МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

В.І.Павловський, А.В.Петрашенко, Д.В.Победа

БАЗИ ДАНИХ ТА ЗАСОБИ УПРАВЛІННЯ

ІНСТРУКЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для здобувачів ступеня бакалавра за освітньо-професійною програмою підготовки бакалаврів спеціальності 123 – «Комп'ютерна інженерія»

Київ
КПІ ім. Ігоря Сікорського
2021

Рецензент *Заболотня Т.М.,* канд. техн. наук, доцент, доцент, КПІ ім. Ігоря Сікорського

Відповідальний редактор *Орлова М.М.*, канд. техн. наук, доцент, доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № від) за поданням Вченої ради факультету прикладної математики (протокол № від)

Електронне мережне навчальне видання

Павловський Володимир Ілліч, канд. техн. наук, доцент
Петрашенко Андрій Васильович, канд. техн. наук, доцент
Победа Дар'я Володимирівна, аспірант

БАЗИ ДАНИХ ТА ЗАСОБИ УПРАВЛІННЯ

ІНСТРУКЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Бази даних та засоби управління. Інструкції до виконання лабораторних робіт з дисципліни «Бази даних та засоби управління» [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – Комп'ютерна інженерія. / В.І. Павловський, А.В. Петрашенко, Д.В. Победа; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: X,XX Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 112 с.

Навчальний посібник розроблено для ознайомлення студентів із методикою використання засобів СУБД PostgreSQL при виконанні лабораторних робіт з дисципліни «Бази даних та засоби управління». Навчальний посібник містить інформацію щодо встановлення PostgreSQL та його ефективного використання, зокрема, створення та модифікації об'єктів бази даних, налаштування прав доступу користувачів, створення резервних копій та відновлення баз даних, а також управління рівнями ізоляції транзакцій.

Навчальний посібник призначений для студентів очної форми навчання за спеціальністю 123 – «Комп'ютерна інженерія» факультету прикладної математики КПІ ім. Ігоря Сікорського.

© В.І.Павловський, А.В.Петрашенко, Д.В.Победа
© КПІ ім. Ігоря Сікорського, 2021

ЗМІСТ

ВСТУП 5
1 ВСТАНОВЛЕННЯ СЕРВЕРУ БД PostgreSQL 6
1.1 Завантаження *6*
1.2 Розгортання *6*
2 СТВОРЕННЯ НОВОЇ БД 12
2.1 Створення нової БД *12*
2.2 Створення нової таблиці *14*
2.3 Додавання нових колонок (стовпців) таблиці *15*
2.4 Визначення первинного ключа таблиці *18*
2.5 Особливості первинних ключів таблиць в PostgreSQL *21*
2.6 Проблеми переносу БД в PostgreSQL з іншої СУБД *22*
2.7 Зовнішні ключі та створення зв'язків між таблицями *24*
2.8 Унікальність значень полів одного або групи стовпчиків *30*
3 ПЕРЕГЛЯД, ВВЕДЕННЯ ТА РЕДАГУВАННЯ ДАНИХ ТАБЛИЦІ 37
4 ВИКОНАННЯ SQL-ЗАПИТІВ КОРИСТУВАЧА 39
4.1 Вікно запитів *39*
4.2 Особливості формування запитів *39*
4.3 Виконання окремого запиту *40*
4.4 Виконання групи запитів *40*
5 ІНДЕКСУВАННЯ ТАБЛИЦЬ БАЗ ДАНИХ 42
5.1 Індекси *42*
5.2 Створення та видалення індексів *42*
5.2.1 Створення індексу 42
5.2.2 Видалення індексу 44
5.2.3 Поведінка та використання індексів 44
5.3 Типи індексів *45*
5.3.1 B-дерева 45
5.3.2 Хеш-індекси 45
5.3.3 Індекси GiST, SP-GiST і GiN 45
5.3.4 Індекси BRIN 47
5.4 Складені індекси *48*
6 ПОВНОТЕКСТОВИЙ ПОШУК 49
6.1 Повнотекстовий пошук *49*
6.2 Засоби повнотекстового пошуку PostgreSQL *49*
6.3 Пошук простої відповідності *51*
6.4 Конфігурування пошуку відповідності *51*
6.5 Пошук в полі таблиці *51*
6.6 Ранжирування результатів пошуку *52*
6.7 Виділення результатів пошуку *52*
6.8 Індекси і прискорення повнотекстового пошуку *53*
6.8.1 Створення та використання індексів текстових полів 53
6.8.2 Порівняння способів індексації GiST і GIN 55
7 ПРАВА ДОСТУПУ 57
7.1 Створення ролей *57*
7.2 Призначення прав доступу *58*

7.3 Відміна прав доступу *59*
7.4 Перевірка прав доступу *59*
8 ІЗОЛЯЦІЯ ТРАНЗАКЦІЙ 63
8.1 Проблеми неузгодженості даних при виконанні паралельних транзакцій *63*
8.2 Рівні ізоляції транзакцій *63*
8.3 Ізоляція транзакцій в PostgreSQL *64*
8.4 Ізоляція транзакцій в сучасних системах програмування *66*
9 СТВОРЕННЯ ТА ВИКОНАННЯ СЕРВЕРНИХ ПРОЦЕДУР 67
9.1 Формування тексту процедури у вікні SQL-запитів користувача *67*
9.2 Використання майстра створення серверної процедури *68*
9.3 Редагування серверної процедури *71*
9.4 Виклик серверних процедур в pgAdmin і прикладній програмі *73*
9.4.1 Виклик серверної процедури в pgAdmin 73
9.4.2 Виклик серверної процедури в прикладній програмі 74
9.4.3 Виклик серверних процедур в Java засобами JDBC 74
9.4.4 Виклик серверних процедур або функцій у Python 77
10 СТВОРЕННЯ ТА ВИКОНАНЯ ТРИГЕРІВ 79
10.1 Синтаксис визначення тригера в PostgreSQL *79*
10.2 Створення тригеру в PgAdmin III *80*
11 РЕЗЕРВНЕ КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ БД 85
11.1 BackUp - Резервне копіювання БД *85*
11.2 Restore - Відновлення БД *87*
11.3 Створення SQL-дампа БД *90*
11.4 Відновлення SQL-дампа засобами pgAdmin III *93*
11.5 Створення дампа БД за допомогою утиліт pg_dump та pg_dumpall *95*
11.6 Відновлення SQL-дампа БД, створеного утилітою pg_dump *97*
12 ОСОБЛИВОСТІ ВЗАЄМОДІЇ СУБД Access ТА PostgreSQL 100
12.1 Підготовка з'єднання PostgreSQL з БД в Access *100*
12.2 Копіювання таблиць Access в таблиці PostgreSQL *102*
12.3 Підключення таблиць PostgreSQL к БД в Access *105*
13 ХАРАКТЕРИСТИКИ PostgreSQL 109
13.1 Розмір бази даних *109*
13.2 Підтримувані вбудовані типи даних *109*
14 АДМІНІСТРУВАННЯ PostgreSQL 111
ЛІТЕРАТУРА 112

ВСТУП

Бази даних і, відповідно, системи управління базами даних (СУБД) займають центральне місце в сучасних інформаційних системах. Володіння відповідними знаннями та навичками є обов'язковим для спеціалістів IT-галузі.

На сьогодні на світовому ринку розповсюджена велика кількість різнорідних СУБД, які працюють з реляційними базами даних (БД). Серед них є комерційні та вільно розповсюджувані системи.

Комерційні СУБД такі як Oracle чи DB2 розробляються та підтримуються потужними IT-компаніями і орієнтовані на роботу з великими обсягами даних та великою кількістю користувачів. Відповідно вони мають високу вартість і застосовуються потужними компаніями та установами.

Вільно розповсюджувані системи такі як PostgreSQL чи MySQL розробляються співтовариствами розробників на добровільній основі і орієнтовані на роботу з відносно невеликими обсягами даних та невеликою кількістю користувачів. Відповідно вони застосовуються середніми та малими компаніями та установами і широко поширені в різних університетах світу.

В даному посібнику розглядаються всі основні питання розгортання та використання СУБД PostgreSQL. Вибір PostgreSQL визначений, зокрема, простотою та наочністю інтерфейсу його оболонки pgAdmin.

Існує значна кількість версій PostgreSQL, остання це PostgreSQL 13. На практиці програмні додатки створені до 2015 р. в основному використовують версії PostgreSQL 9.4 чи 9.5 з оболонкою pgAdmin III. Ця оболонка є окремим компактним додатком, який розробляється під конкретну операційну систему. Зокрема це дає можливість вставляти в текстові документи інформативні копії екранів.

Починаючи з версії PostgreSQL 9.6 застосовується оболонка pgAdmin IV. Вона для своєї роботи використовує єдиний веб інтерфейс. Нажаль копії екрану при цьому виглядають неінформативно.

Для забезпечення наочності відповідних дій в даному посібнику використовується ілюстрація роботи з PostgreSQL 9.5 за допомогою pgAdmin III.

1 ВСТАНОВЛЕННЯ СЕРВЕРУ БД PostgreSQL

1.1 Завантаження

Попередня умова: Наявність файлу установки PostgreSQL, наприклад postgresql-9.5-widows.exe або більш пізні.

Завантажити ці версії можна з великої кількості сайтів, наприклад:

http://www.postgresql.org/download/

http://postgresql.ru.net/download.html

http://www.enterprisedb.com/products/pgdownload.do#windows

http://wwwmaster.postgresql.org/download/

Необхідно звернути увагу, на відмінність у версіях PostgreSQL для 32-х і 64-х розрядних Windows. Отже, необхідно завантажувати відповідну версію файлу установки PostgreSQL.

Для доступу до PostgreSQL з програмних додатків Windows необхідно попередньо інсталювати відповідні пакети або модулі доступу до БД.

Для доступу до PostgreSQL з Python, слід використовувати останню версію модулю psycopg2. Всі потрібні дії можна виконати в відповідній оболонці Python.

Для доступу до PostgreSQL з java слід використовувати останні версії драйвер postgresql-xxx.jdbc. Завантажити їх можна з великої кількості сайтів, наприклад:

http://jdbc.postgresql.org/download.html

Для доступу до PostgreSQL для стандартних програмних додатків Windows можна використовувати odbc-драйвер psqlodbc. Його можна завантажити з великої кількості сайтів, наприклад:

http://www.postgresql.org/ftp/odbc/versions/msi/

http://postgresql.ru.net/download.html

1.2 Розгортання

Нижче показаний процес установки postgresql-9.0.4. За своїм змістом він подібний до процесу для більш ранніх версій, але більш економічний.

Спочатку необхідно запустити файл postgresql-9.0.4.msi. В результаті з'явиться вікно, представлене на рисунку 1.1.

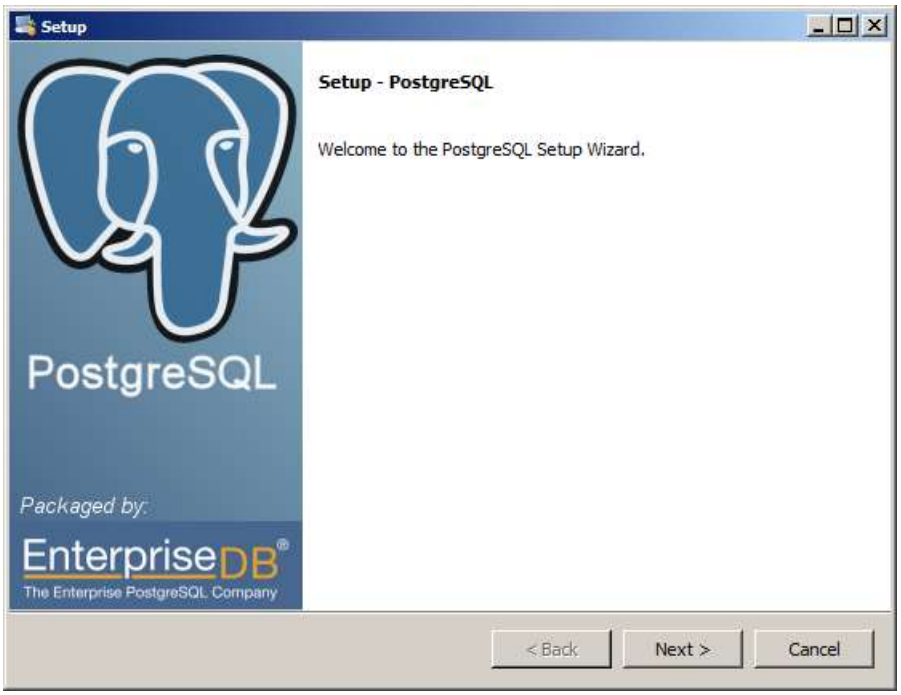


Рисунок 1.1 – Початкове вікно установки PostgreSQL

Вибираємо директорію установки сервера PostgreSQL.

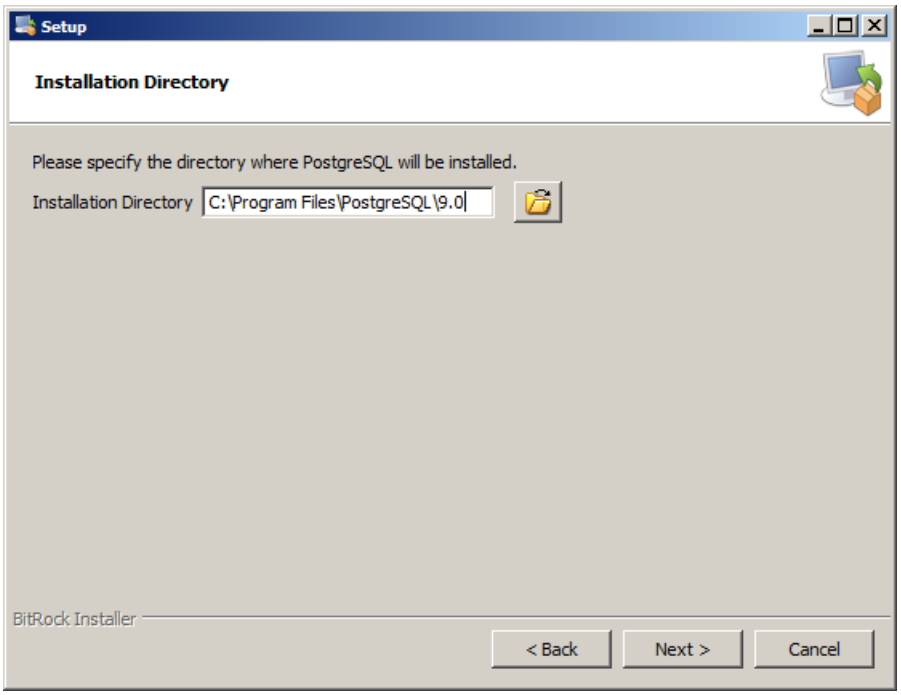


Рисунок 1.2 – Директорія установки PostgreSQL

Вибираємо директорію установки власне БД. За замовчуванням пропонується диск C, але бажано вибрати інший диск, наприклад диск D, оскільки в процесі експлуатації може знадобитися переформатувати диск C.

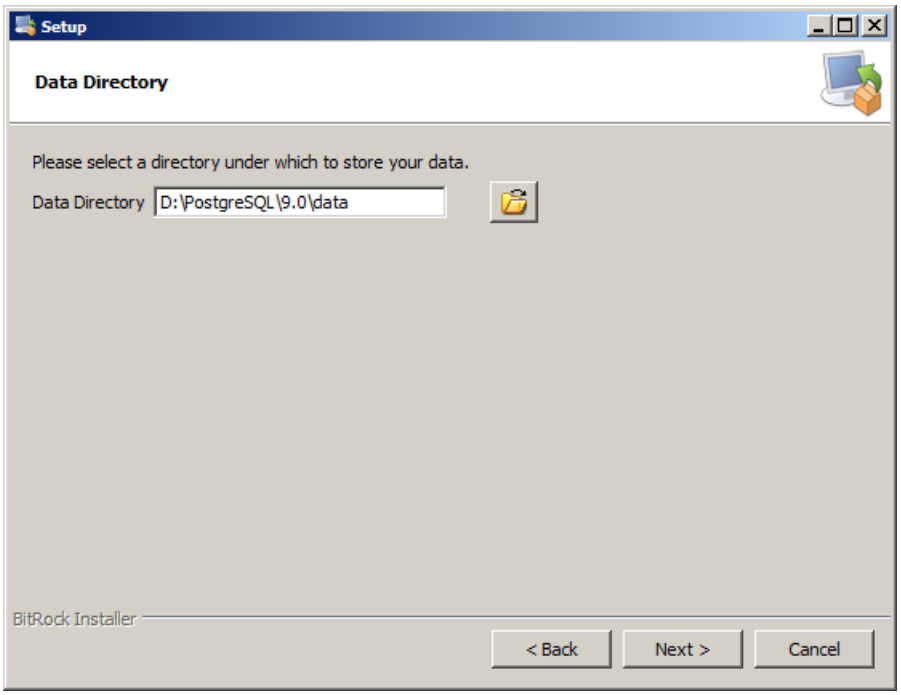


Рисунок 1.3 – Директорія установки БД на диску D

Потім задаємо пароль адміністратора. У навчальних цілях рекомендується пароль qwerty.

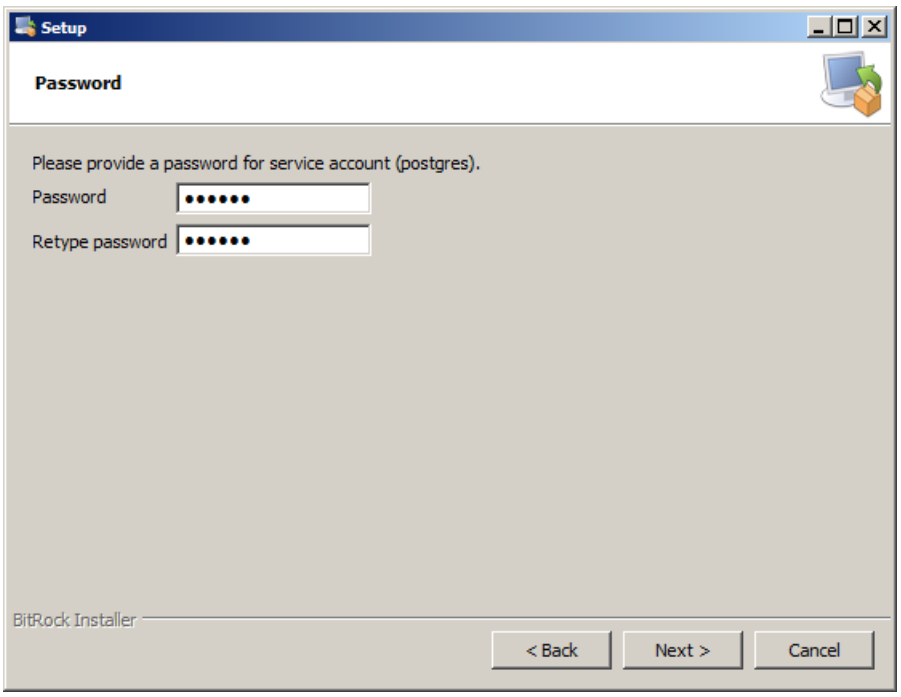


Рисунок 1.4 – Задання паролю адміністратора

Встановлюємо (за замовчуванням) порт 5432, за яким сервер слухає звернення. Якщо на комп'ютері розміщується декілька версій PostgreSQL, то, відповідно, можна вибрати порти 5434, 5436 тощо.

