# СТВОРЕННЯ ТА ВИКОНАННЯ ЗБЕРІГАЄМИХ ПРОЦЕДУР

*Зберігаєма процедура (Storage Procedure)* це традиційна назва процедур, написаних на внутрішній мові конкретної СУБД, до якої можна звернутися з конкретного програмного додатку.

Зберігаємапроцедура в PostrgeSQL називається *зберігаємою функцією*. Насправді вона поводиться як функція, яка повертає одне значення заданого типу, і як процедура, яка може повертати декілька значень відповідних типів.

Для створення зберігаємої процедури можна використовувати 2-ва підходи:

1. Самостійне формування тексту зберігаємої процедури у вікні SQL-запи­тів користувача;

2. Використати майстер створення зберігаємої функції.

## Формування тексту зберігаємої процедури у вікні SQL-запитів користувача

Натиснути кнопку "Виконати SQL-запити користувача" (Рисунок 1.1)

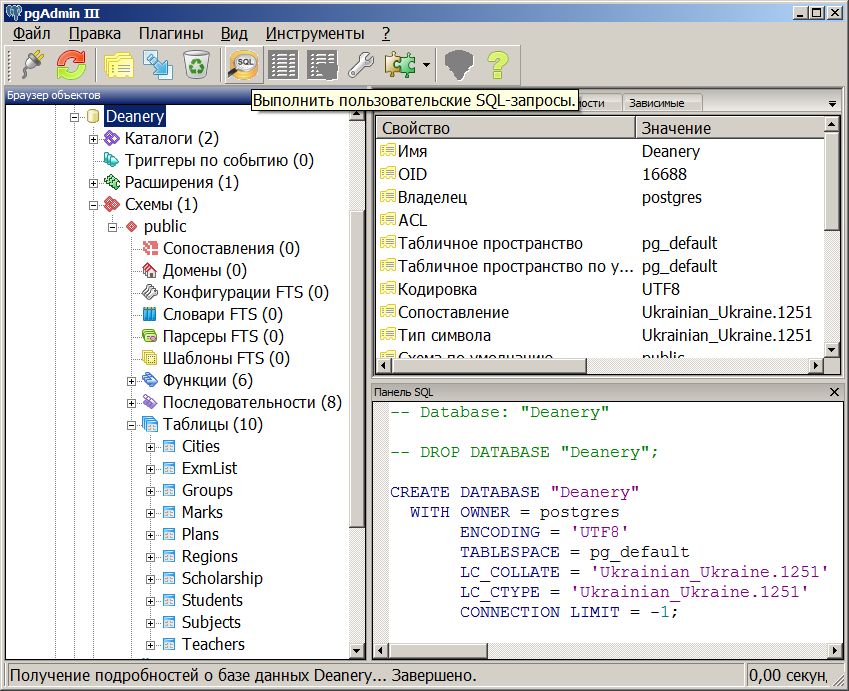


Рисунок 1.1 – Кнопка Виконати SQL-запити

У вікні, що відчиниться, записати текст функції, яка, наприклад, по назві групи повертає список її студентів (Рисунок 1.2).

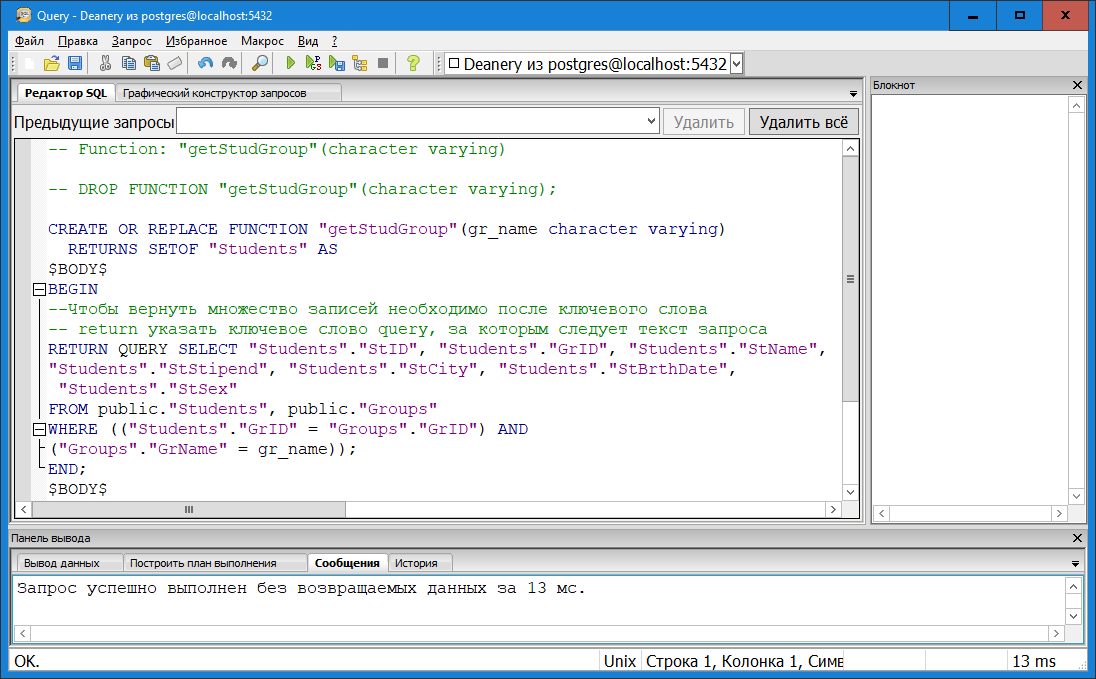


Рисунок 1.2 - Текст функції, яка по назві групи повертає список її студентів

## Використання майстра створення зберігаємої процедури

У контекстному меню елементу Функції вибрати пункт "Нова функція ..." (Рисунок 1.3).

