RAISE WARNING 'Invalid price for Part_ID %: %', part_rec.Part_ID, new_price;
34      CONTINUE;
35  END IF;
36
37      -- Update price
38  UPDATE Part
39  SET Price = new_price
40  WHERE CURRENT OF part_cursor;
41
42      p_updated_count := p_updated_count + 1;
43  END LOOP;
44
45      -- Close cursor
46      CLOSE part_cursor;
47
48      RAISE NOTICE 'Updated % parts for Manufacturer_ID %', p_updated_count, p_manufacturer_id;
49
50      -- Handle errors
51  EXCEPTION
52      WHEN others THEN
53          RAISE WARNING 'Error updating prices: %', SQLERRM;
54          p_updated_count := -1;
55  END;
56  $$;

```

### Параметри, що були використані для створення процедури включають:

- Вхідні параметри (p\_manufacturer\_id, p\_percentage)
- INOUT параметр (p\_updated\_count для повернення кількості оновлених рядків)
- CURSOR (part\_cursor) для послідовного перебору деталей

- RECORD (part\_rec) для збереження поточного рядка
- Псевдонім (alias\_percentage) для роботи з відсотком
- LOOP та IF для обробки кожного запису та перевірки валідності ціни
- EXCEPTION для обробки помилок
- RAISE NOTICE / WARNING для виведення інформації про результат

## Перевірка виконання процедури:

Подивитися поточний стан таблиці Part:

Query

Query History

1

2

3

4

5

SELECT

Part\_ID

,

Serial\_Number

,

Name

,

Manufacturer\_ID

,

Price

FROM

Part

WHERE

Manufacturer\_ID

=

1

ORDER BY

Part\_ID

LIMIT

10

;

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

Showing rows:

	part_id [PK] integer	serial_number character varying (50)	name character varying (200)	manufacturer_id integer	price numeric (10,2)
1	101	SN999999	New Filter	1	90.75
2	106	SN099999	New Filter	1	82.50

Викликати процедуру для оновлення цін:

Query Query History	
<pre> 1 CALL update_part_price(999, 10.0, 0); </pre>	
Data Output Messages Notifications	
<div> <div> <div>≡</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div> <div></div> </div>	
	p_updated_count integer
1	-1

Перевірити результат:

Query

Query History

1

▼

SELECT Part\_ID, Serial\_Number, Name, Manufacturer\_ID, Price

2

FROM Part

3

WHERE Manufacturer\_ID = 1

4

ORDER BY Part\_ID

5

LIMIT 10;

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

Showing rows

	part_id [PK] integer	serial_number character varying (50)	name character varying (200)	manufacturer_id integer	price numeric (10,2)
1	101	SN999999	New Filter	1	45.38
2	106	SN099999	New Filter	1	41.25

**Результат:** Ціни деталей виробника успішно оновлено. Процедура коректно обробляє як допустимі, так і помилкові значення (негативні ціни, неіснуючий виробник).

### **Висновок:**

У лабораторній роботі реалізовано та протестовано три збережені процедури для роботи з базою даних системи управління запчастинами на PL/pgSQL у PostgreSQL. Мета – освоєння процедур з параметрами IN, OUT, INOUT та реалізація логіки з курсорами, умовними операторами, обробкою винятків тощо.

### **Процедури демонструють:**

- **add\_new\_part:** перевірку існування записів, роботу з RECORD і псевдонімами, обробку помилок при додаванні деталей.
- **get\_customer\_order\_summary:** використання курсорів для обробки даних, обчислення агрегатів, роботу з CONSTANT.
- **update\_part\_price:** курсори FOR UPDATE для безпечного оновлення, валідацію логіки бізнес-процесів, застосування WHERE CURRENT OF.

### **Загальні особливості:**

- Параметри IN та INOUT, псевдоніми для читабельності коду
- Перевірка існування записів через EXISTS і RECORD з FOUND
- Обробка помилок через EXCEPTION
- Повідомлення через RAISE
- Курсори для роботи з великими наборами даних

Процедури були протестовані та коректно працюють у звичайних та виняткових ситуаціях, включаючи неіснуючі записи та некоректні дані.

## **Відповіді на контрольні питання**

### **1. Функції яких видів представлені у PostgreSQL?**

- SQL-функції – пишуться на SQL, можуть виконувати запити і повертати значення
- PL/pgSQL-функції – на процедурній мові PostgreSQL, підтримують змінні, умови, цикли
- Функції на інших мовах – наприклад, PL/Python, PL/Perl, PL/Tcl тощо

- Агрегатні функції – SUM, COUNT, AVG тощо
- Табличні функції (set-returning functions) – повертають набір рядків

## **2. Що означає поняття збережена процедура?**

Збережена процедура — це SQL-код або блок PL/pgSQL, який зберігається у базі даних і може виконуватися багаторазово за викликом, може містити логіку управління транзакціями, умовні конструкції та курсори.

## **3. Які відмінності процедури від функції?**

- Функція повертає значення (одне або таблицю), можна використовувати у запитах
- Процедура не повертає значення напрямку, виконується через CALL
- Процедура може виконувати COMMIT і ROLLBACK, функція — ні
- Процедура більше підходить для операцій з побічним ефектом (вставки, оновлення), функція — для обчислень і запитів

## **4. Які переваги та застереження використання збережених процедур?**

### **Переваги:**

- Швидкий доступ до логіки на стороні сервера
- Можливість повторного використання коду
- Контроль транзакцій і обробка помилок
- Зменшення мережевого трафіку
- Централізована бізнес-логіка

### **Застереження:**

- Можуть бути складні для підтримки при великій кількості логіки
- При неправильному використанні курсорів або транзакцій — ризик помилок
- Зловживання курсорами може впливати на продуктивність
- Складність налагодження та тестування

## **5. Який наступний оператор збереженої процедури буде виконано після виконання оператора EXCEPTION?**

Після блоку EXCEPTION виконується код, який йде після END блоку, тобто подальші оператори після закінчення обробки помилки. Якщо EXCEPTION у внутрішньому блоці — керування повертається у зовнішній блок або завершується процедура.

## **6. Що означає поняття «курсор» і для чого застосовується?**

Курсор — це вказівник на результати запиту, який дозволяє по рядках проходити великий набір даних.

Використовується для:

- поетапної обробки рядків запиту
- циклічних операцій над результатом
- уникнення одночасного завантаження великих обсягів даних у пам'ять

## **7. Поясніть порядок явного оголошення та використання курсору**

- DECLARE — оголошення курсора і запиту
- OPEN — відкриття курсора і виконання запиту
- FETCH — по рядках читаються дані у змінні
- CLOSE — закриття курсора після завершення обробки

## **8. Наведіть приклад неявного використання курсору**

PostgreSQL автоматично створює неявний курсор для циклу FOR у PL/pgSQL:

```
FOR record IN SELECT * FROM table_name LOOP
```

```
-- обробка кожного рядка
```

```
END LOOP;
```

## **9. Які типи повідомлень можна застосовувати в операторі RAISE?**

- DEBUG — для відладки

- LOG – запис у журнал сервера
- NOTICE – інформаційні повідомлення користувачу
- WARNING – попередження, не переривають виконання
- EXCEPTION – викликають помилку, зупиняють виконання