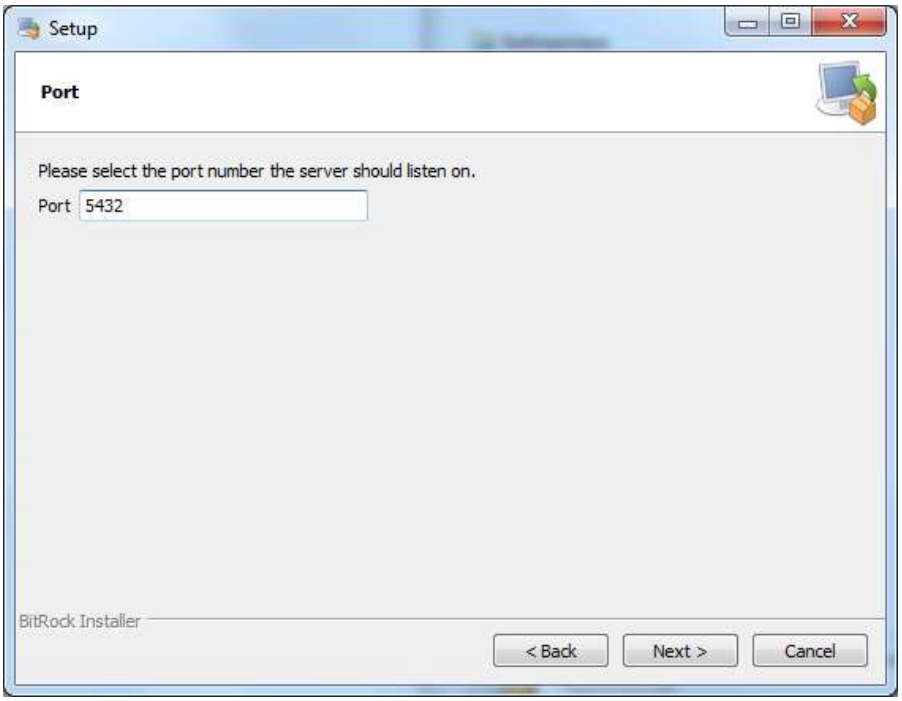


Рисунок 1.5 – Вибір порту

Встановлюємо локалізацію для виведення повідомлень, наприклад, українську. Можна вибрати локалізацію за замовчуванням, тоді буде обрана локалізація ОС.

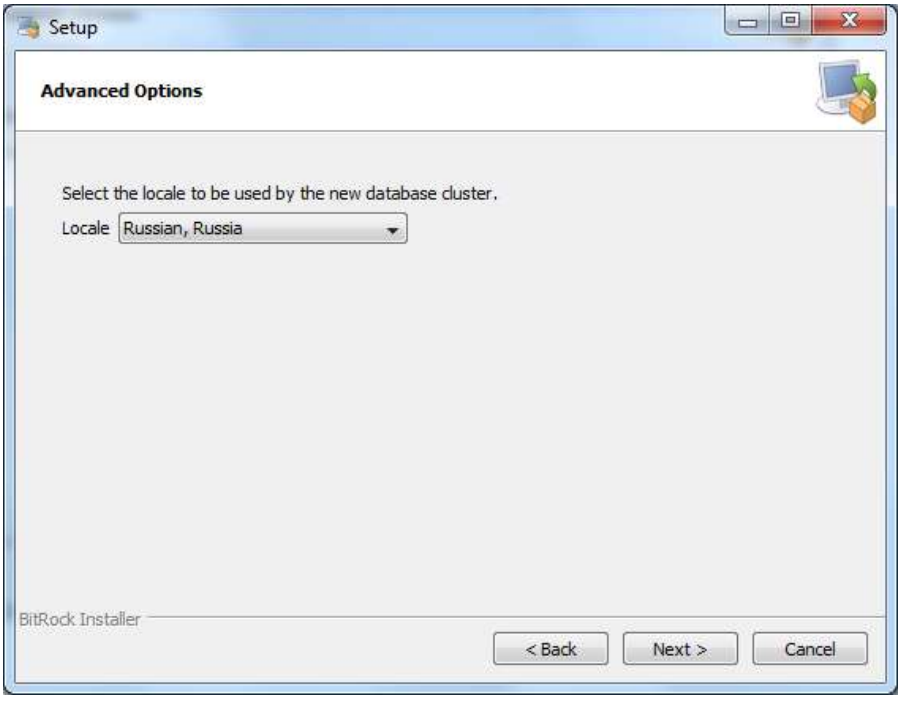


Рисунок 1.6 – Вибір локалізації

Після чого переходимо до процесу інсталяції і завершення.

Рисунок 1.7 – Установка PostgreSQL завершена

Примітка. Stack Builder використовується для завантаження та інсталяції додаткових додатків. Для цього курсу в цьому немає необхідності.

Після завершення установки і при включенні комп'ютера сервер PostgreSQL автоматично активується. Доступ до нього можна отримати через сервіс PgAdmin III, ярлик якого встановлюється на робочому столі автоматично.

Деактивувати сервіс PostgreSQL можна засобами адміністратора системи.

Одночасно з СУБД PostgreSQL буде встановлений менеджер БД pgAdmin III - спеціальний програмний додаток, призначений для адміністрування БД.

Зазвичай ярлик pgAdmin III відразу встановлюється на робочому столі. Якщо цього не відбудеться, то запустити його можна по команді Пуск -> Всі програми -> pgAdmin III -> pgAdmin (Рисунок 1.8).

Увага! pgAdmin III поставляється і працює з версіями PostgreSQL до 9.5 включно. Це найбільш популярний продукт серед розробників БД в PostgreSQL. Але починаючи з версії PostgreSQL 9.6 і далі, поставляється pgAdmin IV. Він для роботи з PostgreSQL використовує web-браузер. Змістовно та структурно інформація, надана pgAdmin IV, нічим не відрізняється від інформації, наданої pgAdmin III, але вона розмивається по всій web-сторінці, що робить її копію в текстовому документі вкрай невиразною.

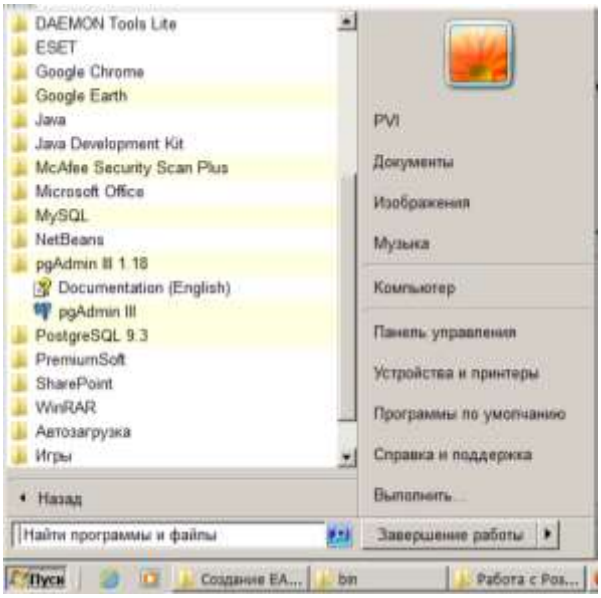


Рисунок 1.8 – Запуск pgAdmin III

Увага! pgAdmin III поставляється розробниками PostgreSQL і тому вміє звертатися до PostgreSQL безпосередньо, минаючи ODBC. Тоді як створювані користувачами Windows-додатки повинні використовувати ODBC або спеціальні бібліотеки, призначені для окремих мов програмування Java, Python тощо.

2 СТВОРЕННЯ НОВОЇ БД

Запустіть менеджер БД pgAdmin III з паролем адміністратора, наприклад qwerty. Ім'я користувача postgres автоматично створювалося при розгортанні PostgreSQL. Далі виконайте з'єднання з сервером БД. Можливі 2 варіанти:

- ✓ за допомогою меню "Інструменти (Сервіс)" (Рисунок 2.1);
- ✓ подвійним натисканням кнопки миші на імені сервера у вікні "Браузер об'єктів".

І в одному, і в іншому випадку з'явиться вікно введення пароля (Рисунок 2.2).

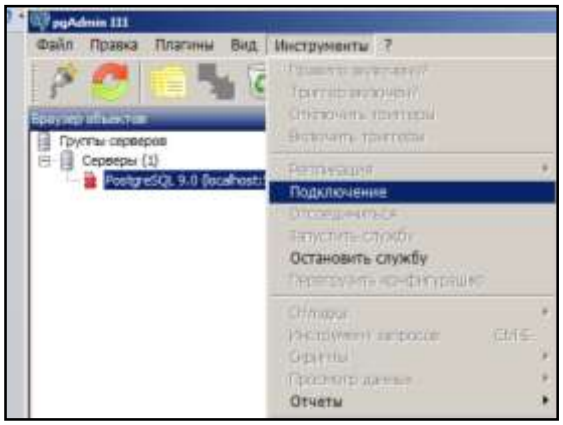


Рисунок 2.1 – Меню Сервіс

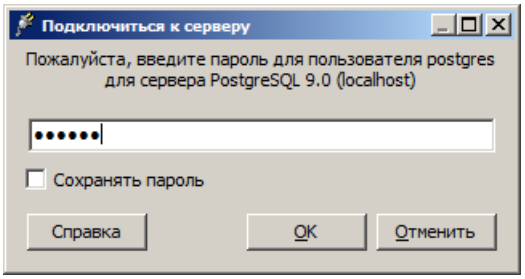


Рисунок 2.2 – Вікно введення паролю

2.1 Створення нової БД

Щоб створити нову БД необхідно у вікні "Браузер об'єктів" вибрати розділ "Бази", викликати його контекстне меню і в ньому вибрати пункт "Нова база даних" (Рисунок 2.3).

Увага! Імена БД, таблиць, стовпців тощо найкраще записувати за допомогою символів латинського алфавіту, цифр та символу "_". Починатися імена повинні тільки з символів латинського алфавіту.

Крім того, PostgreSQL чутливий до регістру символів, тому практикуючі програмісти воліють в усіх назвах використовувати тільки рядкові символи. В іншому випадку в SQL-запитах до БД відповідні імена необхідно укладати в пару подвійних лапок "....". У деяких випадках, коли використовуються компоненти, що автоматично створюють SQL-запити, це не вдасться зробити і проект не працює. Важливо також те, що такий підхід призводить до більшої сумісності програм, що розробляються, для роботи з різними СУБД.

Примітка. На жаль, використання в назвах таблиць тільки малих символів призводить до їх малої читабельності і створює додаткові труднощі особливо для початківців. Тому для більшої наочності в цьому посібнику імена БД, таблиць і стовпців починаються і містять великі (прописні) символи. Тому в SQL-запитах до БД відповідні імена необхідно укладати в пару подвійних лапок "....".

Рисунок 2.3 – Контекстне меню розділу Бази

В результаті виконання цих дій відкриється вікно введення інформації про нову БД (Рисунок 2.4)

Для подальших прикладів буде використана демонстраційна БД "Deanery".

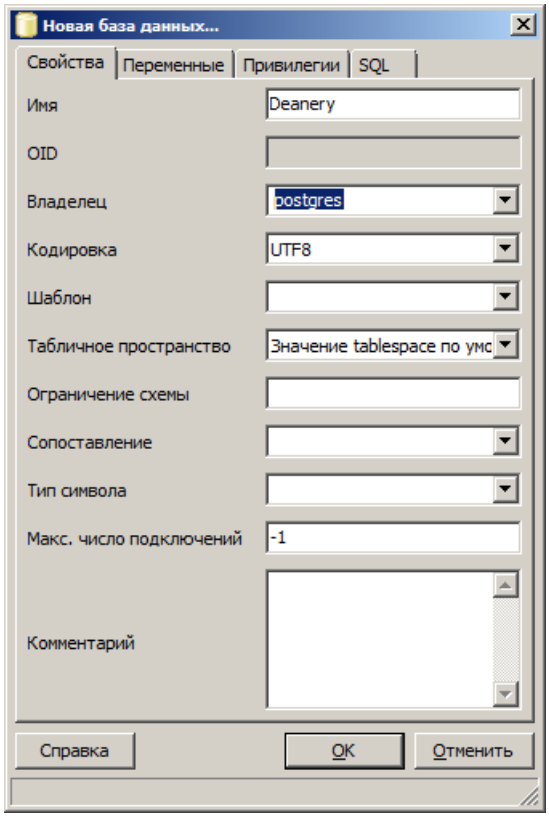


Рисунок 2.4 – Вікно Нова база даних

У вікні на закладці "Властивості" необхідно вказати наступні параметри:

✓ Ім'я - ім'я нової БД (обов'язковий параметр). Він повинен складатися з букв і цифр (спочатку завжди повинна бути буква) і бути не більше 63 символів довжиною. У нашому випадку це Deanery;

✓ Власник - користувач, який має всі права для роботи з БД. Зокрема він може створювати і видаляти БД [1]. За замовчуванням ім'я користувача postgres;

✓ Кодування - формат кодування символів в БД, зокрема, кирилиці - UTF8.

Після чого натиснути кнопку ОК.

В результаті виконання цих дій вікно "Браузер об'єктів" прийме наступний вигляд (Рисунок 2.5):

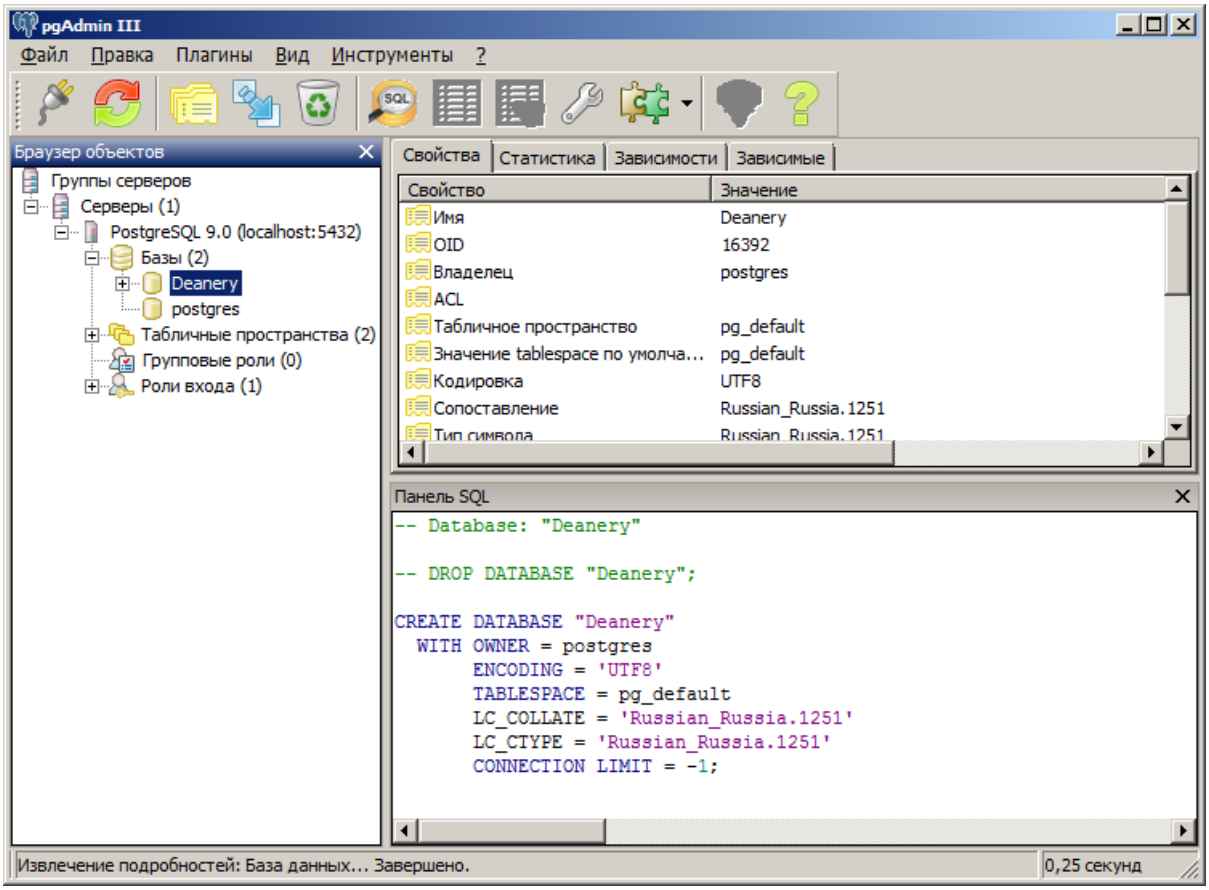


Рисунок 2.5 – Вікно Браузер об'єктів

2.2 Створення нової таблиці

Для створення нової таблиці в вікні "Браузер об'єктів" виберіть необхідну БД, активізуйте контекстне меню, пов'язане з таблицями, і виберіть пункт "Нова таблиця" (Рисунок 2.6).

В результаті з'явиться вікно введення інформації про нову таблицю (Рисунок 2.7).

Аналогічно створенню нової БД, для нової таблиці необхідно вказати ім'я, наприклад Students, і власника (postgres).

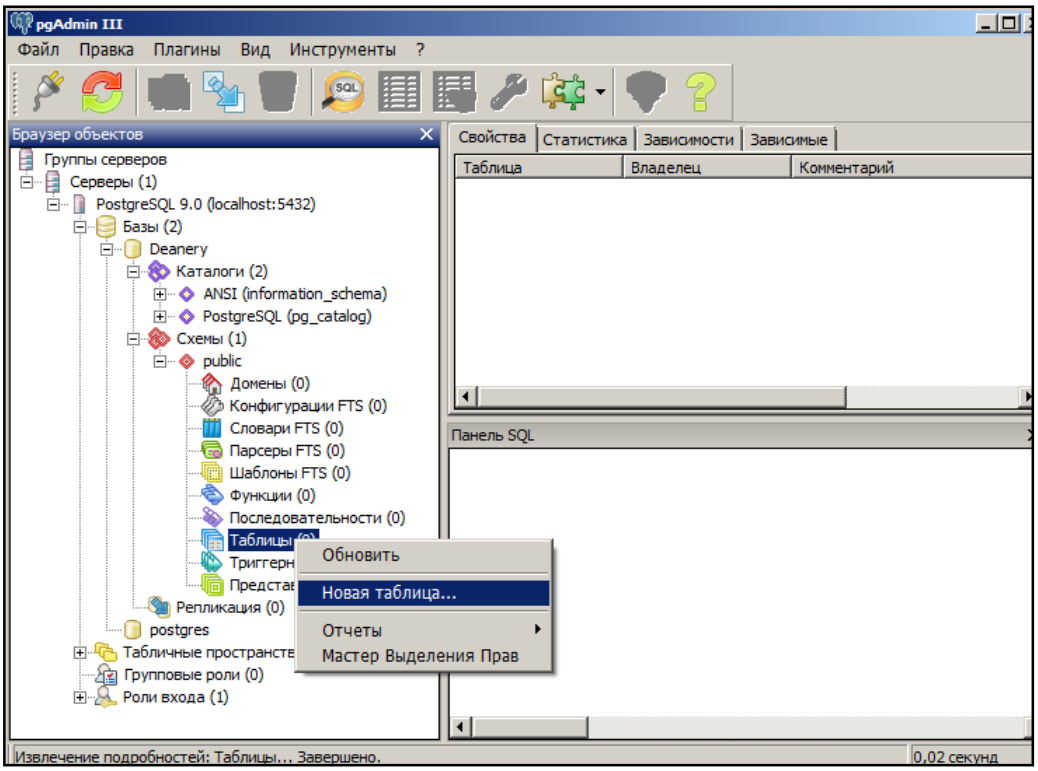


Рисунок 2.6 – Контекстне меню розділу Таблиці

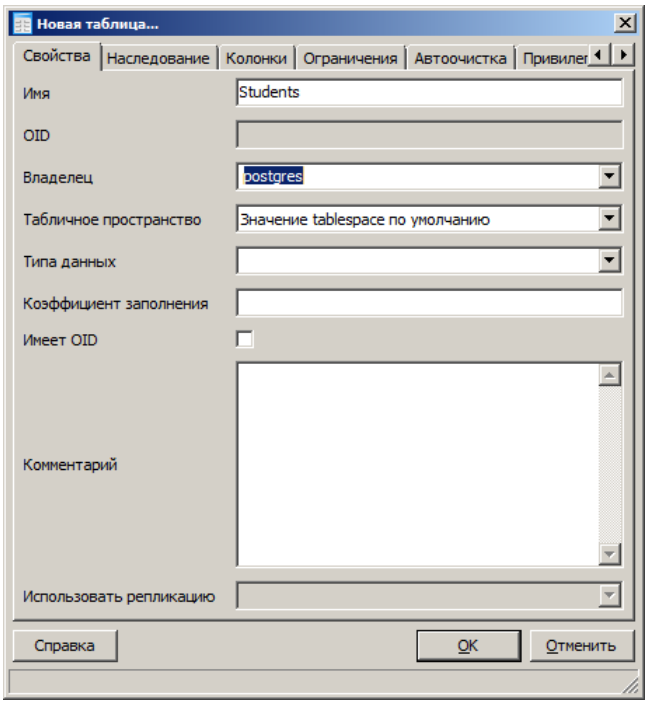


Рисунок 2.7 – Закладка Властивості вікна Нова таблиця

2.3 Додавання нових колонок (стовпців) таблиці

Після створення таблиці необхідно перейти до додавання її колонок (стовпців). Для цього на формі (Рисунок 2.7) слід вибрати закладку Колонки (Рисунок 2.8).