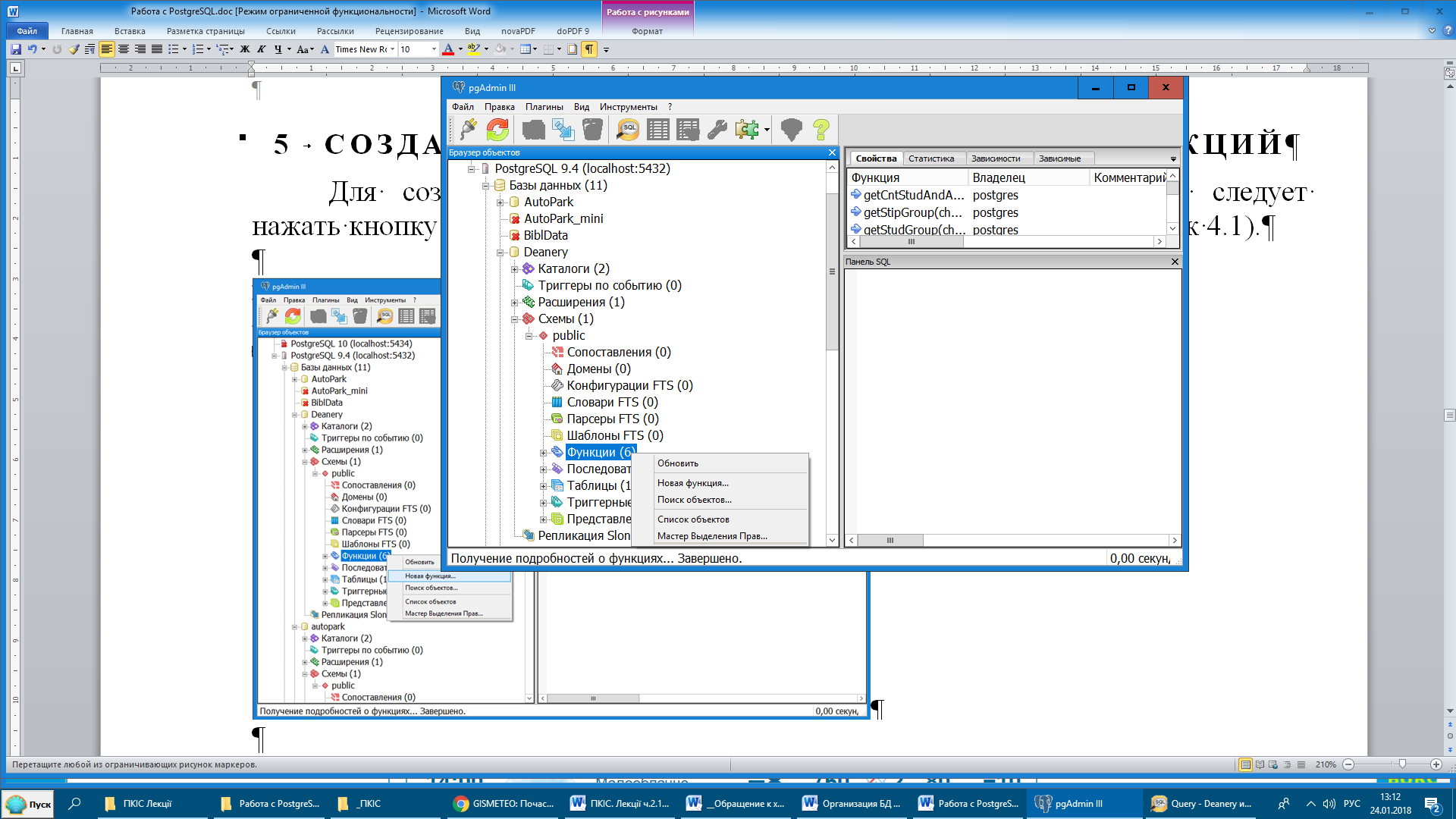


Рисунок 1.3 - Пункт Функції

Відкриється вікно визначення функції

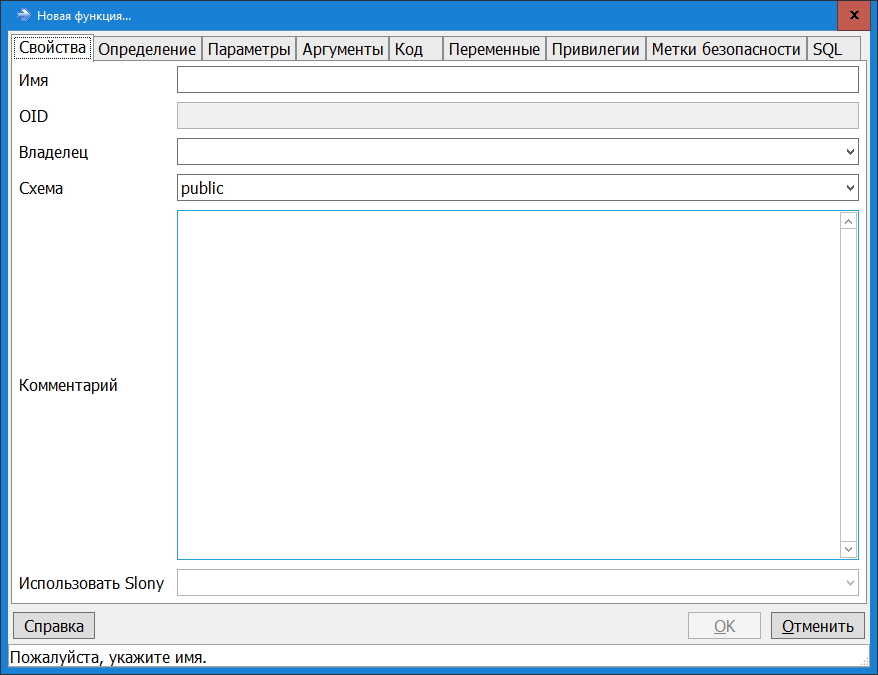


Рисунок 1.4 – Нова функція

У загальному випадку для створення функції необхідно пройти по всім закладкам, на кожній з яких визначається окремий елемент опису функції. Однак, на практиці, це незручно, тому рекомендується виконати початкові дії і потім перейти до редагування функції в екранному редакторі SQL-запитів.

Нехай потрібно створити функцію, яка за назвою групи повертає список її студентів. Для цього необхідно:

1. 1. Задати ім'я функції getStudGroup

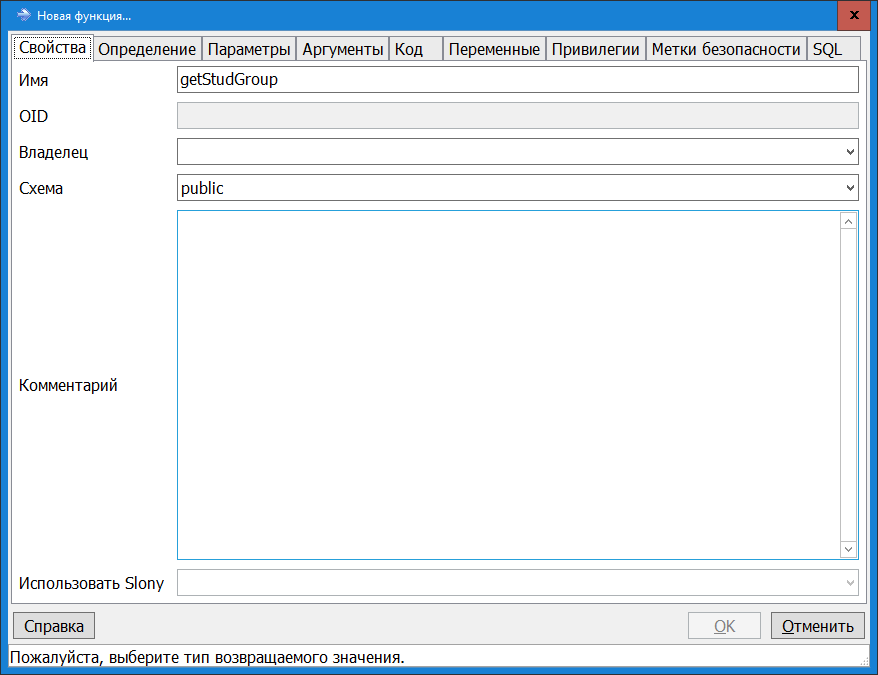


Рисунок 1.5 – Ім'я функції

1. Вказати допустимий в PostgreSQL тип повертаємого значення

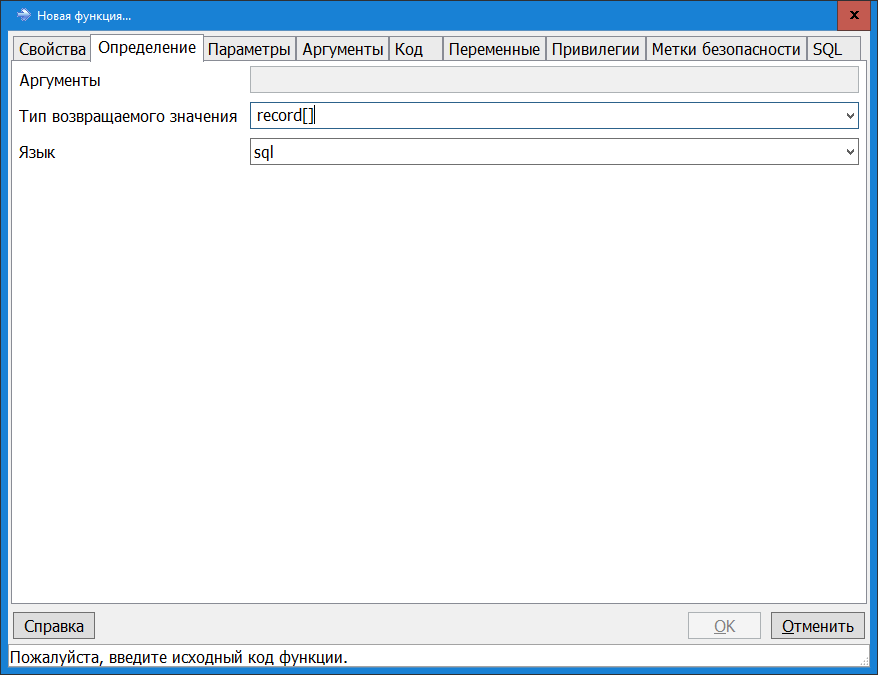
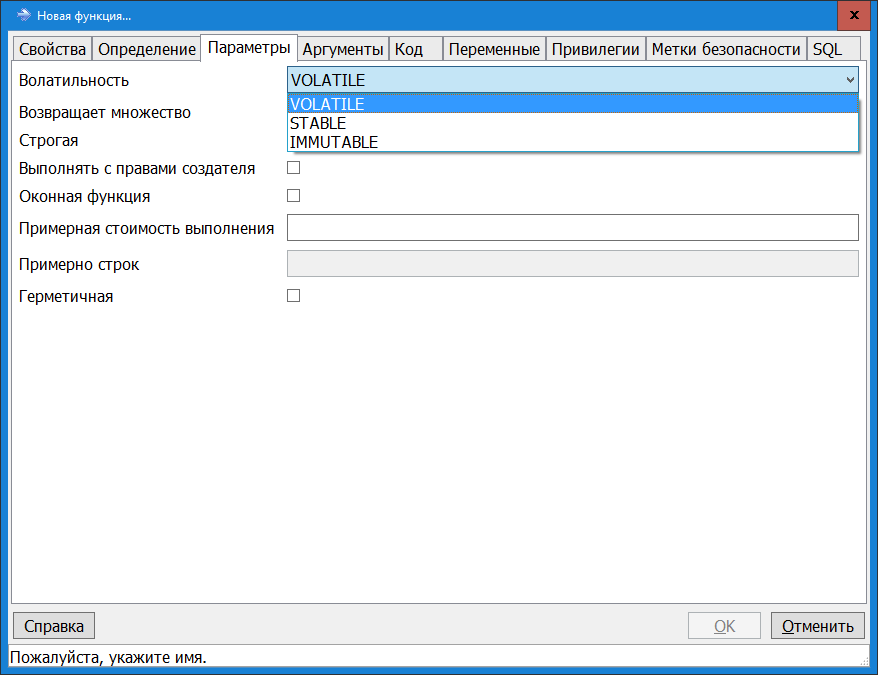


Рисунок 1.6 - Тип повертаємого значення

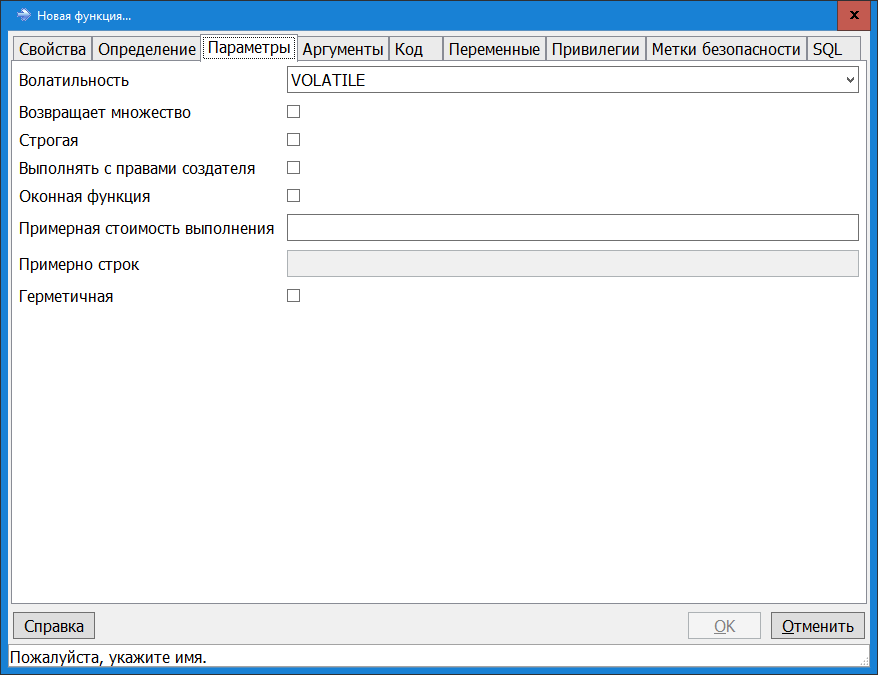
Слід підкреслити, що синтаксис PL/pgsql має більш виразні можливості по завданню типу повертаємого результату, але, в будь-якому випадку, ці типи зводяться до допустимих в PostgreSQL типів значень.

3. Вказати *Волатильність (Мінливість) - VOLATILE* [7] значення функції. Волатильність вказує на те, що значення функції може змінюватися в межах одного сканування таблиці (використовується за умовчанням).

Так само тут можна вказати, що функція повертає набір значень. Це може бути проста таблиця або таблиця БД (Рисунок 1.7);



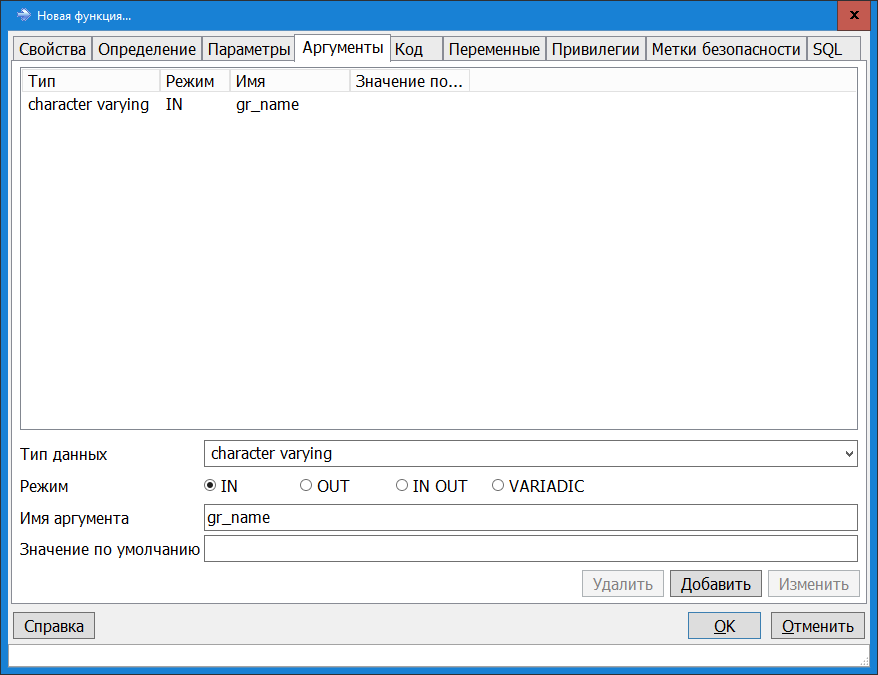
а) – Тип волатильності



б) - Значення волатильності

Рисунок 1.7 – Тип та значення волатильності

1. Вказати необхідні аргументи функції. Це можна зробити і пізніше.