Примітка. Нові стовпці можна додавати до вже існуючої таблиці через форму Властивості таблиці, або до переліку її колонок.

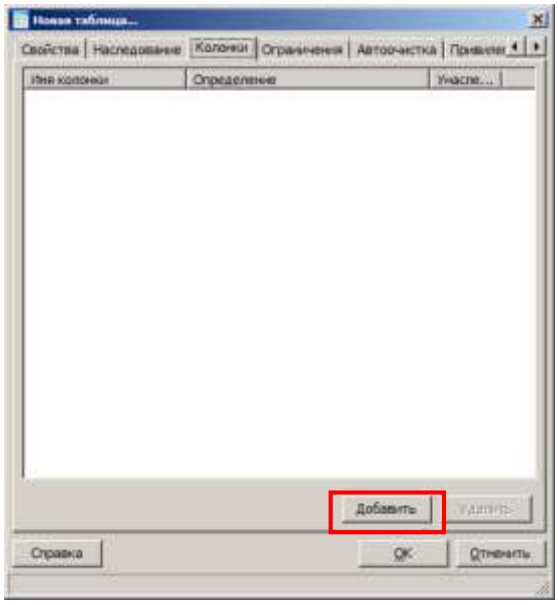


Рисунок 2.8 – Закладка Колонки

Натисніть кнопку Додати. З'явиться вікно "Нова колонка" (Рисунок 2.9). Задаємо в ньому ім'я і тип даних нової колонки.

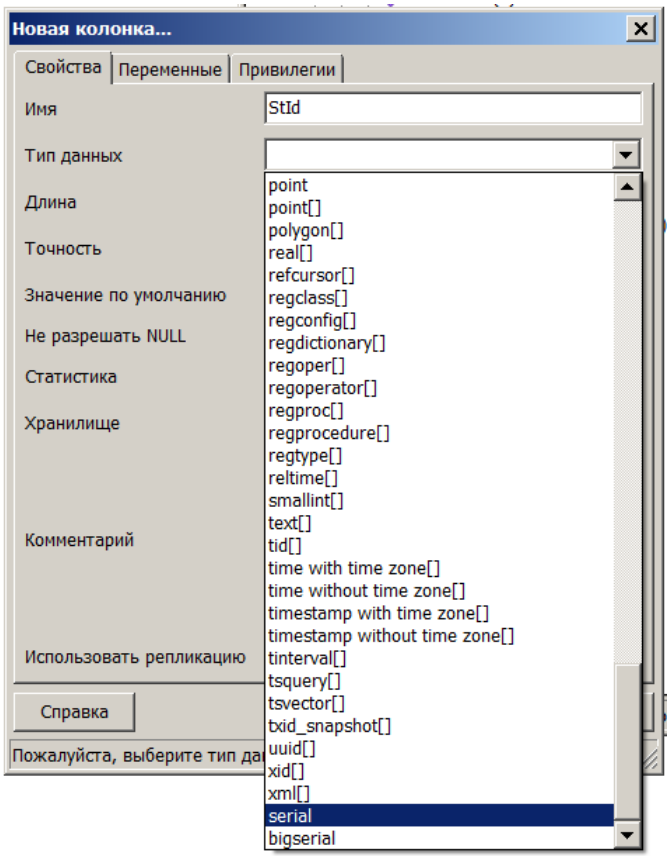


Рисунок 2.9 - Вікно Нова колонка. Типи даних

У властивостях нової колонки (Рисунок 2.10) слід вказати:

- ✓ Ім'я колонки;
- ✓ Тип даних;
- ✓ Значення за замовчуванням, якщо це необхідно;

- ✓ допустимо чи ні NULL-значення в колонці.

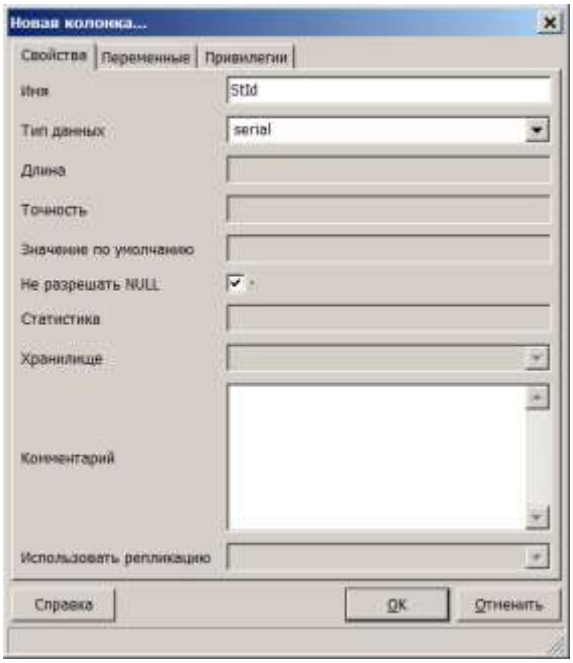


Рисунок 2.10 – Вікно Нова колонка. Властивості поля

Примітка. Слід звернути увагу, що для колонки StId – первинний ключ (див. нижче), в таблиці Students був обраний квазітип serial (Рисунок 2.10). Типи даних serial і bigserial не є повноцінними типами, а служать зручною нотацією для створення колонок унікальних цілочисельних ідентифікаторів (ID) - ключів, з послідовно зростаючими значеннями. Така властивість, подібно з AUTO_INCREMENT, підтримується багатьма СУБД. Вибір квазітипа serial автоматично призводить до того, що створюється послідовність (Sequence) Students_StId_seq (Рисунок 2.11), яка і задає механізм створення послідовно зростаючих значень ідентифікаторів. В результаті при додаванні нового запису значення поля StId починається з 1-ці та буде автоматично збільшуватися на відповідну величину – 1-цю (за замовчуванням). У всьому іншому це поле веде себе як поле типу integer.

У всіх інших випадках застосування цілочисельних даних слід вибирати один з варіантів типу integer.

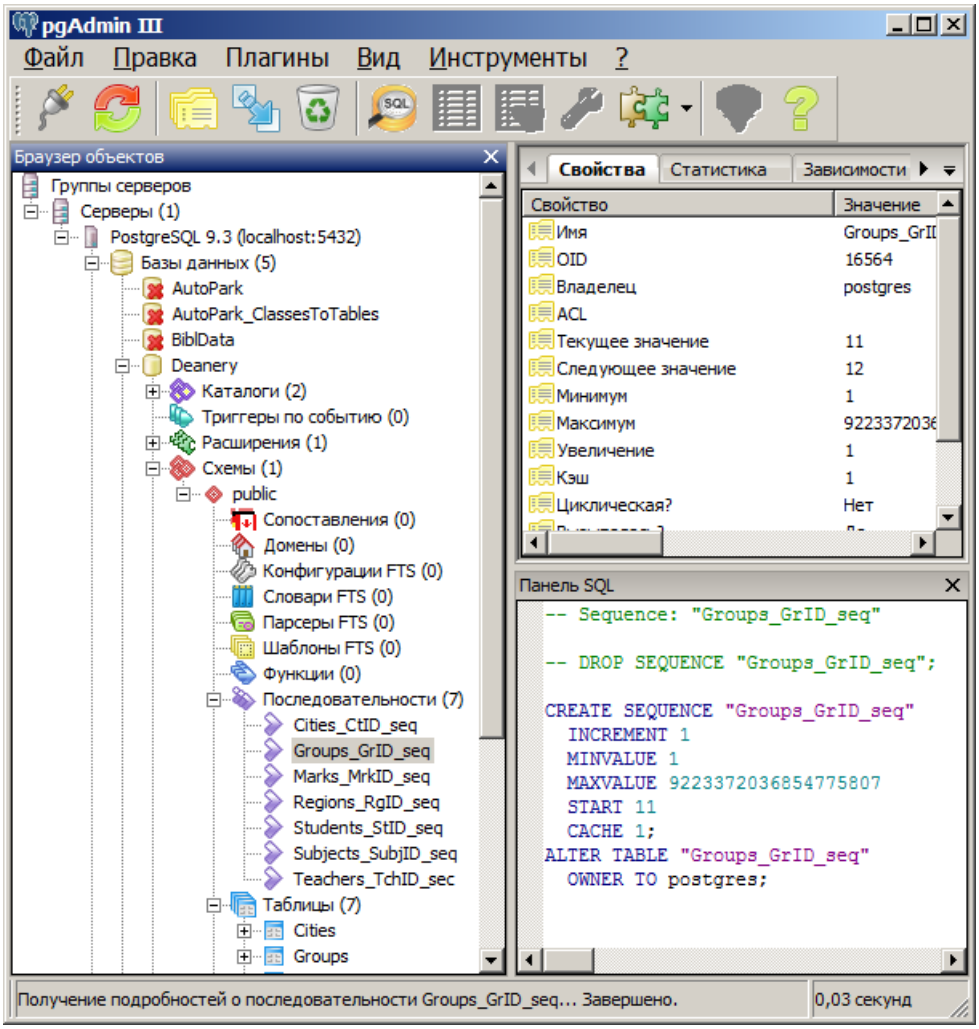


Рисунок 2.11 – Збережені послідовності

Таким же чином створюються всі інші колонки таблиці Students з вибором відповідних типів даних для кожної колонки.

2.4 Визначення первинного ключа таблиці

Кожна таблиця БД в PostgreSQL повинна мати первинний ключ, який однозначно ідентифікує відповідний рядок таблиці. В іншому випадку вона буде недоступна для редагування.

Порада: Якщо логіка таблиці не передбачає наявність в ній первинного ключа (наприклад, таблиця Marks БД "Deanery"), то його необхідно ввести примусово, попередньо створивши додаткову колонку ID і визначивши тип її елементів як serial.

Для визначення первинного ключа (Primary Key) таблиці необхідно вибрати закладку Обмеження вікна Нова таблиця (Рисунок 2.12)

Рисунок 2.12 – Закладка Обмеження вікна Нова таблиця

Примітка. Як видно з рисунка, крім первинного ключа для таблиці можна вказати зовнішній ключ (Foreign Key), задати унікальність (Unique) значень в колонках таблиці, ввести обмеження цілісності - перевірка (Check).

Щоб створити первинний ключ потрібно на вкладці Обмеження вибрати тип обмеження Первинний ключ і натиснути кнопку Додати. В результаті відкриється вікно Новий первинний ключ (Рисунок 2.13).

Для таблиці Students первинним ключем є колонка StId - штучний (сурогатний) ключ, який прийнято називати ID-записи.

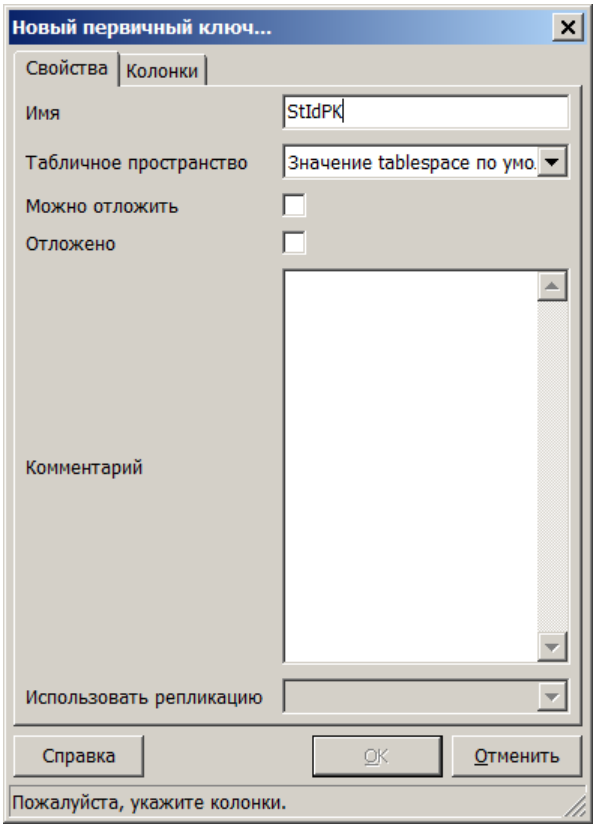


Рисунок 2.13 – Вікно Новий первинний ключ

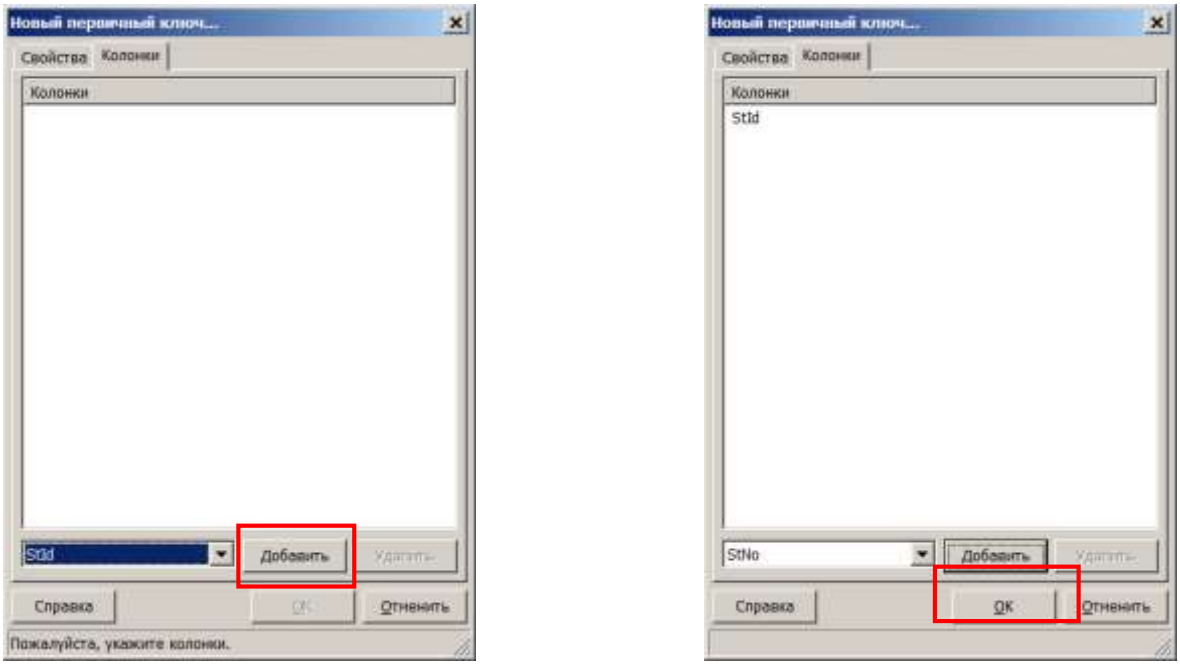
Особливістю СУБД PostgreSQL є те, що вона розрізняє первинний ключ таблиці, в нашому випадку це StID, і свій опис цього ключа. Тому тут на закладці Властивості слід вказати ім'я первинного ключа і табличний простір.

Ім'я первинного ключа для зручності може складатися з імені колонки і символів PK (від Primary Key). Це ім'я, наприклад StIdPK, використовується при проектуванні БД в PgAdmin III.

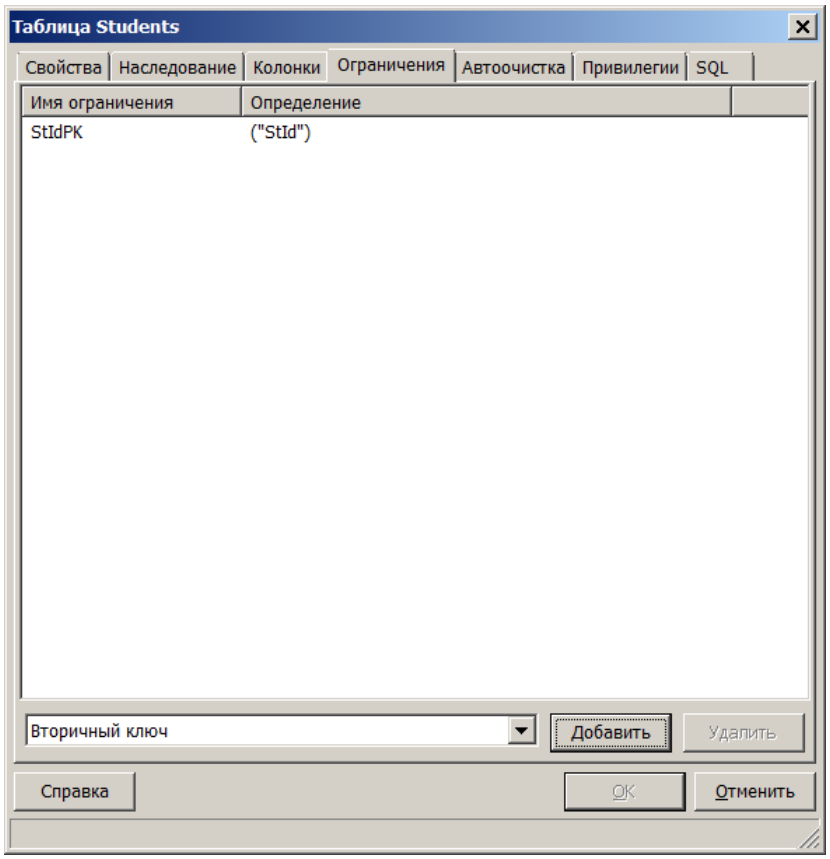
Табличний простір вибираємо за замовчуванням.

Після цього переходимо на закладку Колонки цього ж вікна (Рисунок 2.14).