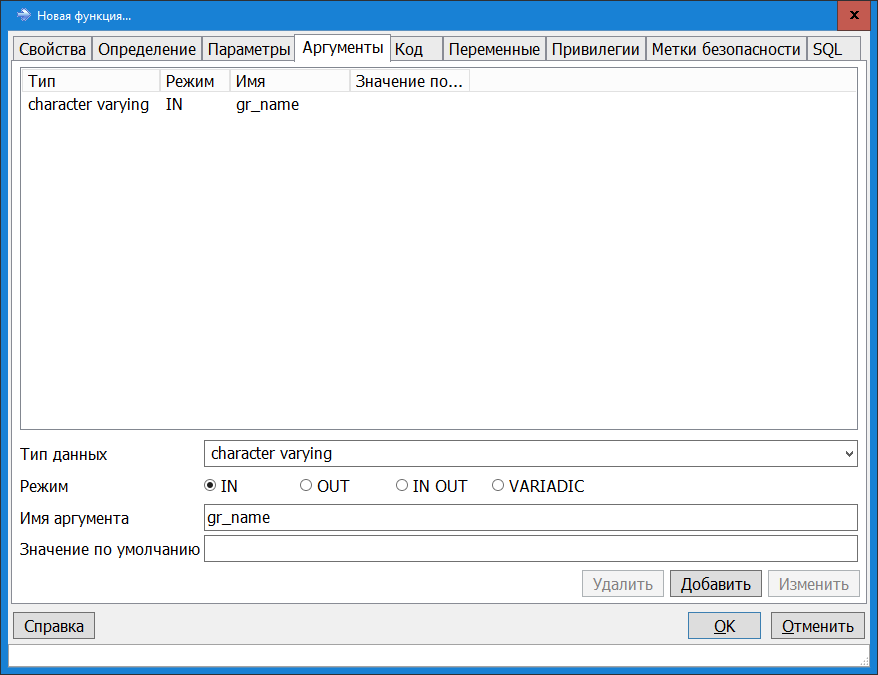


Рисунок 1.8 - аргументи функції

1. Задати код

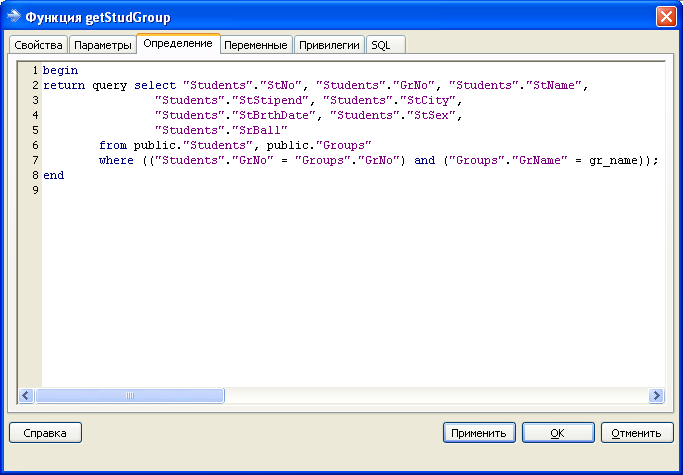


Рисунок 1.9 – Код функції

1. Отриманий текст функції можна побачити на закладці SQL

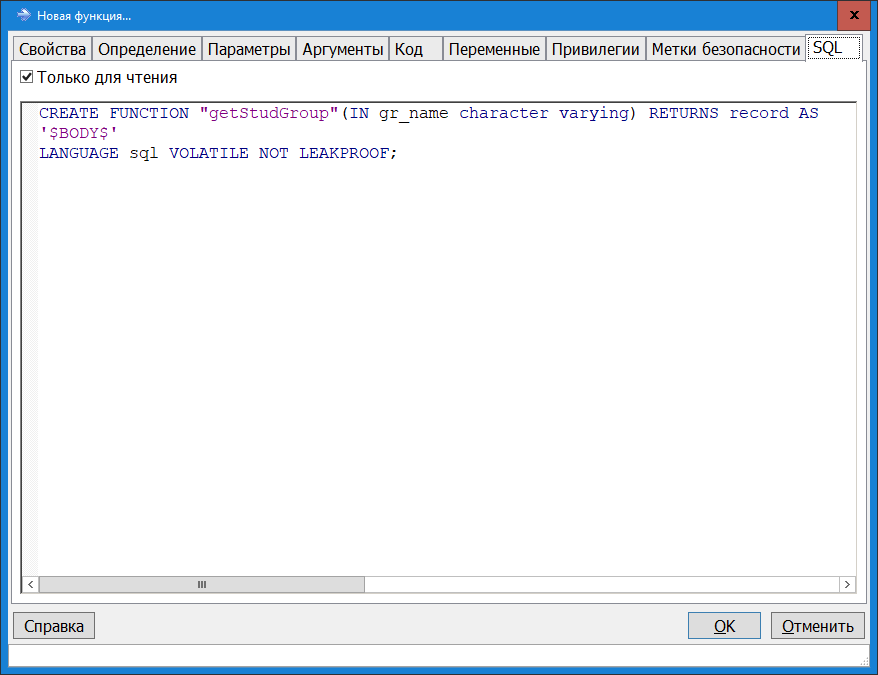


Рисунок 1.10 - Текст функції

## Редагування зберігаємої процедури

Для редагування зберігаємої функції слід в контекстному меню необхідної функції вибрати пункт "Скрипт CREATE" (Рисунок 1.11).