На закладці Колонки зі списку потрібно вибрати необхідну колонку StId, і натиснути кнопку Додати. По завершенню слід натиснути кнопку ОК.



а) – Закладка Колонки вікна Новий первинний ключ



б) Закладка Обмеження з первинним ключем

Рисунок 2.14 – Закладки вікна Таблиця
при створенні первинного ключа

Примітка. При написанні SQL-запитів до БД використовується ім'я колонки StId, а не ім'я первинного ключа StIdPK.

SQL-запит на створення таблиці Student можна подивитися на закладці SQL вікна Таблиця (Рисунок 2.15).

```
CREATE TABLE "Students"
(
  "StNo" integer NOT NULL,
  "GrNo" integer NOT NULL,
  "StName" character varying(20),
  "StSex" character varying(1),
  "StBrthDate" date,
  "StCity" integer,
  "StStipend" real,
  "StId" serial NOT NULL,
  CONSTRAINT "StIdPK" PRIMARY KEY ("StId")
)
WITH (
  OIDS=FALSE
);
ALTER TABLE "Students" OWNER TO postgres;
```

Рисунок 2.15 – Текст запиту на закладці SQL вікна Нова таблиця

Аналогічним чином створюються первинні ключі всіх інших таблиць БД.

2.5 Особливості первинних ключів таблиць в PostgreSQL

Відразу слід підкреслити, що незалежно від ситуації хорошою практикою є використання в якості ключа додаткового поля ID типу serial.

Робота з первинними ключами в PostgreSQL має ряд особливостей. Ми розглянемо деякі з них.

Слід підкреслити, що в PostgreSQL сурогатний первинний ключ може бути представлений тільки в вигляді одного поля. Такий ключ повинен бути локалізовано за допомогою додаткового поля ID квазітипу serial, а поля, що утворюють складовою ключ, повинні не допускати значення NULL. Це скоротить витрати при роботі з декількома таблицями, що з'єднуються. (див. нижче)

Якщо необхідно, щоб ключ був представлений реальним полем, наприклад, Номер залікової книжки студента, або групою полів таблиці, наприклад Прізвище та Ім'я, то скрізь, де це можливо, тип поля слід вибирати рядковим, хоч він і виглядає як ціле число. Якщо ж ключ представити з типом поля integer, то при спробі введення з прикладної програми нового запису виникнуть конфлікти з PostgreSQL. Однак практика показала, що навіть коли в якості первинного ключа таблиці виступають рядкові дані, наприклад Прізвище, краще використовувати штучний первинний ключ - ID, створивши додаткову колонку і визначивши тип її елементів як serial.

2.6 Проблеми переносу БД в PostgreSQL з іншої СУБД

Слід підкреслити, що при перенесенні БД з деякої СУБД, наприклад Access, в СУБД PostgreSQL тип даних полів, аналогічних AUTO_INCREMENT, перетворюється в тип integer, що призводить до проблем при додаванні нових даних. Якщо перепроектування БД з введенням нових полів типу serial не є доцільним, що зазвичай має місце, то слід виконати заміну типу таких полів з integer на serial.

Однак пряма заміна типу поля integer на тип serial у властивостях поля в PgAdmin III недоступна.

Для перекладу в PgAdmin III типу поля integer в serial або bigserial слід спочатку для необхідного ключового поля таблиці створити послідовність (Sequence), в якій задати закон збільшення значень послідовності. Ім'я послідовності формується, як правило, за схемою "таблиця_колонка_seq". Для таблиці Groups отримаємо Groups_GrNo_seq.

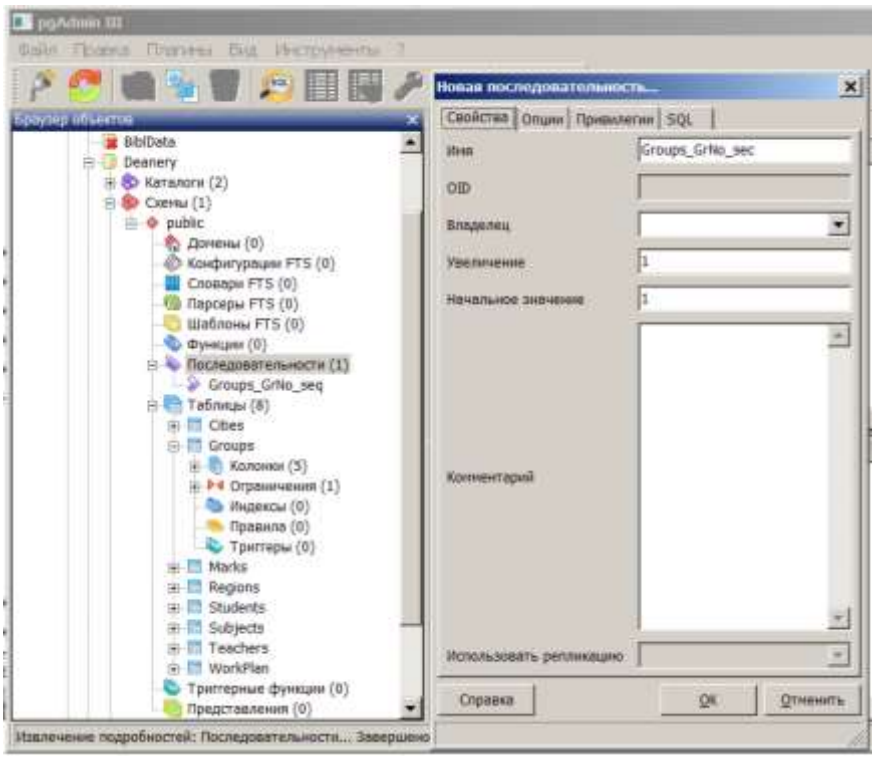


Рисунок 2.16 – Налаштування нової послідовності

Ця послідовність зберігається.

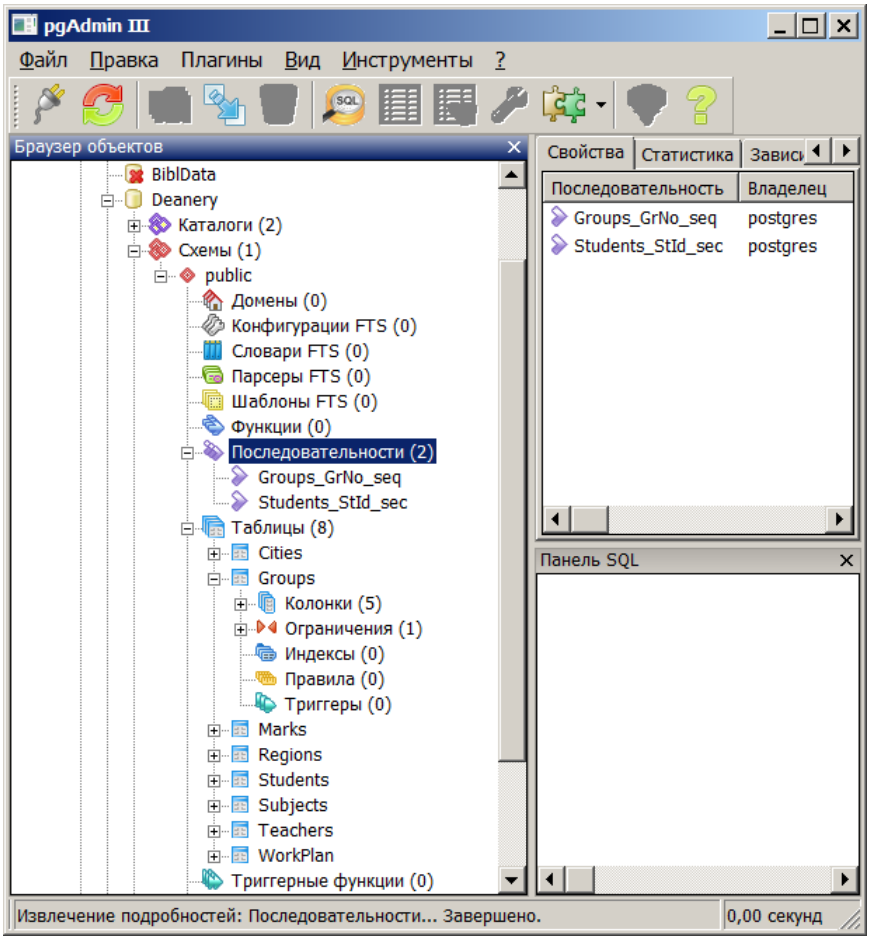


Рисунок 2.17 – Збережені послідовності

Далі формуємо SQL-запит на зміну типу

ALTER TABLE "Groups"
ALTER COLUMN "GrNo" **SET DEFAULT** nextval('"Groups_GrNo_sec"')
и виконуємо його (Рисунок 2.18)

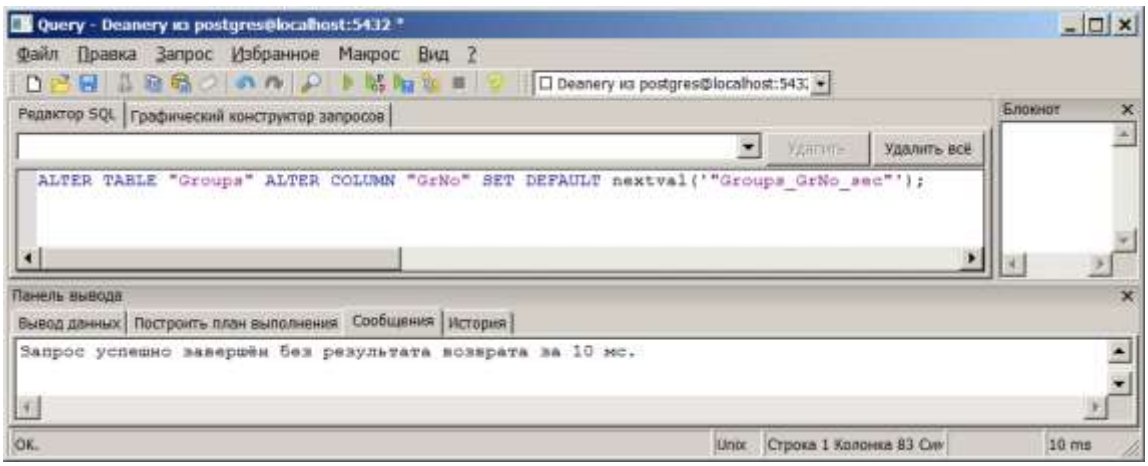


Рисунок 2.18 – Збережені послідовності

Тепер у властивостях колонки можна побачити, що задано значення за замовчуванням (Рисунок 2.19)

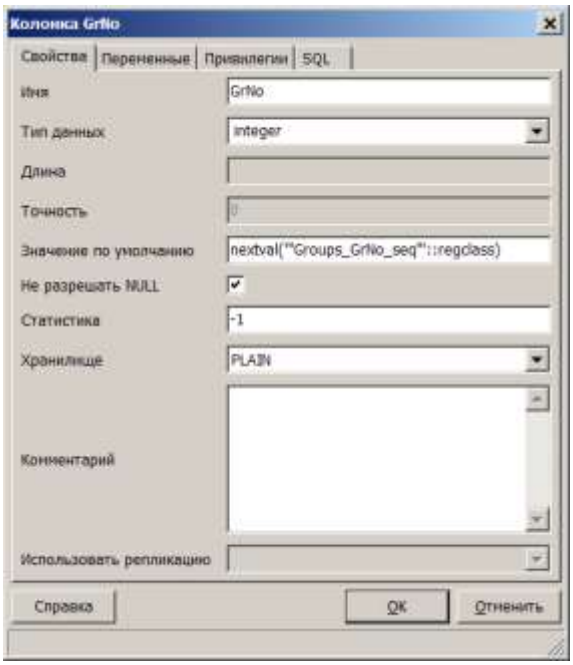


Рисунок 2.19 – В полі Значення за замовчанням задана потрібна послідовність

2.7 Зовнішні ключі та створення зв'язків між таблицями

Після того, як всі таблиці БД створені, необхідно пов'язати їх між собою. Для встановлення зв'язків між таблицями використовуються первинні і зовнішні ключі.

Для БД "Deanery" зв'язок між таблицями Students і Groups задається за допомогою зіставлення первинного ключа таблиці Groups - провідна, і зовнішнього ключа таблиці Students - ведена. Первинний ключ таблиці Groups до цього часу вже повинен бути створений.

Обмеження зовнішнього ключа каже, що значення в колонці (або групі колонок) веденої таблиці, має повністю збігатися зі значеннями, які існують в деяких рядках провідною таблиці. Для прикладу: у таблиці Students значення в колонці код групи GrNo повинні збігатися з відповідними значеннями в колонці код групи GrNo таблиці Groups.

Щоб створити зовнішній ключ в таблиці потрібно вибрати у вікні Браузер об'єктів необхідну таблицю, наприклад Students, активізувати її контекстне меню і в ньому вибрати пункт Властивості (Рисунок 2.20)

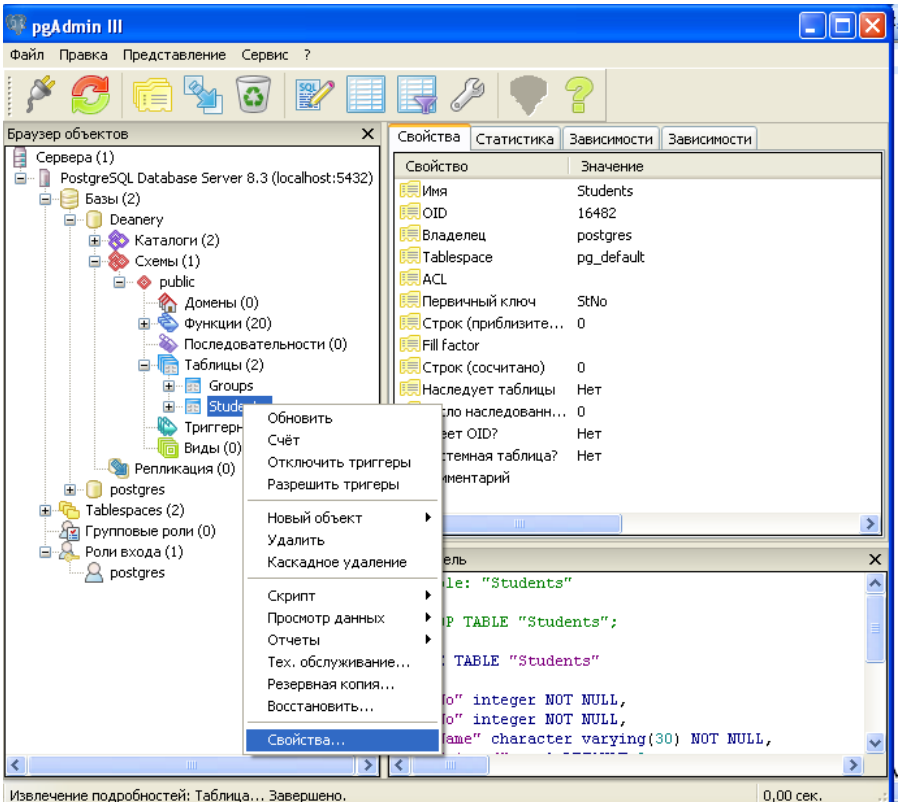


Рисунок 2.20 – Контекстне меню об'єкта Таблиця

Після цього відкриється вікно характеристик обраної таблиці, в якій слід вибрати закладку Обмеження (Рисунок 2.21).

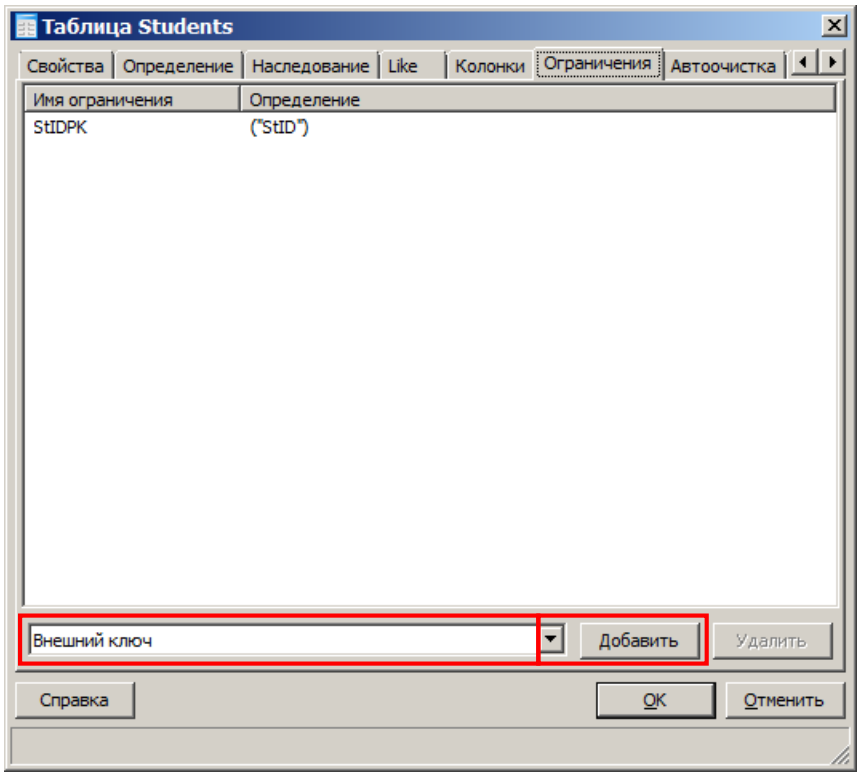
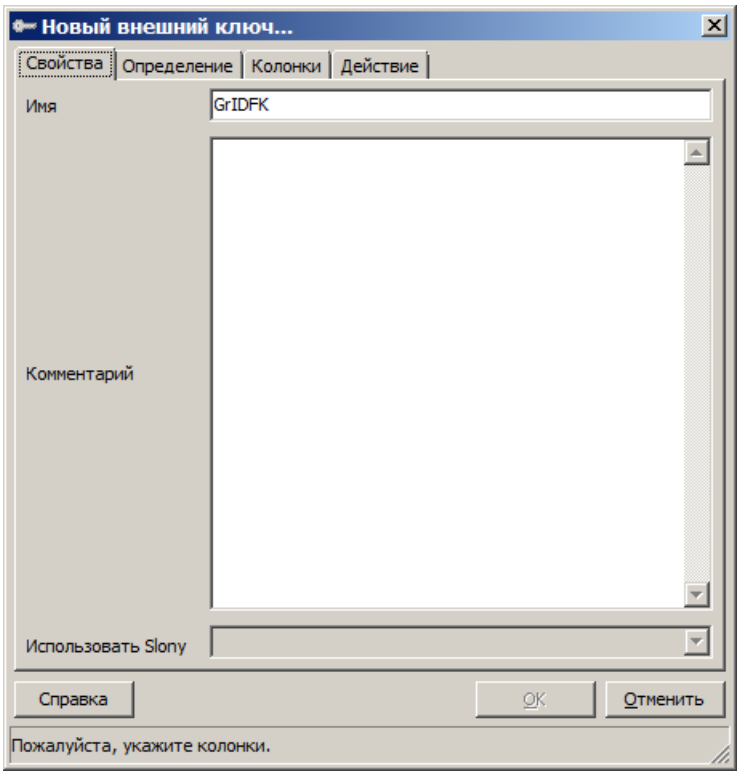
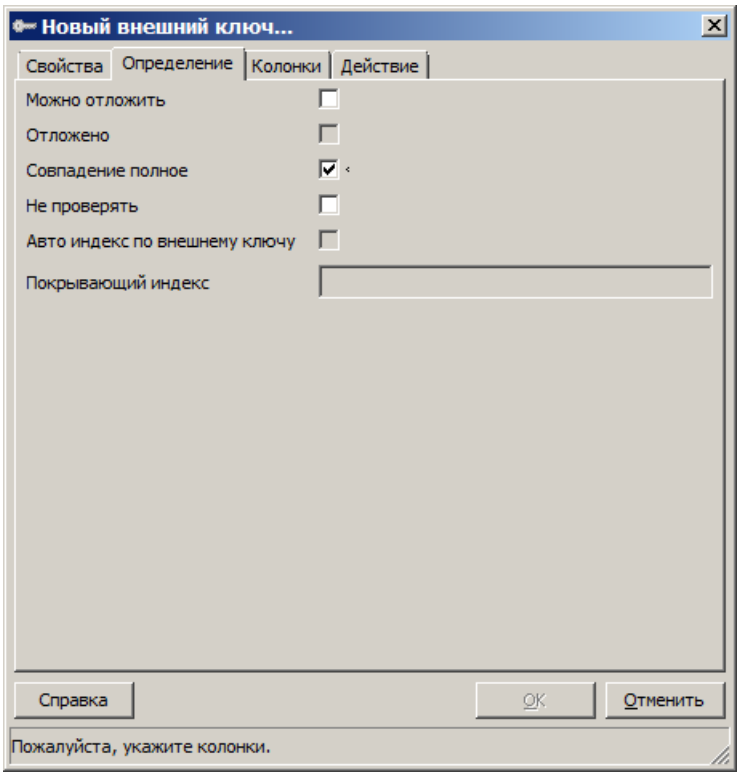


Рисунок 2.21 – Закладка Обмеження вікна властивостей таблиці

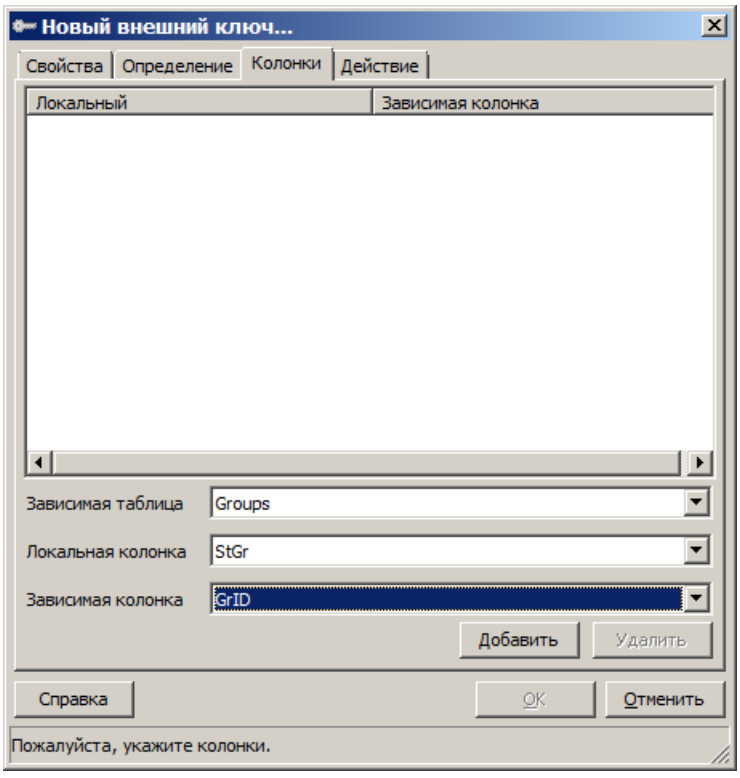
У ньому вікні потрібно відкрити закладку Обмеження, вибрати в списку, що випадає, тип обмеження Вторинний (Зовнішній) ключ і натиснути кнопку Додати. В результаті виконання цих дій з'явиться вікно Новий зовнішній ключ (Рисунок 2.22)



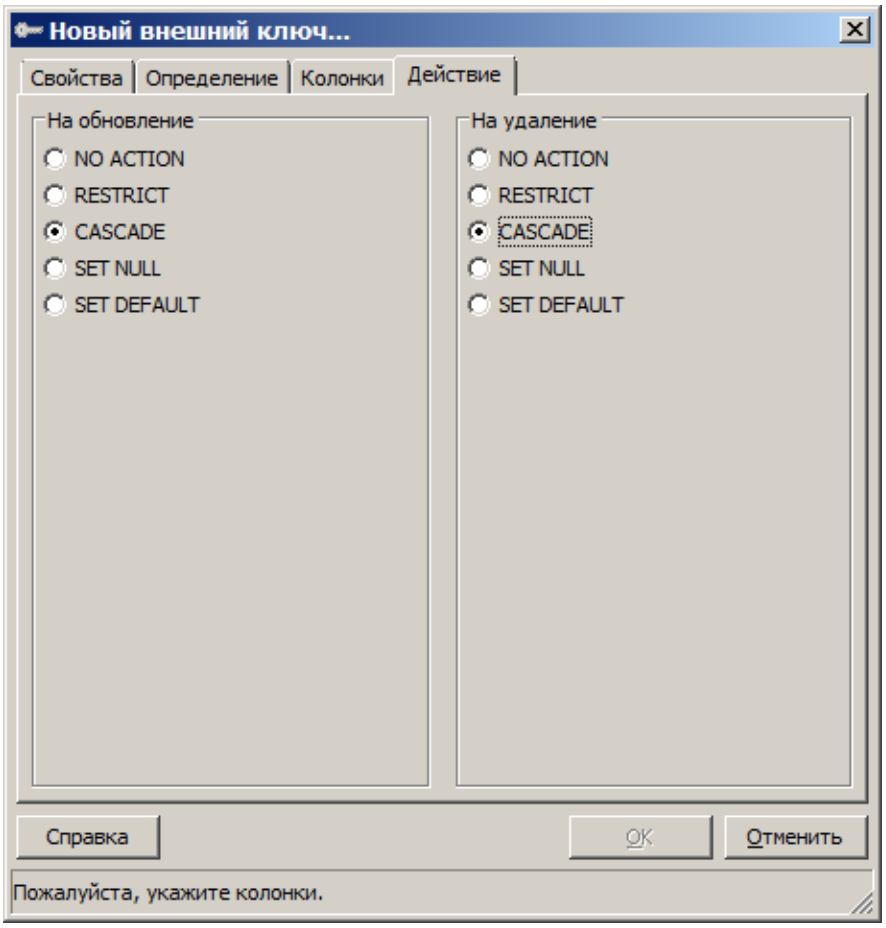
а) - Ім'я



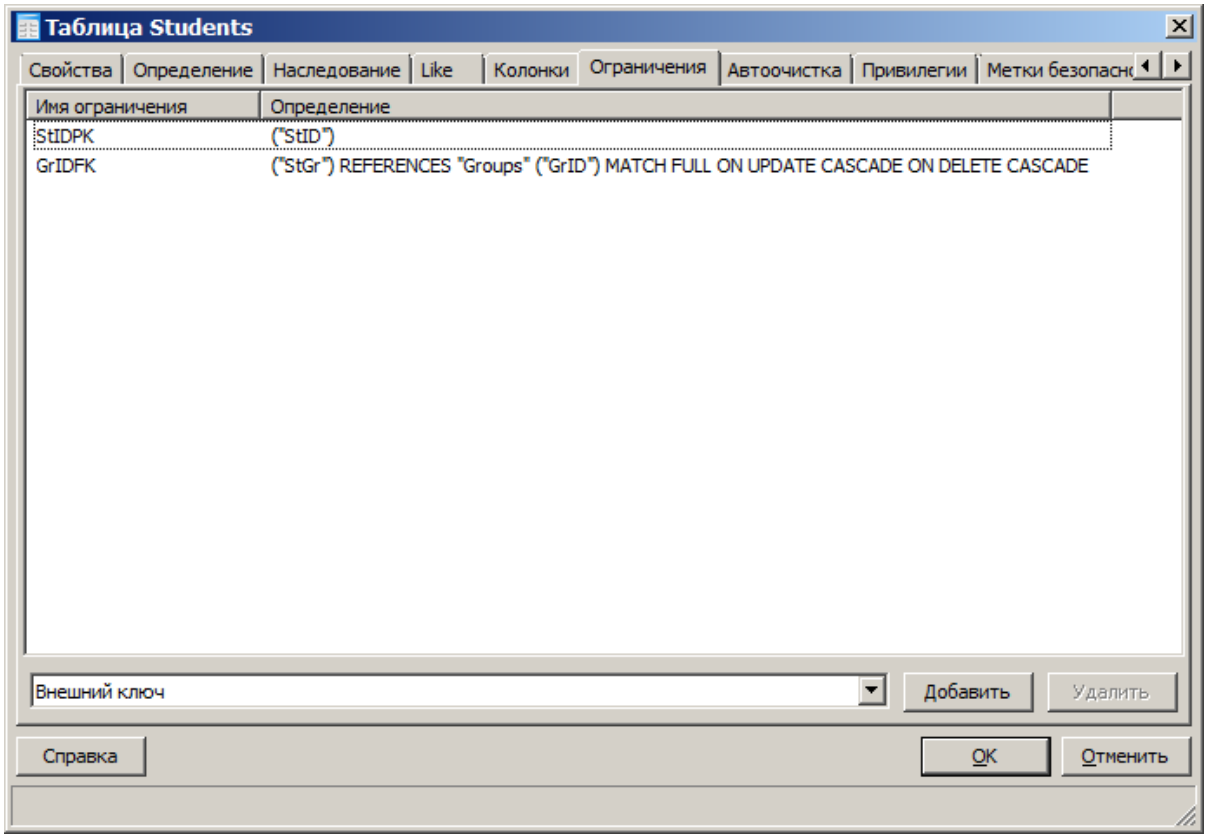
б) - Обмеження



в) - Колонки



г) - Дії



д) - Обмеження

Рисунок 2.22 – Вікно Новий зовнішній ключ. Закладка Властивості

Щоб створити новий зовнішній ключ необхідно виконати наступні дії:

1) на закладці Властивості задати ім'я зовнішнього ключа, наприклад, GrIDFK (FK від англ. Foreign Key - зовнішній ключ);

2) на закладці Визначення відзначити властивість Повний збіг, яка говорить про те, що значення зовнішнього ключа повинні повністю збігатися зі значеннями первинного ключа;

3) на закладці Колонки вибрати:

а) у полі Залежна таблиця вибрати таблицю, на яку посилається зовнішній ключ;

б) у полі Локальна колонка вибрати колонку вихідної таблиці, яка визначає зовнішній ключ (для наведеного прикладу це колонка GrNo таблиці Students);

в) у полі Залежна колонка вибрати колонку первинного ключа залежної таблиці, на яку посилається зовнішній ключ (колонка GrID таблиці Groups).

Примітка. Для наочності ім'я колонки зовнішнього ключа таблиці Students, може збігатися з ім'ям колонки первинного ключа GrID таблиці Groups. На практиці це не обов'язково і не завжди вітається;

4) на закладці Дія вибрати дію, яка виконуватиметься при зміні або видаленні значення первинного ключа. Ці дії є засобом забезпечення цілісності БД.

Можливі варіанти дій:

а) NO ACTION – ніяких додаткових дій і обмежень;

б) RESTRICT – видалення / зміна значення первинного ключа забороняється, якщо на нього посилається який-небудь зовнішній ключ. Цю дію встановлено за замовчуванням;

в) CASCADE – видалення рядка в батьківській таблиці призводить до видалення всіх пов'язаних з нею рядків дочірньої таблиці (наприклад, якщо видалити запис про групу в таблиці Groups, то автоматично будуть видалені всі записи про її студентах в таблиці Students). Зміна значення первинного ключа батьківської таблиці призводить до відповідної зміни значень зовнішніх ключів дочірньої таблиці. Наприклад, якщо змінити будь-яке значення в колонці GrNo таблиці Groups, то автоматично будуть змінені пов'язані з ним значення колонки GrNo в таблиці Students;

г) SET NULL – видалення / зміна первинного ключа батьківської таблиці призводить до установки в значення NULL всіх зовнішніх ключів дочірньої таблиці, які посилаються на видалене / змінене значення первинного ключа батьківської таблиці. Наприклад, якщо видалити / змінити значення в колонці GrNo таблиці Groups, то пов'язані з ним значення колонки GrNo в таблиці Students автоматично будуть встановлені в NULL;

д) SET DEFAULT – видалення / зміна первинного ключа батьківської таблиці призводить до установки в значення за замовчуванням всіх зовнішніх ключів дочірньої таблиці, які посилаються на видалене / змінене значення первинного ключа батьківської таблиці.

Щоб побачити результат змін колонки, таблиці або БД необхідно виділити відповідний об'єкт у вікні Браузер об'єктів і виконати його оновлення (Рисунок 2.23).

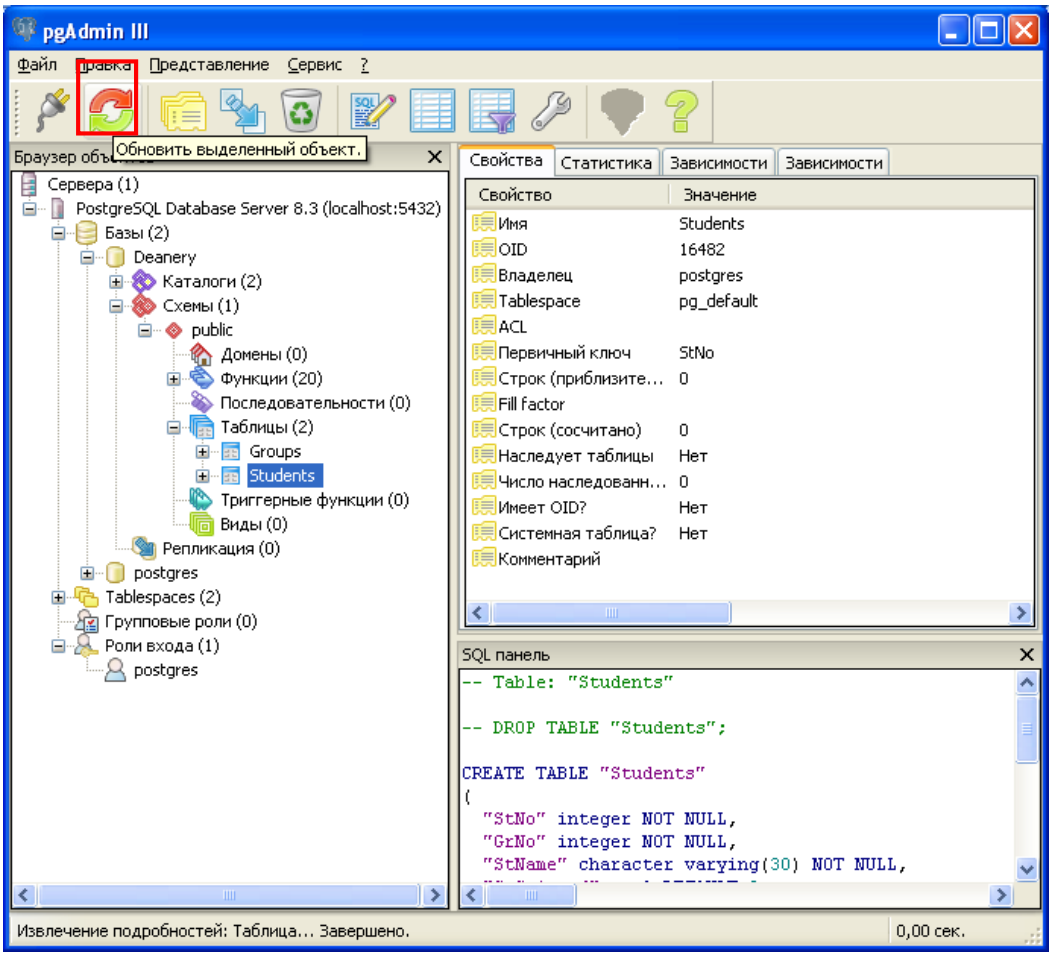


Рисунок 2.23 – Оновлення об'єкта БД

2.8 Унікальність значень полів одного або групи стовпчиків

Є такі задачі, коли значення полів одного, пари, трійки стовпчиків не може повторюватися. Наочний приклад – це номер студентського білета, який не може бути однаковий у декількох студентів.

Більш складна ситуація, коли не може повторюватися пара значень, наприклад, не може бути декілька білетів на одне й теж місце в одному вагоні потягу, тобто пара "номер вагона"-"номер місця" має бути унікальною.

Аналогічно не може дублюватися трійка значень.

Розглянемо спосіб встановлення унікальності пари значень 2-х стовпчиків таблиці. Інші випадки будуть очевидні.