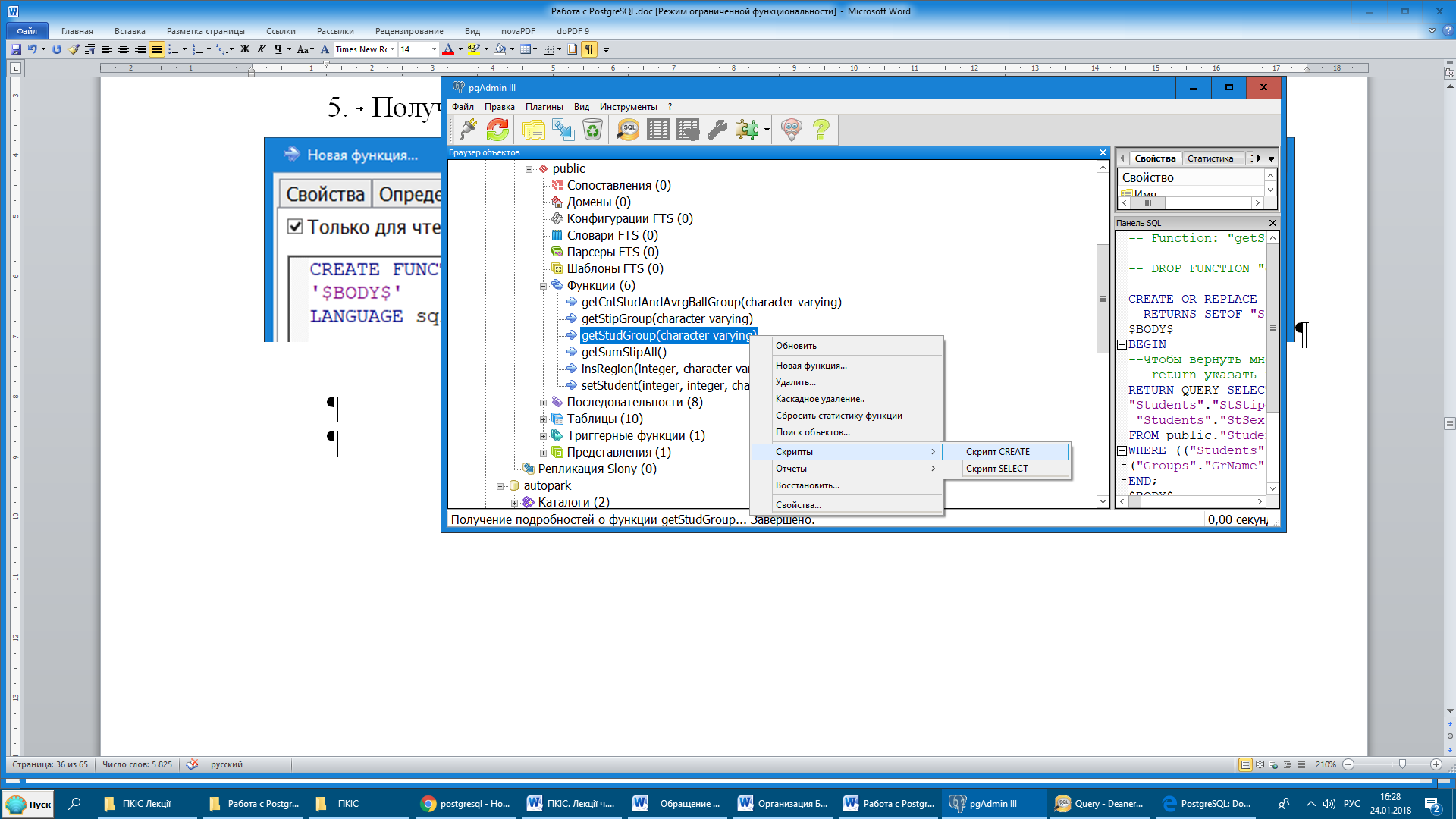


Рисунок 1.11 – Вибір скрипту

Тепер в екранному редакторі можна остаточно сформувати текст функції і відправити її на виконання (Рисунок 1.12 та Листинг 1.1)

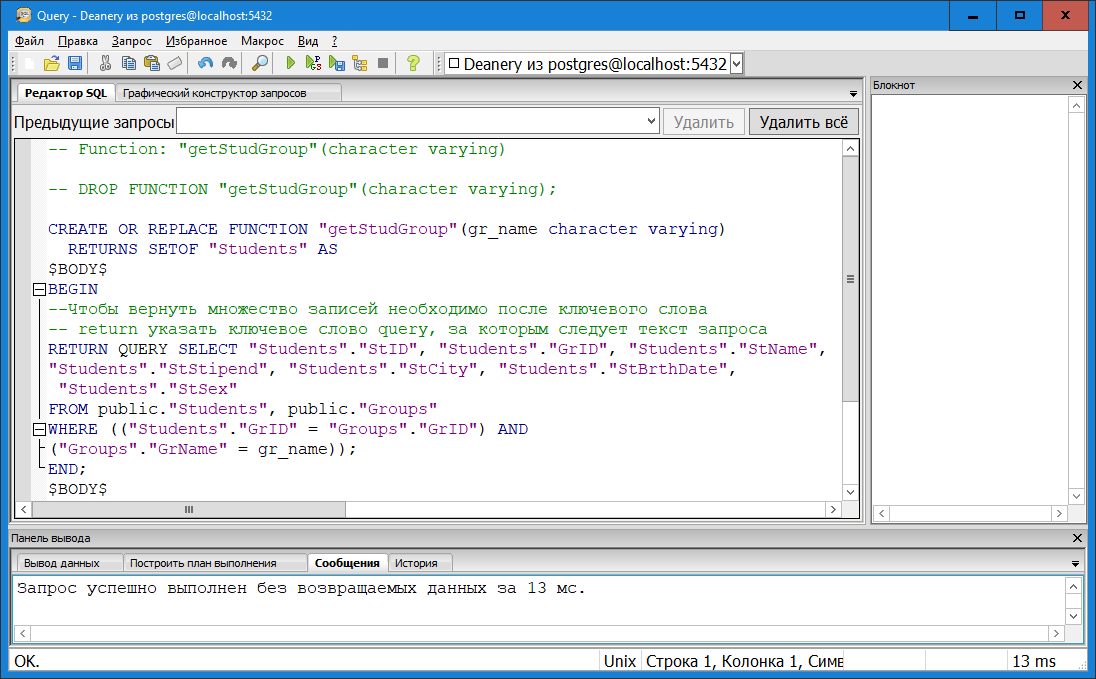


Рисунок 1.12 – Відредагований текст функції

|  |
| --- |
| **getStudGroup\_1** |
| -- Function: "getStudGroup\_1"(character varying)  -- DROP FUNCTION "getStudGroup\_1"(character varying);  CREATE OR REPLACE FUNCTION "getStudGroup\_1"(gr\_name character varying)  RETURNS SETOF "Students" AS  $BODY$  BEGIN  --Чтобы вернуть множество записей необходимо после ключевого слова  -- return указать ключевое слово query, за которым следует текст запроса  RETURN QUERY SELECT "Students"."StNmb", "Students"."StName",  "Students"."StSex", "Students"."StBrthDate", "Students"."StCity",  "Students"."StStipend", ""Students"."StAvgBall", Students"."StID",  "Students"."GrID",  FROM public."Students", public."Groups"  WHERE (("Students"."GrID" = "Groups"."GrID") AND  ("Groups"."GrName" = gr\_name));  END;  $BODY$  LANGUAGE plpgsql VOLATILE  COST 100  ROWS 1000;  ALTER FUNCTION "getStudGroup\_1"(character varying)  OWNER TO postgres; |

Листинг 1.1 – Відредагований текст функції

## Виклик зберігаємих процедур в pgAdmin і прикладній програмі

Узагальнений виклик функції мови PL/pgSQL виглядає наступним чином:

**SELECT** \* **FROM** public."Им’я функції"([параметр, …]);

Так для наведеного вище прикладу виклик функції має вигляд:

**SELECT** **\*** **FROM** public."getStudGroup"('КС-981');

### Виклик зберігаємої процедури в pgAdmin

Для виклику функції необхідно у вікні SQL-команд ввести відповідну команду і, якщо потрібно, параметри та відправити її на виконання (Рисунок 1.13)

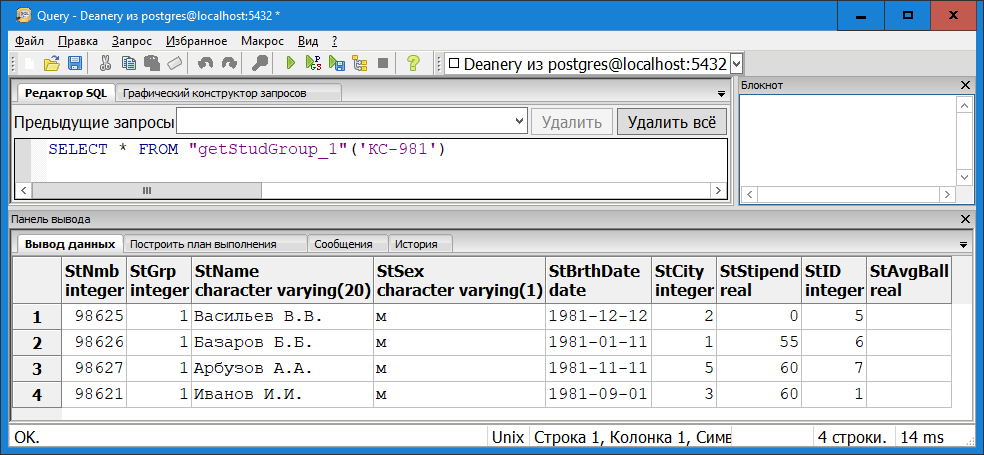


Рисунок 1.13 - Виклик зберігаємої процедури

### Виклик зберігаємої процедури в прикладній програмі

Можливі 2-ва варіанта виклику зберігаємої процедури в прикладної про- грамі.

Перший варіант має місце, якщо виклик процедури не має параметрів або значення цих параметрів задані явно (статично), наприклад

**SELECT** **\*** **FROM** public."getStudGroup"('КС-981');

В цьому випадку виклик процедури розглядається як звичайний SQL-запит і виконується відповідними засобами мови програмування.