Дисципліна Subject (Рисунок 2.24) може викладатися з різною кількістю годин Hours, але не потрібно мати подібні пари значень Subject-Hours тобто такі пари мають бути унікальними. Тому в таблиці значення пар SbjName-SbjHours не можуть повторюватися

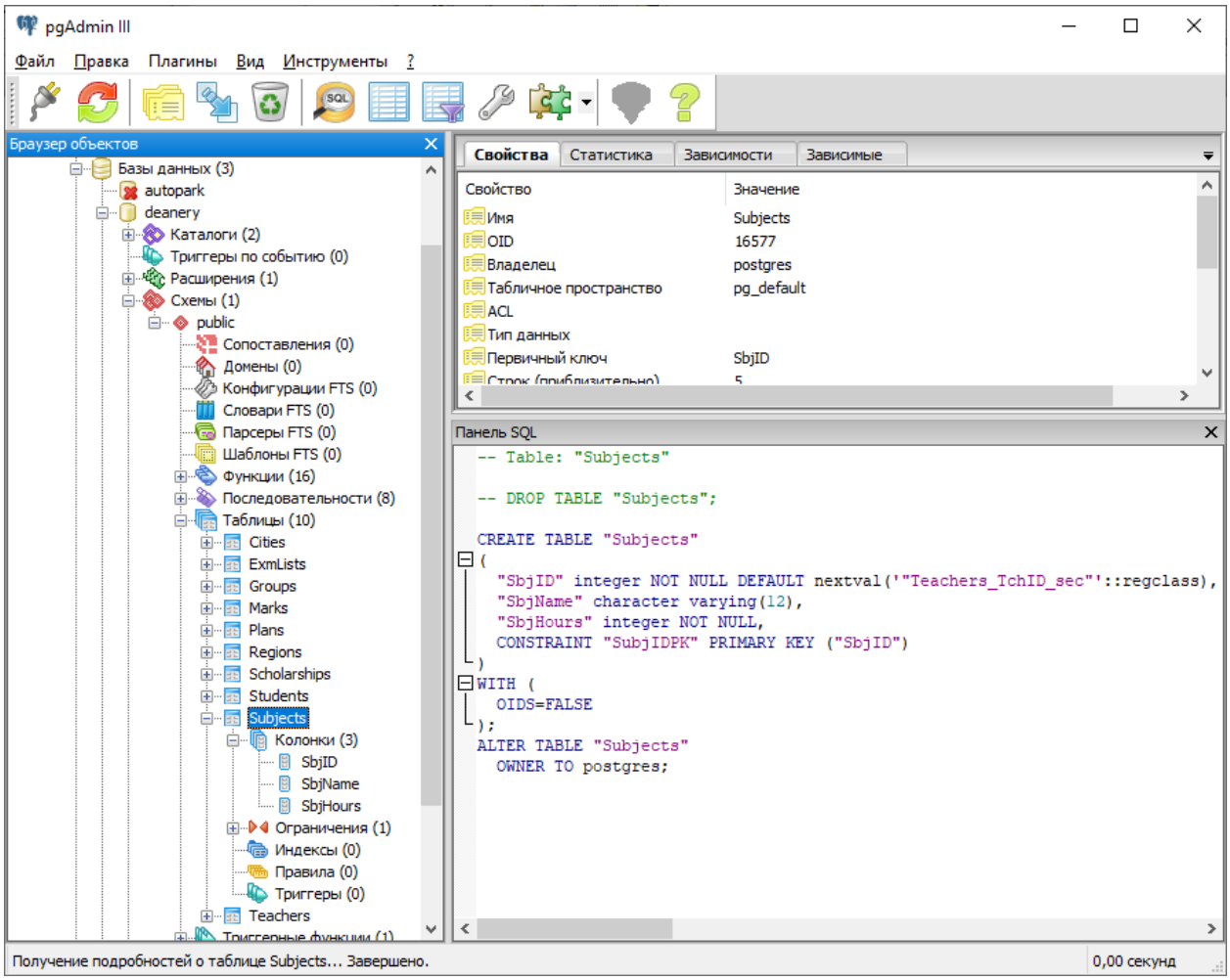


Рисунок 2.24 – Таблиця Subjects

Вибираємо Властивості таблиці (Рисунок 2.25)

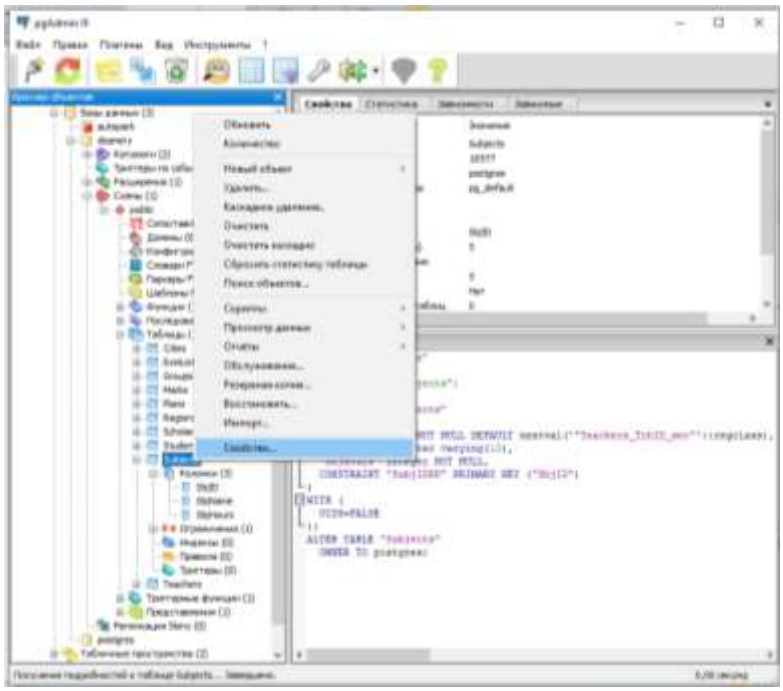


Рисунок 2.25 - Властивості

Переходимо на закладку Обмеження і у спадаючому списку вибираємо Обмеження унікальності (Рисунок 2.26)

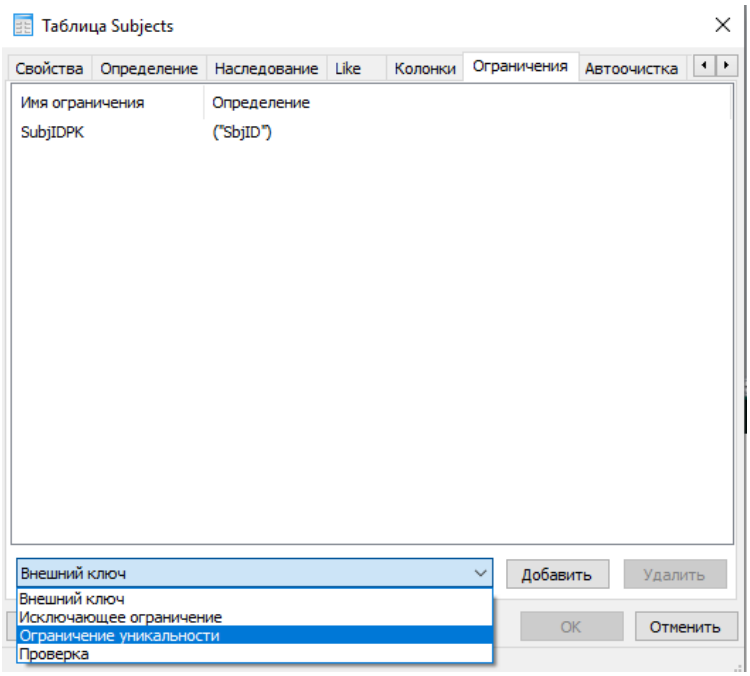


Рисунок 2.26 – Обмеження унікальності

Натискаємо кнопку Додати (Рисунок 2.27)

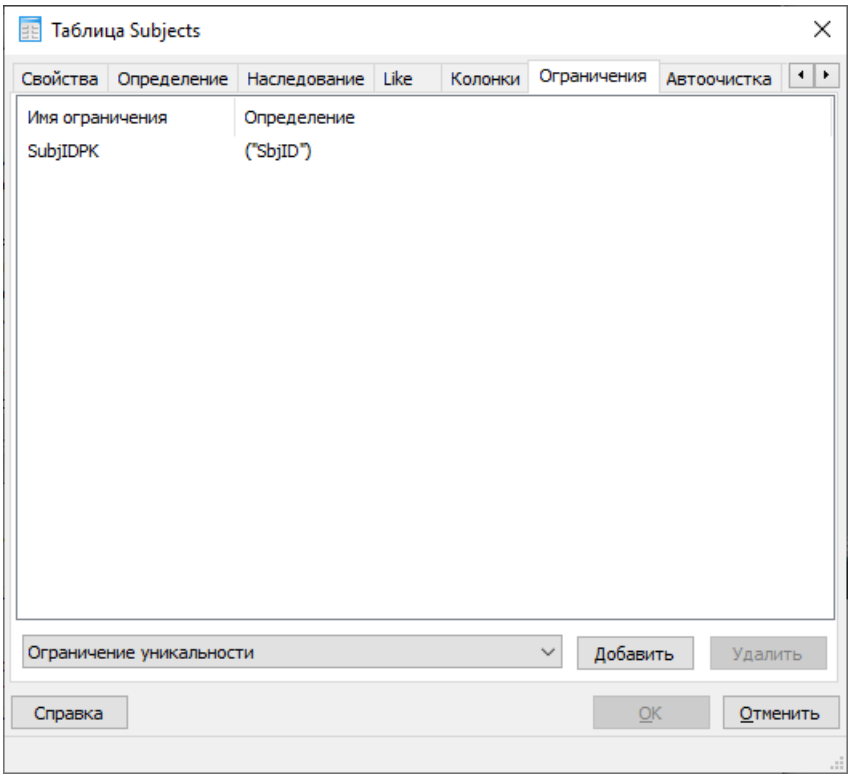


Рисунок 2.27 – Додавання обмеження

Вводимо назву обмеження (Рисунок 2.28)

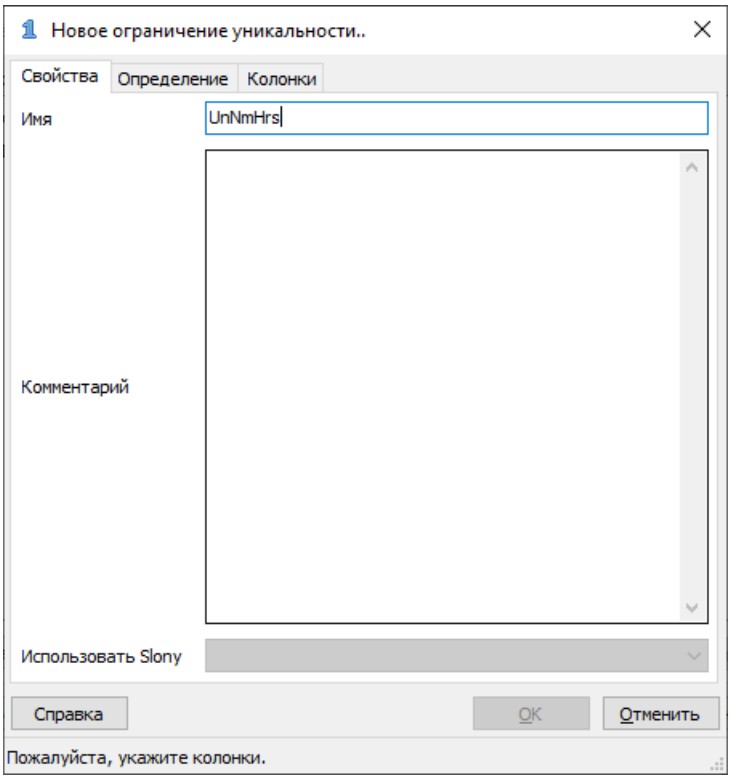


Рисунок 2.28 – Назва обмеження

Переходимо до закладки Колонки.
Вибираємо і додаємо першу колонку (Рисунок 2.29)

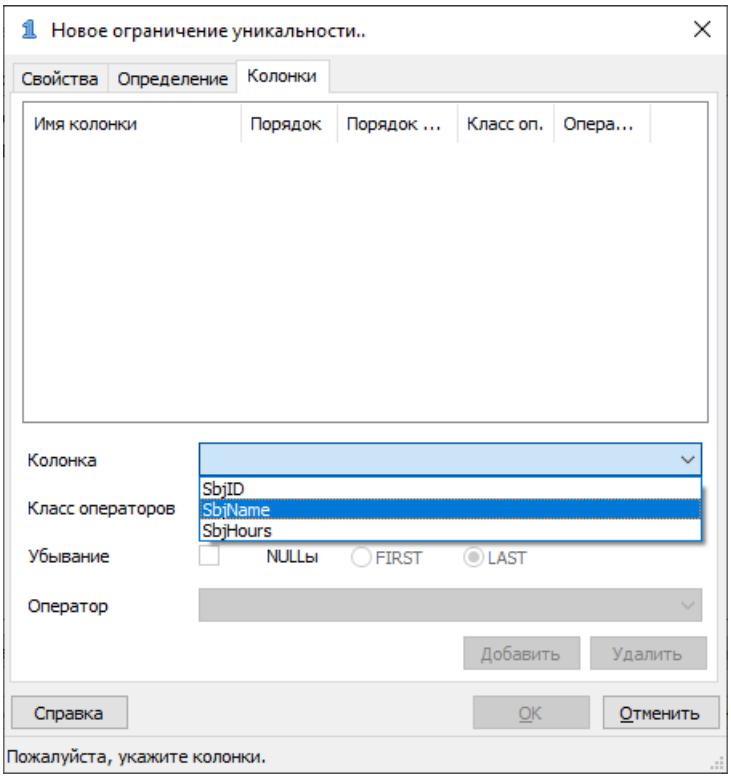


Рисунок 2.29 – Перша колонка обмеження

Вибираємо і додаємо другу колонку (Рисунок 2.30)

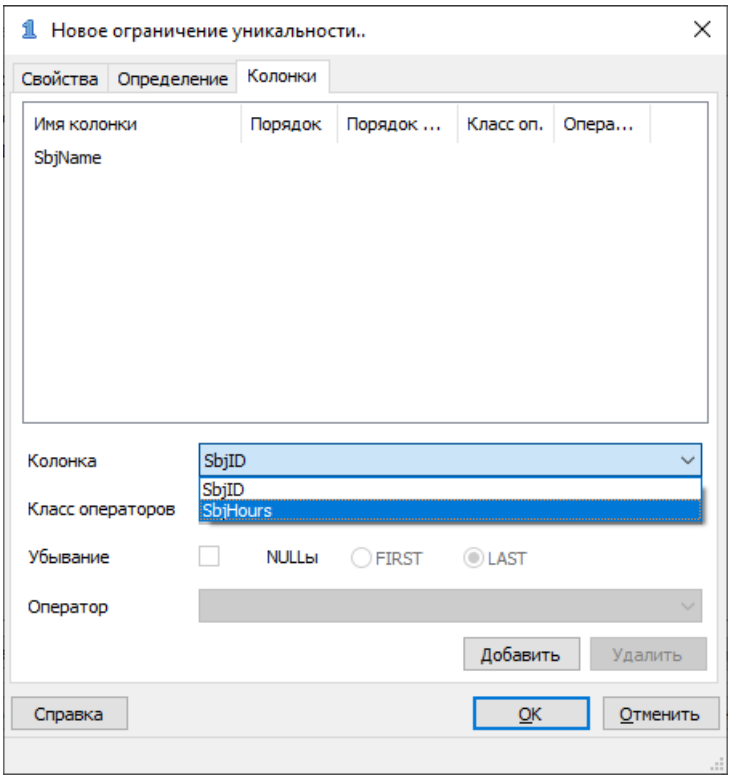


Рисунок 2.30 - Друга колонка обмеження

Отримаємо (Рисунок 2.31)

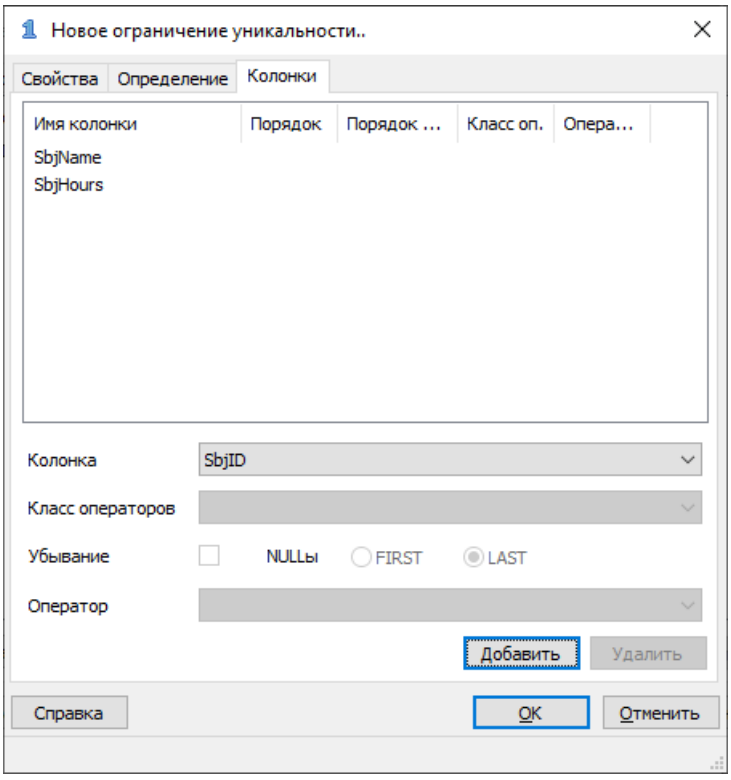


Рисунок 2.31 – Отримані обмеження унікальності

Натискаємо OK і отримуємо нове обмеження (Рисунок 2.32)

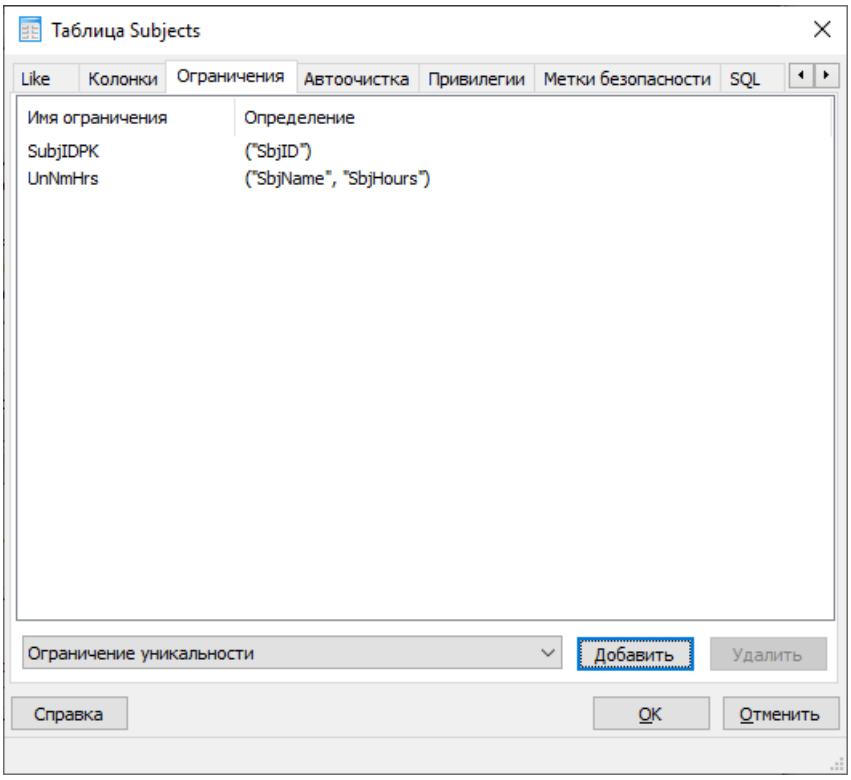


Рисунок 2.32 – Нове обмеження

Переходимо на закладку SQL і бачимо фрагмент скрипту (Рисунок 2.33)

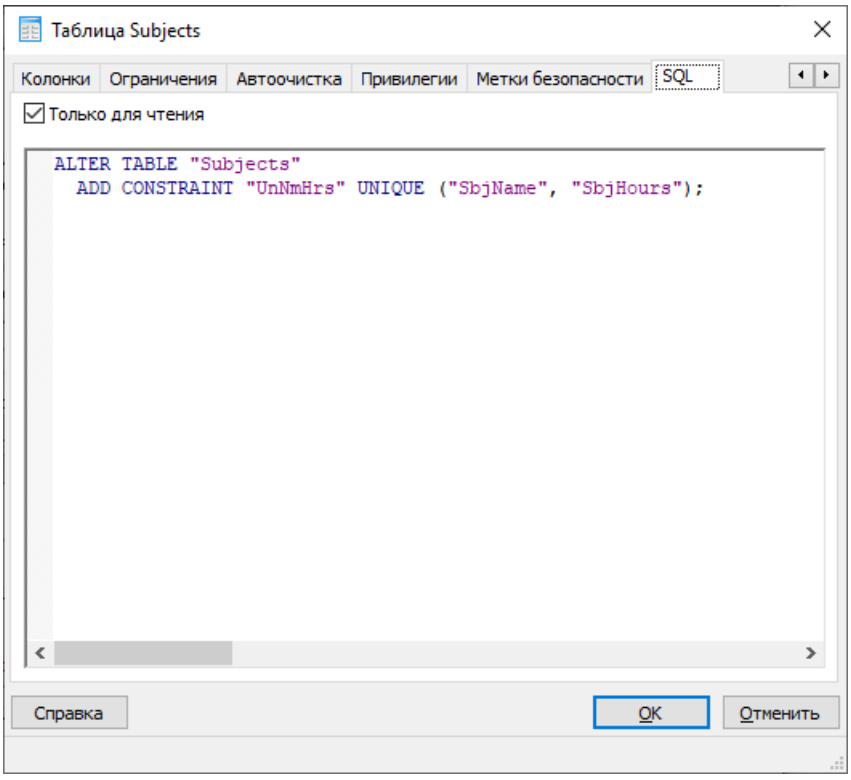


Рисунок 2.33 - Фрагмент скрипту

Повертаємося до опису таблиці (Рисунок 2.34)