Другий варіант має місце, якщо виклик процедури має параметри і значення цих параметрів задаються динамічно. В цьому випадку в мовах програмування передбачаються спеціальні засоби для роботи зі зерігаємими процедурами, підготовки їх вхідних параметрів і виконання процедури.

Після виконання процедури необхідно виділити по черзі повертаємі значення полів оператора Select.

### Виклик зберігаємих процедур в Java засобами JDBC

Виклик зберігаємих процедур з Java-додатку практично нічим не відрізняється від виконання звичайного SQL-запиту на вибірку даних. Інтеграція з JDBC - це величезна перевага для зерігаємих процедур, оскільки для того, щоб їх викликати з програми, не потрібно змінювати класи або використовувати будь-які конфігураційні файли.

JDBC підтримує виклик зерігаємих процедур за допомогою класу CallableStatement. Цей клас є підкласом класу PreparedStatement. Рядок, який подається в якості параметра методу prepareCall() - це специфікація виклику процедури. Вона визначає ім'я викликаємої процедури і символи '?', які визначають необхідні параметри. Можливі 2-ва варіанта специфікація виклику процедури:

* якщо функція повертає значення

{?= call <procedure-name>[(<arg1>, <arg2>, ...)]}

* якщо функція не повертає значення

{call <procedure-name>[(<arg1>, <arg2>, ...)]}

У загальному випадку процедура повертає набір рядків. Нумерація стовпців в цьому наборі починається з 1. Зокрема, процедура може повернути тільки 1-н рядок або тільки 1-не значення.

При виконанні процедур можливе виникнення помилкових ситуацій, в цьому випадку буде згенеровано виняток SQLException.

Розглянемо варіанти виклику зерігаємих процедур, наведених в Рисунок 1.11.

*Приклад 1. Виклик функції додати відомості про новий студента*

insStudent(st\_no integer, st\_name character varying)

Функція з 2-ма вхідними параметрами та значень не повертає

|  |
| --- |
| connection.setAutoCommit(**false**);  **CallableStatement** proc = connection.prepareCall("{call insStudent( ? , ? ) }");  proc.setInt(1, 958977);  proc.setString(2, "Іванов");  proc.execute(); |

У функції insStudent() використовується 2-ва параметра. Для присвоєння параметрам значень використовуються методи setInt(1, 958 977) для завдання значення типу int першому параметру і setString(2, "Іванов") для завдання значення типу String другому параметру функції insStudent().

*Приклад 2. Виклик функції підрахувати суму витрат на стипендію всіх студентів.*

getSumStipAll()

Функція не має вхідних параметрів і повертає одне значення типу real.

|  |
| --- |
| connection.setAutoCommit(**false**);  **CallableStatement** proc = connection.prepareCall("{ ?= call getSumStipAll() }");  proc.registerOutParameter(1, Types.REAL);  proc.execute();  **float** sum = proc.*getFloat(1)*; |

За допомогою методу registerOutParameter() реєструється SQL-тип вихідного параметра. В Java замість типу real використовується тип float, і тому для отримання результату функції типу float використовується метод getFloat().

*Приклад 3. Виклик функції підрахувати суму витрат на стипендію всіх студентів групи Х.*

getStipGroup(gr\_name character varying)

Функція з одним вхідним параметром і повертає одне значення

|  |
| --- |
| connection.setAutoCommit(**false**);  **CallableStatement** proc = connection.prepareCall("{ ?= call getStipGroup(?) }");  proc.registerOutParameter(1, Types.REAL);  proc.setString(2, "КС-981");  proc.execute();  **float** sum = proc.*getFloat(1)*; |

З точки зору процедури, PostgreSQL у неї 2-ва параметра. Перший параметр вихідний OUT - результат функції. Другий параметр - вхідний IN. Для його завдання використовується метод setString(2, "КС-071"), де значення 2 вказує, що другому параметру зерігаємої процедури, тобто першому параметру функції getStipGroup() необхідно присвоїти рядкове значення.

*Приклад 4. Виклик функції підрахувати кількість студентів і їх середню стипендію в групі Х.*

getStipGroup(gr\_name character varying)

Функція з одним вхідним параметром і повертає 2-ва значення

|  |
| --- |
| connection.setAutoCommit(**false**);  **CallableStatement** proc = connection.prepareCall("{ ?= call getCntStudAndAvrgStipGroup (?) }");  proc.registerOutParameter(1, Types.INTEGER);  proc.registerOutParameter(2, Types.REAL);  proc.setString(3, "КС-981");  proc.execute();  **float** sum = proc.*getFloat(1)*; |

У зерігаємої процедури 3-ри параметра. 1-ий і 2-ий параметри вихідні OUT. 3-ий параметр - вхідний IN. Для його завдання використовується метод setString(3, "КС-981"), де значення 3 вказує, що 3-му параметру зерігаємої процедури, тобто 1-му параметру функції getCntStudAndAvrgStipGroup() необхідно присвоїти рядкове значення.

*Приклад 5. Отримати список студентів групи. Функція з одним вхідним параметром і повертає безліч рядків (записів) таблиці*

getStudGroup(gr\_name character varying).

|  |
| --- |
| connection.setAutoCommit(**false**);  **CallableStatement** proc = connection.prepareCall("{ ?= call getStudGroup(?) }");  proc.registerOutParameter(1, Types.OTHER);  proc.setString(2, "КС-981");  proc.execute();  **ResultSet** rs = (**ResultSet**) proc.*getObject(1)*;  **while** (rs.next()){  String name = rs.getString(3);  **float** stip = proc.getFloat(4);  System.out.println("Cтудент " + name + " стипендия " + stip);  }  rs.close(); |

Результатом цієї функції є набір рядків з таблиці Students. Набір рядків розглядається як об'єкт класу ResultSet. За допомогою методу registerOutParame­ter(1, Types.OTHER) реєструється тип вихідного параметра OUT як Types.OTHER. В Java для зберігання об'єктів цього типу використовується клас Object. Після виклику зерігаємої процедури необхідно провести спадаюче перетворення до типу ResultSet. Далі використовується стандартний спосіб обходу записів об'єкта ResultSet.

### Виклик зерігаємих процедур або функцій в Python

Для виконання зерігаємих процедур або функції PostgreSQL в Python можна використати модуль psycopg2.

Виклик зерігаємих процедур з Python -додатку практично нічим не відрізняється від виконання звичайного SQL-запиту на вибірку даних.

Модуль psycopg2 підтримує виклик зерігаємих процедур за допомогою звернення

cursor.callproc()

Процес виклику та виконання зерігаємих процедур або функції складається з наступних кроків:

1. Імпорт в файл зі зверненням

|  |
| --- |
| **import psycopg2** |

1. Створення об'єкту з'днання з БД

|  |
| --- |
| ps\_connection = psycopg2.connect(Connection Arguments) |

1. Створення об'єкту курсор

|  |
| --- |
| cursor = ps\_connection.cursor() |

1. Виклик та виконання зерігаємої процедури або функції, використовуючи метод cursor.callproc(). Тут необхідно знати назву зерігаємої процедури та її параметри IN та OUT. Параметри IN та OUT мають бути розділені комами.

|  |
| --- |
| cursor.callproc('Ім'я функції або процедури', [IN або OUT параметри, ]) |

Процес поверне результат виклику callproc(). Це можуть бути рядки БД або що завгодно відповідно до реалізації функції.

Треба перехоплювати будь-які винятки SQL, які можуть виникнути під час цього процесу;

1. По завершенню треба закрийте об'єкт курсору та з'єднання з БД.