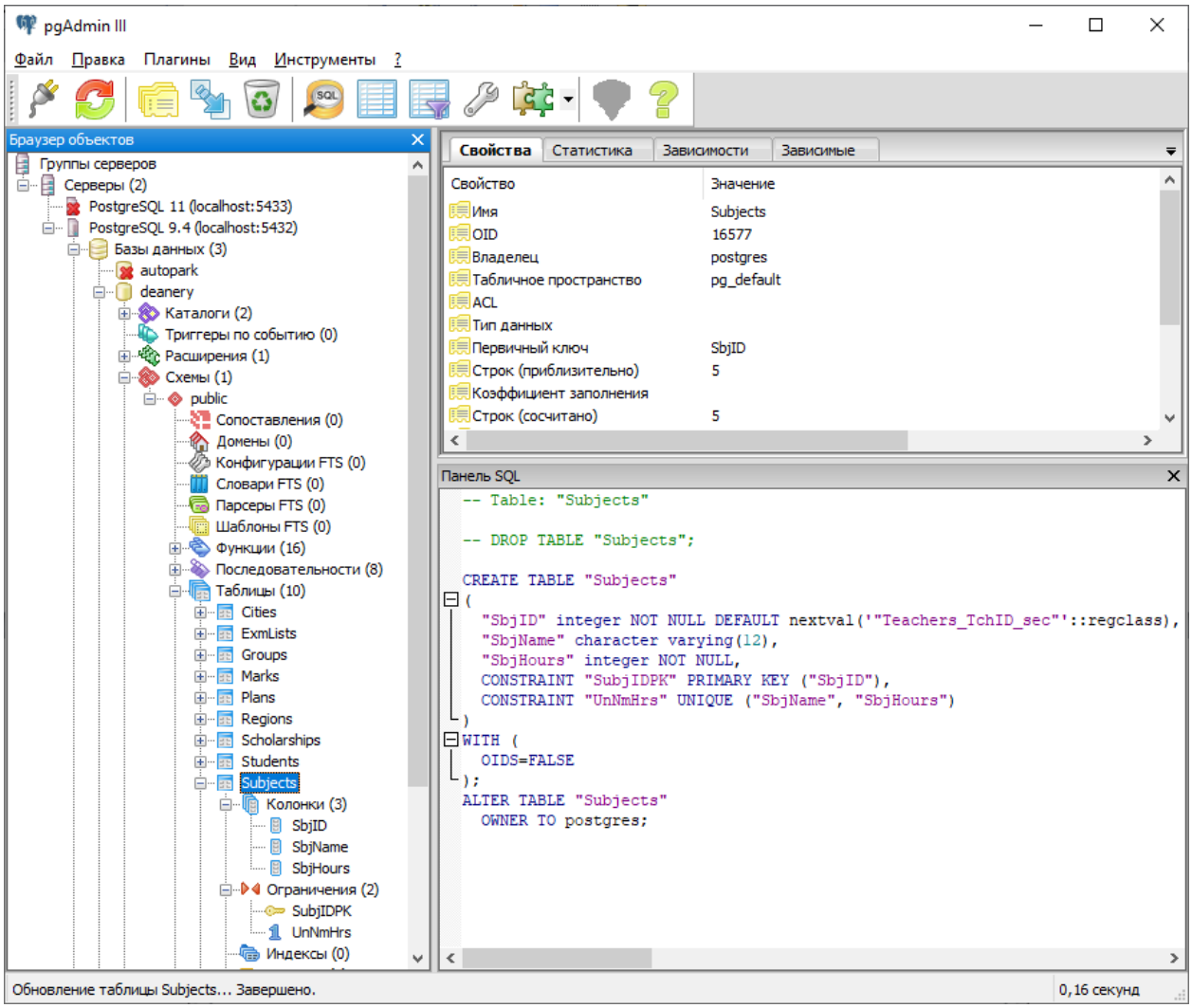


Рисунок 2.34 – Опис таблиці Subject

3 ПЕРЕГЛЯД, ВВЕДЕННЯ ТА РЕДАГУВАННЯ ДАНИХ ТАБЛИЦІ

Для перегляду введення або редагування вмісту певної таблиці необхідно вибрати її у вікні "Браузер об'єктів" і натиснути кнопку "Перегляд даних в обраному об'єкті" (Рисунок 3.1)

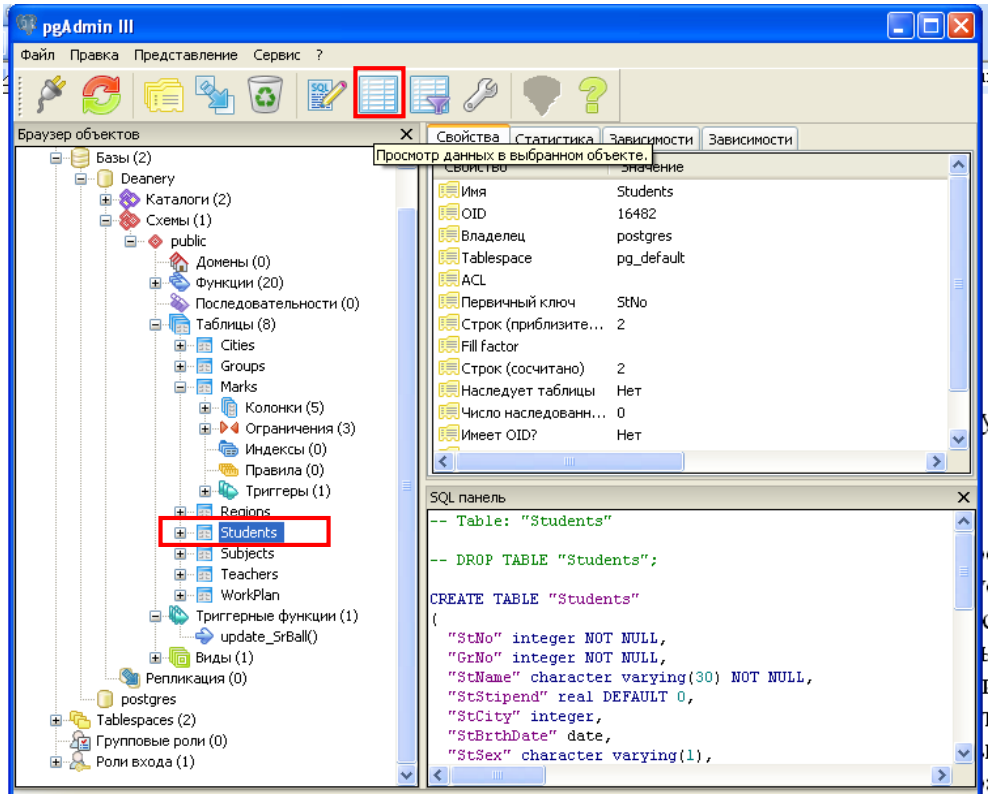


Рисунок 3.1 – Вибір таблиці для редагування

Після цього відкриється вікно перегляду/редагування даних обраної таблиці (Рисунок 3.2)

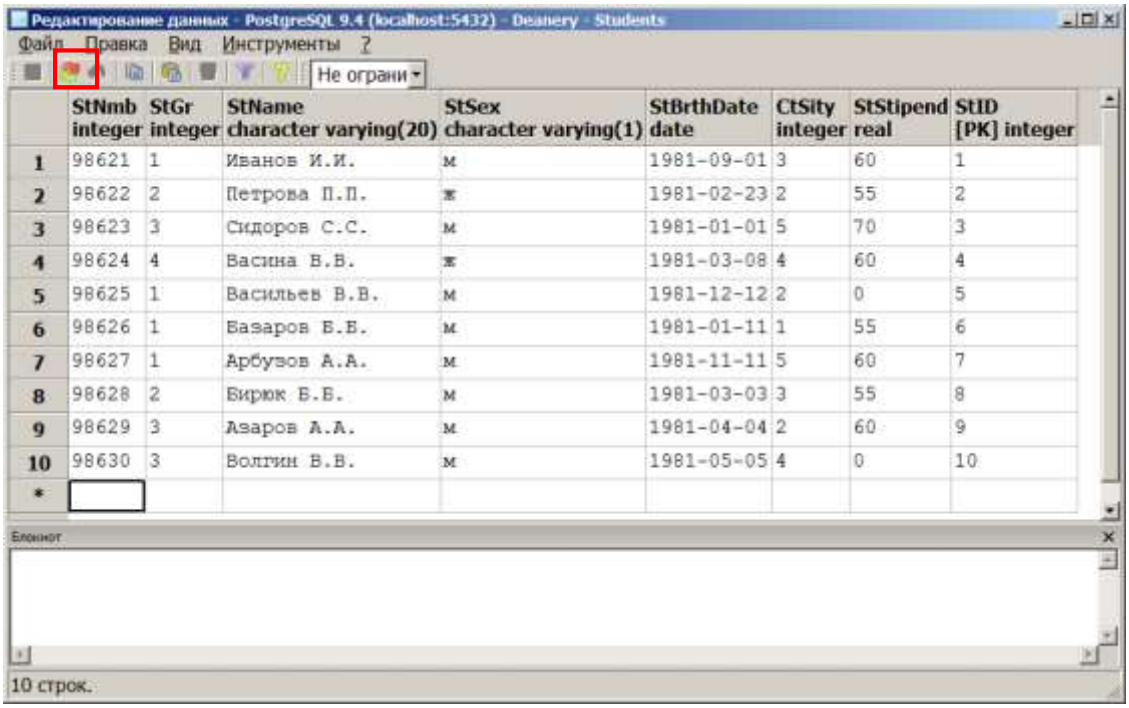


Рисунок 3.2 – Вікно редагування даних таблиці

Введення даних виконується в відповідні поля рядка, зазначеного символом "*****". Введення поля ключа StID виконувати не треба. Це поле має тип "serial" і заповнюється автоматично після збереження/оновлення даних в БД.

Редагування даних виконується в відповідних полях виділеного рядка. Редагування поля ключа StID виконувати не треба. Воно залишається незмінним на весь час існування БД.

Для збереження нових чи оновлення існуючих даних в БД слід натиснути кнопку "Оновити" (Рисунок 3.2).

4 ВИКОНАННЯ SQL-ЗАПИТІВ КОРИСТУВАЧА

4.1 Вікно запитів

Для створення і виконання SQL-запитів користувача слід натиснути кнопку "Виконати SQL-запити користувача" (Рисунок 4.1).

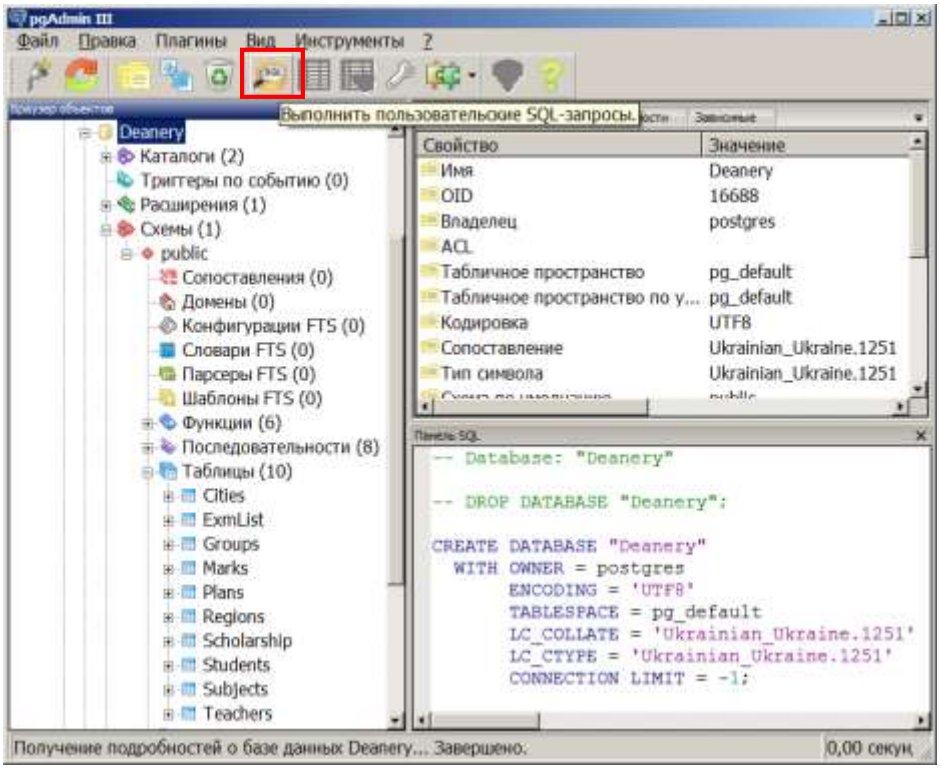


Рисунок 4.1 – Виклик вікна SQL-запитів користувача

У вікні, що відкрилося, на закладці "Редактор SQL", формується запит.

Слід врахувати, що PostgreSQL чутливий до регістру. За замовчуванням для іменування баз даних, їх таблиць і стовпців використовуються латинські символи нижнього регістру, цифр і символ підкреслення. Якщо ж використовуються інші символи, то імена таблиць і стовпців слід укладати в пару подвійних лапок "".

4.2 Особливості формування запитів

Використання PostgreSQL [12] має деякі особливості:

1. PostgreSQL чутливий до регістру. Тому імена БД, таблиць, стовпчиків і значень полів в SQL-запитах слід записувати суворо так, як вони визначені в БД. Щоб уникнути непорозумінь, краще назви БД, таблиць та стовпчиків писати малими (рядковими) літерами!
2. Якщо імена БД, таблиць і стовпчиків містять не лише малі літери, ці імена в SQL-запитах слід укладати в подвійні лапки "…..";
3. Рядкові значення в SQL-запитах слід укладати в одинарні лапки '…';
4. При програмному доступі до БД у запитах – рядкових змінних чи константах, слід передбачити особливість представлення символу ', наприклад в запиті

2. String query = "SELECT * FROM \"BOOKS\" WHERE \"BookID\" < 5
a. AND \"BName\" = '10 хвилин на урок Windows 10' ";

Значення рядкової змінної, наприклад, query визначено через пару лапок "…", тобто рядкове значення відкривається символом " і закривається символом ". Таким чином перший символ " відкриває рядок, а другий по порядку символ " має закрити рядок. Але за вимогами PostgreSQL ці символи використовується, щоб показати, що імена таблиць або стовпчиків містять прописні літери, наприклад BOOKS. Тому внутрішні символи " в SQL-запиту треба зв'язати. Для цього використовується пара символів \".

Відповідно рядкове значення в SQL-запитах представляється парою символів '…', наприклад '10 хвилин на урок Windows 10'.

4.3 Виконання окремого запиту

Для виконання запиту слід натиснути кнопку "Виконати запит". Результат з'явиться на закладці "Вивід даних" (Рисунок 4.2).

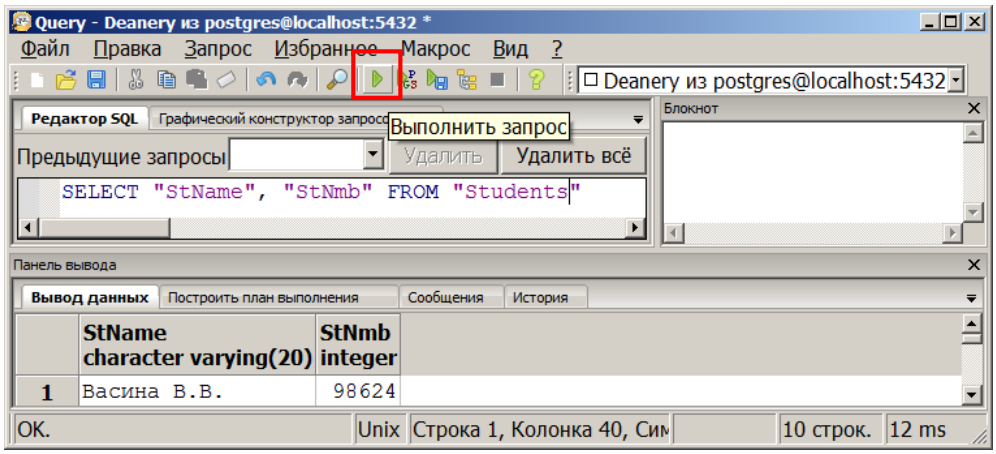


Рисунок 4.2 - Виконання запиту

4.4 Виконання групи запитів

У вікні "Редактор SQL" можна сформувати декілька запитів (Рисунок 4.3). Їх можна виконати одразу або почергово.

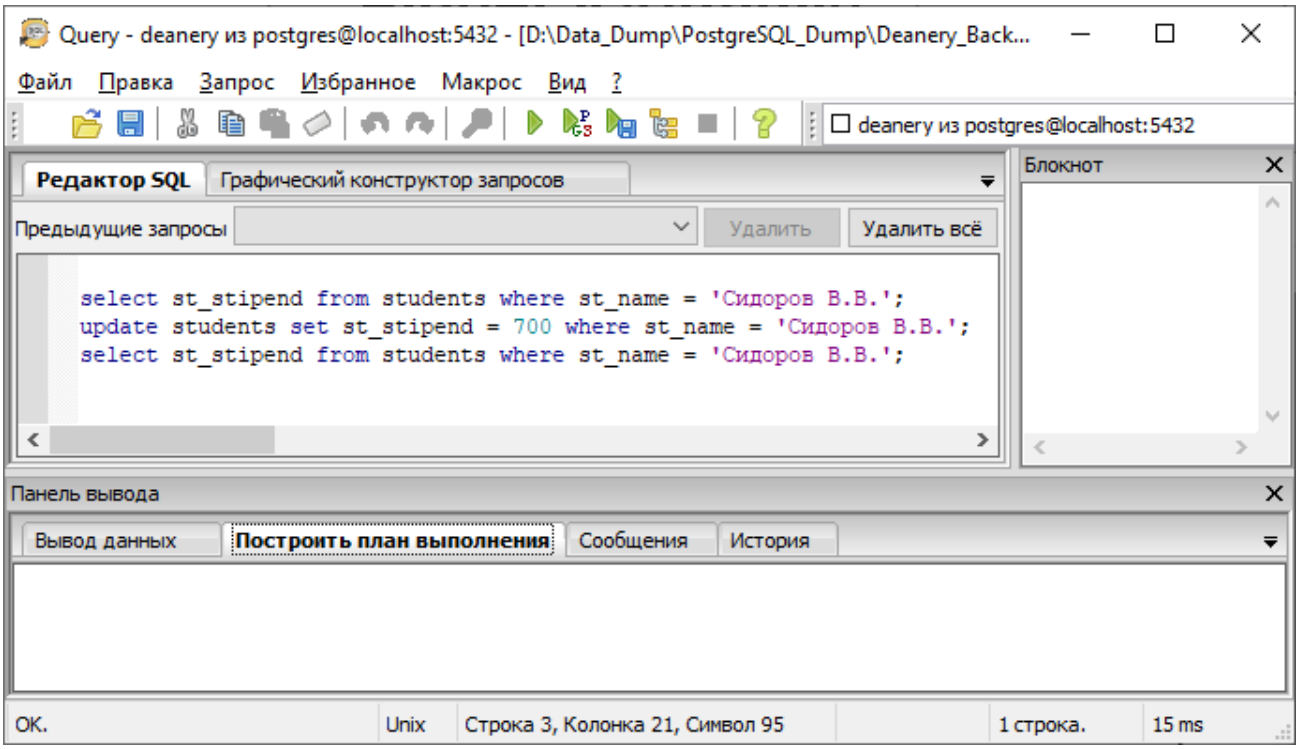


Рисунок 4.3 - Декілька запитів у вікні Редактор SQL

Для виконання всієї групи запитів одразу необхідно просто натиснути кнопку "Виконати запит". Повідомлення, пов’язане з останньою командою з’явиться у Вікні виводу даних.