Можна побачити, що функція отримує вхідний параметр – назву групи і повертає перелік студенті ціє групи.

|  |
| --- |
| **import** psycopg2  **from** psycopg2 **import** Error  **try**:  psy\_connection = psycopg2.connect(user="postgres",  password="qwerty",  host="127.0.0.1",  port="5432",  database="deanery")    cursor = psy\_connection.cursor()    #виклик зберігаємої процедури  cursor.callproc('getStudGroup\_1', ['КС-972', ])  print("Перелік студентів групи")  result = cursor.fetchall()  **for** row **in** result:  print("StNmb = ", row[0],)  print("StName = ", row[1])  print("StSex = ", row[2])    **except** (Exception, psycopg2.DatabaseError) as error :  print ("Помилка при з'єднанні з PostgreSQL", error)  **finally**:  #закриття з'єднання з БД  **if**(ps\_connection):  cursor.close()  ps\_connection.close()  print("з'єднанні з PostgreSQL закрите") |

# СТВОРЕННЯ ТА ВИКОНАНЯ ТРИГЕРІВ

Тригер БД - це процедура, яка зберігається у вигляді об'єкта БД і виконання якої неявно зініціюється при настанні певних подій. Як правило, такими подіями є зміни в стані БД, які ініціюються пропозиціями INSERT, UPDATE, DELETE.

Слід зазначити, що тригер в PostgreSQL - це зв'язок тригерної функції з одною (або декількома) з операцій модифікації, які виконуються над таблицею (UPDATE, INSERT, DELETE), та автоматично запускається при отриманні відповідного запиту. Тобто, з кожним тригером повинна бути пов'язана тригерна функція.

У PostgreSQL тригерна функція відрізняється від звичайної функції тим, що тип її результату - trigger.

## Синтаксис визначення тригера в PostgreSQL

Синтаксис визначення тригера в PostgreSQL наступний::

**CREATE TRIGGER** ім'я\_трігера час\_ініціювання\_трігера

подія\_тригера **ON** ім'я \_таблиці

рівень\_тригера

**EXECUTE ROCEDURE** заголовок\_тригерної\_функції;

*Ім'я тригера*. Ім'я тригера має бути унікальним серед імен тригерів схеми БД.

*Час ініціювання тригера*. При визначенні тригера надається можливість вказати момент ініціювання його виконання, а саме - чи повинен тригер бути ініційований перед виконанням події (BEFORE) або після (AFTER).

*Подія тригера*. Як події тригера можуть виступати тільки зміни станів БД. Вони, природно, пов'язані з командами вставки INSERT, зміни UPDATE і видалення DELETE рядків таблиці.

*Ім'я таблиці*. Ім'я таблиці БД, причому тільки однієї.

*Рівень тригера*. Рівень тригера визначається однією з 2-х наступних фраз: FOR EACH ROW або FOR EACH STATEMENT. Тим самим можна вказати, скільки разів тригер буде виконуватися по відношенню до ініціювання його події. Можливі 2-ва варіанти:

По одному разу щодо кожного рядка, який буде підданий впливу події тригера (FOR EACH ROW);

Один раз щодо події незалежно від того, скільки рядків обробляє твердження (FOR EACH STATEMENT).

*Заголовок тригерної функції*. Вказується тригерна функція, яка буде виконуватися при ініціалізації тригера.

Під час роботи тригера доступні спеціальні змінні:

* OLD - запис перед оновленням або перед видаленням;
* NEW - запис, який буде додано або оновлено.

Більш наочно використання цих змінних наведено в Таблиця 2.1.

Таблиця 2.1 – Використання змінних OLD та NEW

|  |  |
| --- | --- |
| *Подія* | *Змінна* |
| UPDATE | OLD, NEW |
| INSERT | NEW |
| DELETE | OLD |

## Створення тригеру в PgAdmin III

Розглянемо процес створення тригера в pgAdmin III на конкретному прикладі.

*Приклад 1*: *При додаванні запису в таблицю Marks БД «Deanery» необхідно автоматично перераховувати середній бал студента (колонка SrBall в таблиці Students)*.

Спочатку необхідно написати тригерну функцію, яка буде підраховувати середній бал студента по таблиці Marks і оновлювати відповідний запис в таблиці Students. Для цього у вікні Браузер об'єктів для БД "Deanery" необхідно виділити розділ Тригерні функції і активізувати відповідне контекстне меню (Рисунок 2.1).

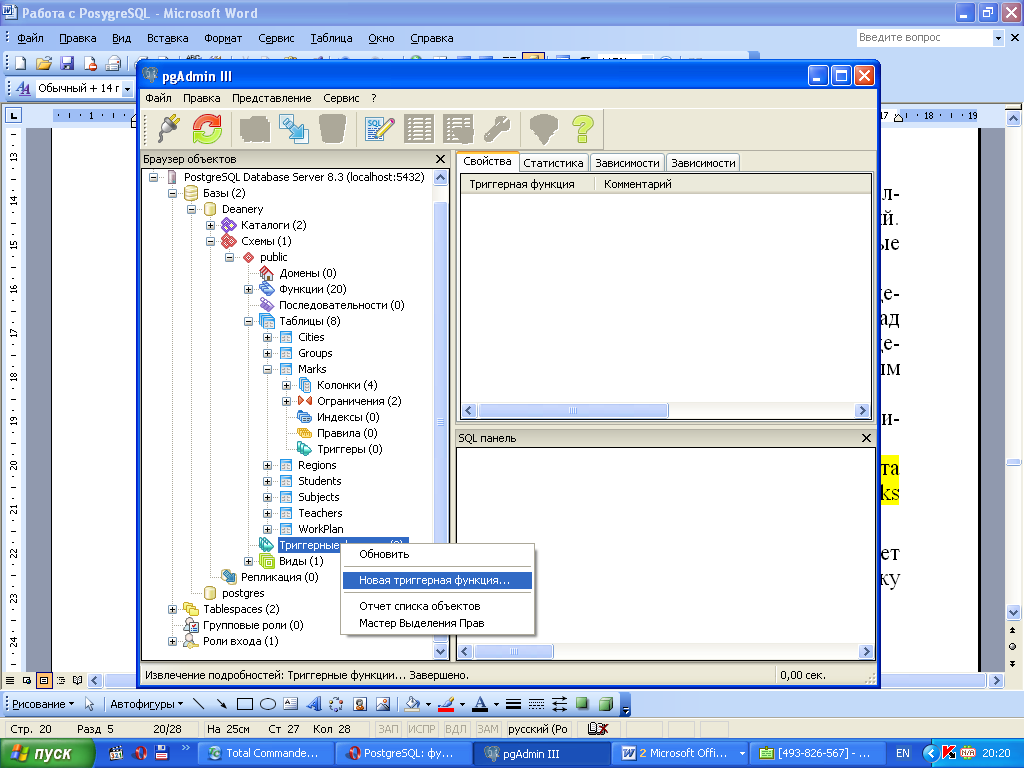


Рисунок 2.1 – Контекстне меню розділу Тригерні функції

В результаті відкриється вікно створення нової критичної функції (Рисунок 2.2).

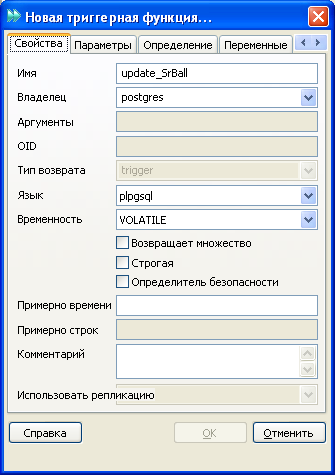


Рисунок 2.2 – Вікно Нова тригерна функція. Закладка Властивості

На закладці Властивості необхідно вказати наступні параметри:

1. Ім'я - ім'я тригерної функції, наприклад update\_SrBall, має бути унікальним серед імен тригерних функцій схеми БД;

2. Власник - postgres;

3. Мова - процедурний мову plpgsql;

4. Тимчасовість - VOLATILE [7] використовується за умовчанням і вказує на те, що значення функції може змінюватися в межах одного сканування таблиці;

На закладці Визначення вводиться "тіло" функції (Рисунок 2.3):

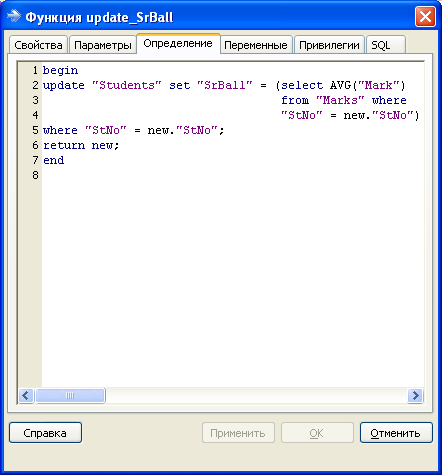


Рисунок 2.3 – Закладка Визначення вікна Нова тригерна функція

Повний код критичної функції можна переглянути на закладці SQL (Рисунок 2.4).

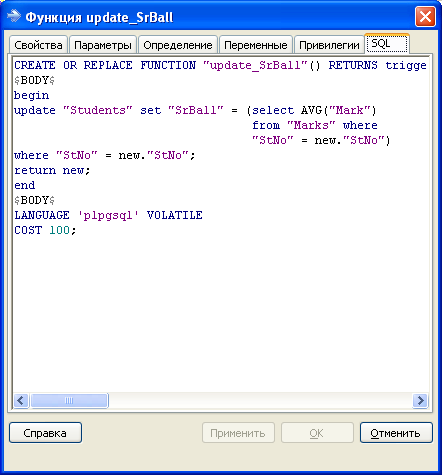


Рисунок 2.4 – Закладка SQL вікна Нова тригерна функція

Після цього необхідно створити власне тригер. Для цього у вікні Браузер об'єктів потрібно вибрати таблицю Marks, активізувати її контекстне меню і в ньому вибрати пункт Новий тригер (Рисунок 2.5).

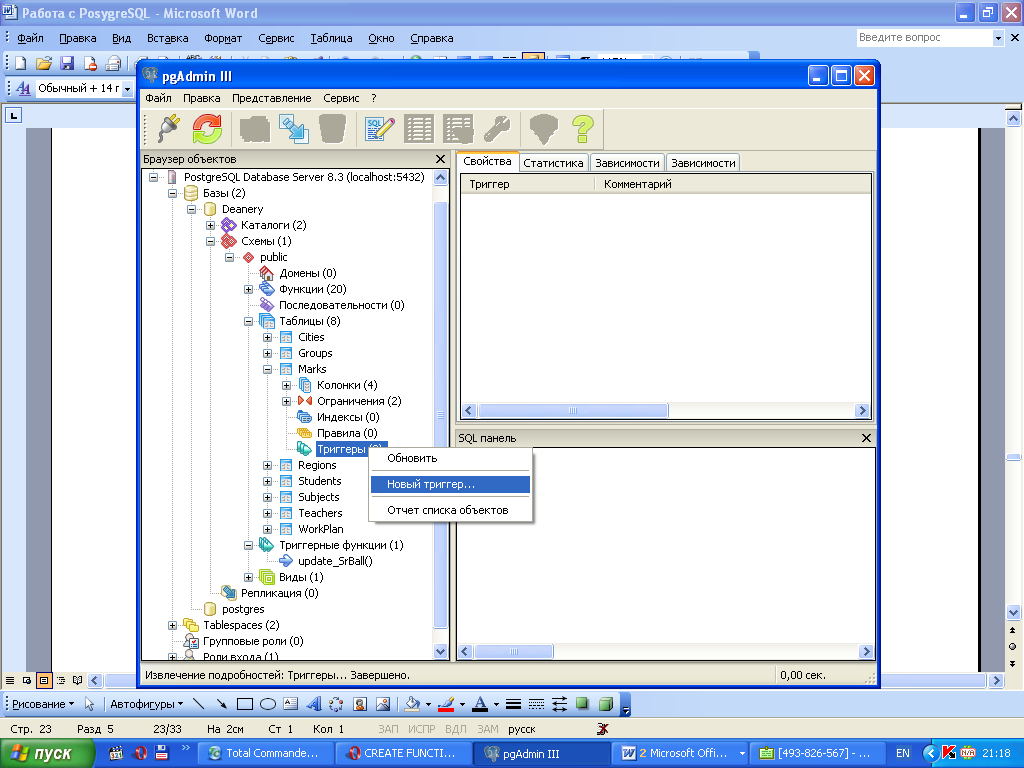


Рисунок 2.5 – Контекстне меню розділу Тригери

В результаті відкривається вікно створення нового тригеру (Рисунок 2.6).

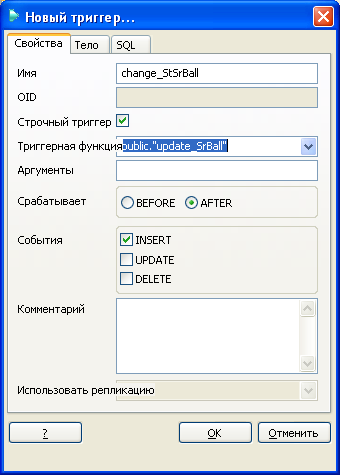


Рисунок 2.6 – Вікно Новий тригер. Закладка Властивості

У властивостях нового тригера необхідно вказати наступне:

* + Ім'я - ім'я нового тригера;
  + властивість Рядковий тригер задає рівень тригера: FOR EACH ROW (властивість вибрано) або FOR EACH STATEMENT (властивість не вибрано).
  + Тригерна функція - функція, яка буде виконуватися при спрацьовуванні тригера (вибирається зі списку);
  + властивість Спрацьовує вказує момент ініціювання тригера, а саме - чи повинен тригер бути ініційований перед виконанням події (BEFORE) або після (AFTER);
  + Події - вказується подія, за яким ініціюється тригер. Важливо підкреслити, що для одного тригера можна вибрати одне або кілька подій.

Перевірити роботу тригера можна при додаванні нового запису в таблицю Marks (Рисунок 2.7). Автоматично буде змінено значення колонки SrBall таблиці Students для заданого студента ( Рисунок 2.8).

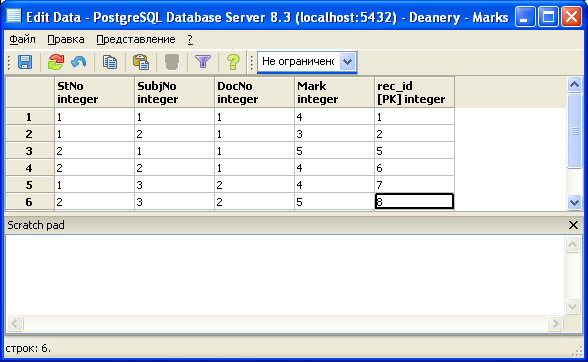


Рисунок 2.7 – Вміст таблиці Marks

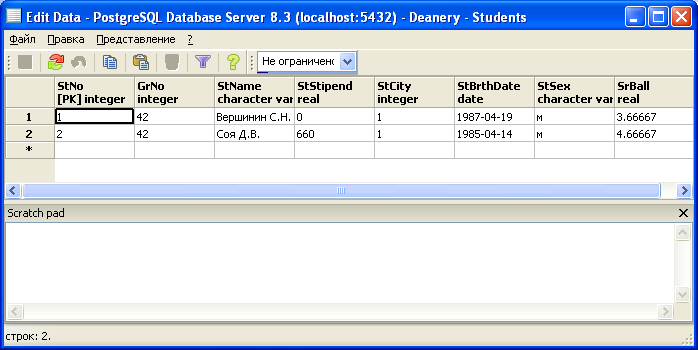


Рисунок 2.8 – Вміст таблиці Students