5 ІНДЕКСУВАННЯ ТАБЛИЦЬ БАЗ ДАНИХ

5.1 Індекси

Індекси є ефективним засобом збільшення продуктивності БД. Використовуючи індекс, сервер БД може знаходити і вибирати потрібні рядки таблиці набагато швидше, ніж без нього. Однак з індексами пов'язана додаткове навантаження на СУБД в цілому, тому застосовувати їх слід зважено.

Індекси обов'язково створюються для всіх первинних ключів, якщо є, то й альтернативних ключів.

На практиці вважається, що допустимо не більше 3-х індексів по неключовим полям таблиці.

Індекси, які використовуються в запитах зрідка або взагалі ніколи не використовуються, повинні бути видалені.

5.2 Створення та видалення індексів

5.2.1 Створення індексу

Створити індекс для стовпчика таблиці можна за допомогою наступної команди:

CREATE INDEX ім'я_індексу ON ім'я_таблиці (ім'я стовпчика)

CREATE INDEX ім'я_індексу ON ім'я_таблиці (ім'я стовпчика) USING ***метод***

Команда CREATE INDEX створює індекси за вказаним стовпцем (ями) заданого відношення, яким може бути таблиця або матеріалізоване представлення (уявлення).

Ім'я індексу може бути довільним, головне, щоб воно дозволяло зрозуміти, для чого цей індекс.

Повний синтаксис команди

CREATE [UNIQUE] INDEX [CONCURRENTLY]
[[IF NOT EXISTS] ***і'мя***] ON ***і'мя_таблиці*** [USING ***метод***]
({ ***і'мя_стовпчика*** | (***вираз***) } [COLLATE ***правило_сортування***]
[***клас_операторів***] [ASC | DESC] [NULLS { FIRST | LAST }] [, ...])
[WITH (***параметр_зберігання*** [= ***значення***] [, ...])]
[TABLESPACE ***табл_простір***]
[WHERE ***предикат***]

Команда CREATE INDEX є достатньо складною і її параметри та виконання пов'язане з особливостями організації і функціонування бази даних. Тому нижче будуть розглянуті найбільш загальні параметри команди, а за поясненнями інших параметрів треба звернутися до документації.

Параметри:

✓ UNIQUE. Вказує, що система повинна контролювати повторювані значення в таблиці при створенні індексу (якщо в таблиці вже є дані) і при кожному додаванні даних. Спроби вставити або змінити дані, при яких буде порушена унікальність індексу, будуть завершуватися помилкою;

✓ CONCURRENTLY. З цим зазначенням PostgreSQL побудує індекс, не встановлюючи ніяких блокувань, які б запобігали додавання, зміна або видалення записів в таблиці, тоді як за замовчуванням операція побудови індексу блокує запис (але не читання) даних в таблиці до свого завершення. Зі створенням індексу в цьому режимі пов'язаний ряд особливостей (див документацію).

Для тимчасових таблиць CREATE INDEX завжди виконується більш простим, неблокуючим способом, оскільки вони не можуть використовуватися ніякими іншими сеансами;

✓ IF NOT EXISTS. Не вважати помилкою, якщо індекс з таким ім'ям вже існує. У цьому випадку буде видано повідомлення;

✓ ім'я. Ім'я створюваного індексу. Вказівка на схему при цьому не допускається; індекс завжди відноситься до тієї ж схеми, що і батьківська таблиця. Якщо ім'я не вказано, PostgreSQL формує відповідне ім'я на імені батьківської таблиці і іменах індексованих стовпців;

✓ ім'я_таблиці. Ім'я таблиці, що індексується (можливо, доповнене схемою);

✓ метод. Ім'я застосовуваного методу індексу. Можливі варіанти: btree, hash, gist, spgist, gin і brin. За замовчуванням мається на увазі метод btree;

✓ ім'я_стовпчика. Ім'я стовпчика таблиці;

✓ вираз. Вираз з одним або декількома стовпцями таблиці. Зазвичай вираз має записуватися в дужках, однак дужки можна опустити, якщо вираз записано у вигляді виклику функції;

✓ правило_сортування. Ім'я правила сортування, що застосовується для індексу. За замовчуванням використовується правило сортування, задане для стовпчика, що індексується, або отримане для результату вирази індексу. Індекси з нестандартними правилами сортування можуть бути корисні для запитів, що включають вирази з такими правилами;

✓ клас_операторів. Ім'я класу операторів (див. документацію):

- ASC. Вказує порядок сортування по зростанню (за замовчуванням);
- DESC. Вказує порядок сортування по спадаючій;
- NULLS FIRST. Вказує, що значення NULL після сортування виявляються перед іншими. Це поведінка за умовчанням з порядком сортування DESC;
- NULLS LAST. Вказує, що значення NULL після сортування виявляються після інших. Це поведінка за умовчанням з порядком сортування ASC;

✓ параметр_зберігання. Ім'я специфічного для індексу параметра зберігання (див. документацію);

✓ табл_простір. Табличний простір, в якому буде створено індекс. Якщо не визначено, вибирається default_tablespace, або temp_tablespaces, при створенні індексу тимчасової таблиці;

✓ предикат. Вираз обмеження для часткового індексу.

5.2.2 Видалення індексу

Для видалення індексу використовується команда

DROP INDEX ім'я

Команда DROP INDEX є достатньо складною і її параметри та виконання пов'язане з особливостями організації і функціонування бази даних. Тому нижче будуть розглянуті найбільш загальні параметри команди, а за поясненнями інших параметрів треба звернутися до документації.

Повний синтаксис команди

DROP INDEX [CONCURRENTLY] [IF EXISTS] ***ім'я*** [, **...**] [CASCADE | RESTRICT]

Параметри:

✓ CONCURRENTLY. З цим зазначенням індекс видаляється, не блокуючи одночасні операції вибірки, додавання, зміни і видалення даних в таблиці індексу. Звичайний оператор DROP INDEX запитує виняткове блокування для таблиці, не допускаючи інші звернення до неї до завершення видалення. Якщо ж додано цю вказівку, команда, навпаки, буде чекати завершення конфліктуючих транзакцій.

Застосовуючи вказівку CONCURRENTLY, треба враховувати кілька особливостей. Зокрема, при цьому можна задати ім'я тільки одного індексу, а параметр CASCADE не підтримується. (Таким чином, індекс, що підтримує обмеження UNIQUE або PRIMARY KEY, так видалити не можна.) Крім того, звичайну команду DROP INDEX можна виконати в блоці транзакції, а DROP INDEX CONCURRENTLY - немає.

Для тимчасових таблиць DROP INDEX завжди виконується більш простим, неблокуючим способом, так як вони не можуть використовуватися ніякими іншими сеансами.

✓ IF EXISTS. Не вважати помилкою, якщо індекс не існує. У цьому випадку буде видано повідомлення.

✓ ім'я. Ім'я індексу, можливо доповнене схемою, що підлягає видаленню.

✓ CASCADE. Автоматично видаляти об'єкти, залежні від даного індексу, і, в свою чергу, всі залежні від них об'єкти.

✓ RESTRICT. Відмовити у видаленні індексу, якщо від нього залежать будь-які об'єкти. Це поведінка за умовчанням.

5.2.3 Поведінка та використання індексів

Додавати і видаляти індекси можна в будь-який час існування БД.

Коли індекс створений, ніякі додаткові дії не потрібні. PostgreSQL буде сам оновлювати індекс при зміні даних в таблиці і буде сам використовувати цей індекс в запитах, де, на його думку, це буде ефективніше, ніж сканування всієї таблиці.

Для ефективного використання індексів в запитах, необхідно періодично запускати команду ANALYZE для поновлення статистичних даних, на основі яких планувальник запитів приймає рішення.

Індекси можуть застосовуватися в пошуку зі з’єднанням. Таким чином, індекс, визначений для стовпчика, який бере участь в умові з'єднання, може значно прискорити запити з JOIN.

Індекси можуть бути корисні також при виконанні команд UPDATE і DELETE з умовами пошуку.

5.3 Типи індексів

На сьогодні PostgreSQL підтримує такі типи індексів – B-дерево, хеш, GiST, SP-GiST, GIN і BRIN. Для різних типів індексів використовуються різні алгоритми впорядкування, орієнтовані на певні типи запитів.

За замовчуванням команда CREATE INDEX створює індекси типу B-дерево, ефективні в більшості випадків.

5.3.1 B-дерева

За замовчуванням команда CREATE INDEX створює індекси типу B-дерево, ефективні в більшості випадків. B-дерева можуть працювати в умовах пошуку на рівність - дорівнює, не дорівнює та інші, а також у перевірках діапазонів з даними типу "більше A і менше B" або аналогічних їм конструкціях BETWEEN і IN.

5.3.2 Хеш-індекси

Хеш-індекси працюють тільки з простими умовами порівняння "дорівнює". Створити такий індекс можна за допомогою такої команди:

```
CREATE INDEX імя_індекса ON ім'я_таблиці
        USING HASH (ім'я стовпця)
```

Однією з особливостей є його орієнтація на роботу з деяким типом структурованих даних. Так стандартний дистрибутив PostgreSQL надає можливість роботи з декількома типами двовимірних геометричних даних, що використовуються в різних областях картографії. Для можливості індексування таких типів є ряд інструментів.

5.3.3 Індекси GiST, SP-GiST і GiN

Індекси GiST, SP-GiST і GiN представляють собою інфраструктуру, що дозволяє реалізувати різні стратегії індексування в залежності від заданого набору (класу) операторів.

5.3.3.1 Індекси GiST

Узагальнене пошукове дерево - індекси GiST (Generalized Search Tree) це структура, яка є узагальненим різновидом R-дерева - RD-деревом, і

надає стандартні методи навігації по дереву і його оновлення - розщеплення і видалення вузлів.

Примітка 1. R-дерево подібно B-дереву, але використовується для організації доступу до просторових даних, тобто для індексації багатовимірної інформації, наприклад, географічних даних. Ця структура даних розбиває багатовимірний простір на множину ієрархічно вкладених і, можливо, таких прямокутників, що перетинаються, прямокутних паралелепіпедів тощо.

Примітка 2. RD-дерево є адаптацією R-дерева, яка використовує природну аналогію між просторовими об'єктами та множинами.

Індекс GiST є збалансоване по висоті RD-дерево, кінцеві вузли (листя) якого містять пари (key, rid), де key - ключ, а rid - покажчик на відповідний запис на сторінці даних. Внутрішні вузли дерева містять пари елементів, що виконують роль навігаторів з доступу до необхідного листу дерева.

Створити GiST індекс можна командою:

CREATE INDEX імя_індекса ON ім'я_таблиці
USING GiST (ім'я стовпця)

Для GiST-дерева визначені базові методи SEARCH, INSERT, DELETE, і інтерфейс для написання користувальницьких методів, з допомогою яких можна управляти роботою цих (базових) методів.

Стандартний дистрибутив PostgreSQL включає класи операторів GiST для двовимірних типів геометричних даних.

Примітка. Індекси GiST і GIN (див. нижче) широко використовуються при повнотекстовому пошуку PostgreSQL. Це означає, що для кожного вектору tsvector, що описує всі лексеми документа, створюється сигнатура, що описує, які з лексем входять в даний tsvector.

5.3.3.2 Індекси SP-GiST

Індексація з розбивкою простору SP-GiST (Space-Partitioned GiST), полягає в розбитті області значень на під-області, що не перетинаються, кожна з яких, в свою чергу, також може бути розбита. Таке розбиття породжує незбалансовані дерева на відміну від B-дерев GiST.

Індекси SP-GiST зіставляє вузли дерева пошуку зі сторінками на диску так, щоб при пошуку потрібно звертатися тільки до багатьох сторінок на диску, навіть якщо при цьому потрібно переглянути безліч вузлів.

Створити SP-GiST індекс можна за допомогою такої команди:

CREATE INDEX імя_індекса ON ім'я_таблиці
USING SP-GiST (ім'я стовпця)

Стандартний дистрибутив PostgreSQL включає класи операторів SP-GiST для точок в двовимірному просторі.

5.3.3.3 Індекси GIN

Узагальнені інвертовані індекси GIN (Generalized Inverted Index) представляють собою інвертовані індекси.

Індекси GIN працюють з типами даних, значення яких не є атомарними, а складаються з окремих елементів. При цьому індексуються не самі значення, а їх окремі елементи. Кожен елемент посилається на ті значення, в яких він зустрічається. Прикладом таких значень служать масиви.

Інвертований індекс може ефективно працювати в запитах, які перевіряють присутність елемента в неатомарних значеннях полів таблиці. Ці індекси можуть підтримувати різні, визначені користувачем, стратегії і в залежності від них можуть застосовуватися з різними операторами.

Індекси GIN зберігаються у вигляді B-дерева. До кожного елементу цього дерева прив'язаний упорядкований набір посилань на рядки таблиці, які містять значення з цим елементом.

Створити GIN-індекс можна за допомогою такої команди:

```
CREATE INDEX імя_індекса ON ім'я_таблиці
        USING GIN (ім'я стовпця)
```

Стандартний дистрибутив PostgreSQL включає клас операторів GIN для масивів як значень полів таблиці.

Як уже відмічалося, індекси GiST і GIN широко використовуються при повнотекстовому пошуку PostgreSQL. Це означає, що для кожного вектору tsvector, що описує всі лексеми документа, створюється сигнатура, що описує, які з лексем входять в даний tsvector.

Основна область застосування індексів GIN - це прискорення повнотекстового пошуку. Для цього в структурі індексів з кожної лексемою зіставляється відсортований список номерів документів, в яких вона зустрічається. Пошук по такій структурі набагато ефективніше, ніж при використанні GiST, однак процес додавання нового документа досить тривалий.

Ще одним прикладом комплексного типу даних, для якого є вбудована GIN-підтримка, є значення в форматі JSON. Для роботи зі значеннями JSON в даний час визначено ряд операторів і функцій, частина з яких може бути прискорена за допомогою індексів:

5.3.4 Індекси BRIN

Індекси зон блоків BRIN (Block Range Indexes) з'явилися, починаючи з PostgreSQL 9.5, і зберігають узагальнені відомості про значення, що знаходяться в *фізично послідовно розташованих блоках таблиці*.

Індекси BRIN можуть підтримувати певні визначені користувачем стратегії, і в залежності від них застосовуватися з різними операторами. Для типів даних, що мають лінійний порядок сортування, записам в індексі відповідають мінімальні і максимальні значення даних в стовпчику для кожної зони блоків.

Індекси BRIN призначені для обробки великих таблиць, в яких значення індексованого стовпчика має деяку природну кореляцію з фізичним розташуванням рядка в таблиці, тобто значення в стовпчику мають лінійний порядок сортування. Наприклад, це характерно для зберігання замовлень магазину, які пишуться послідовно, а тому вже на фізичному рівні впорядковані за датою / номером, і в той же час такі таблиці з такими даними зазвичай розростають-

ся до гігантських розмірів. Аналогічно це характерно для зберігання квитанцій на оплату комунальних послуг тощо.

Завдяки своїй природі індекс BRIN має дуже малий розмір. В результаті сканування індексу додає невеликі накладні витрати тому Індекси BRIN володіють такими якостями таблиць, як швидка вставка рядка і швидке створення індексу.

Створити індекс BRIN можна за допомогою такої команди:

```
CREATE INDEX імя_індекса ON ім'я_таблиці
USING BRIN (ім'я стовпця)
```

5.4 Складені індекси

Якщо пошук йде по декількох стовпчиках таблиці, то можна створити складений індекс за цими стовпчиками.

Створити індекс за кількома стовпчиками таблиці можна за допомогою наступної команди:

```
CREATE INDEX імя_індекса ON ім'я_таблиці (ім'я стовпця1,
                ім'я стовпця2, ...)
```

В даний час складеними можуть бути тільки індекси типів B-дерево, GiST, GIN і BRIN. Кількість стовпців в індексі обмежується 32.

6 ПОВНОТЕКСТОВИЙ ПОШУК

6.1 Повнотекстовий пошук

Повнотекстовий пошук або пошук тексту - це можливість знаходити документи на природній мові, відповідні запиту, і, можливо, додатково сортувати їх за релевантністю для цього запиту.

Документ - це одиниця обробки в системі повнотекстового пошуку; наприклад, технічна стаття або звіт. Система пошуку тексту повинна вміти розбирати документи і зберігати зв'язку лексем (ключових слів) з документом, який їх містить. Згодом ці зв'язки можуть використовуватися для пошуку документів з заданими ключовими словами.

Під *релевантністю (Relevance)* розуміють ступінь відповідності знайденого документа або набору документів інформаційним потребам користувача.

Найбільш поширене завдання повнотекстового пошуку - знайти всі документи, що містять задані слова запиту, і видати ці документи, відсортованими за ступенем відповідності запиту.

6.2 Засоби повнотекстового пошуку PostgreSQL

Історично реляційні БД використовувалися для представлення економічних, технічних і подібних до них даних, тобто даних, які мають переважно табличний вигляд. Однак, в подальшому, в зв'язку з їх широким розповсюдженням, реляційні БД стали використовувати і для подання текстових документів. І хоча на сьогодні з'явилися документо-орієнтовані БД, наприклад MongoDB, реляційні БД розвинули засоби, які в багатьох випадках дозволяють обійтися без залучення документо-орієнтованих БД. Додаткові можливості для цього виникли в зв'язку з появою текстового формату обміну даними JSON (JavaScript Object Notation).

PostgreSQL (pgSQL) має розвинені засоби повнотекстового пошуку див. Глава 12, підрозділи 12.1-12.3 документації по PostgreSQL 9.4 - 12, наприклад

https://postgrespro.ru/docs/postgresql/11/textsearch або
https://postgrespro.com/docs/postgresql/11/textsearch

У концепції pgSQL *документ* - це зазвичай вміст текстового поля рядка таблиці або поєднання (об'єднання) значень полів, які можуть зберігатися в одній або різних таблицях або формуватися динамічно. Наприклад, таким поєднання можна отримати, склеївши поля StName || TchName || SbjName 3-х відповідних таблиць. Таким чином, документ для індексації може створюватися з декількох частин і не зберігатися де-небудь як єдине ціле.

Слова природної мови крім власне кореня містять приставки, суфікси, закінчення та інше. Функція to_tsvector все це вирізає і перетворює текст в набір токенів (token), по яким і виконується пошук. В результаті кожен документ зводиться до його спеціального формату tsvector, який містить компактне представлення всього документу. Пошук і ранжування виконується виключно з цим представленням документу - вхідний текст буде потрібно витягти, тільки коли документ буде відібраний для виведення користувачеві.

Функція to_tsvector має вигляд

to_tsvector('мова', 'початковий текст')

і перетворює (нормалізує) заданий текст у значення tsvector, наприклад

SELECT to_tsvector('ukraine', 'Здав екзамен з програмування на відмінно')

отримаємо

'''відмінн':6 'програмуванн':4 'здав':1 'екзам':2'"

Приведений до спеціального формату запит на повнотекстовий пошук зберігається в структурі tsquery. Значення типу tsquery містить шукані токени, можливо об'єднані в вирази операторами AND (**&**), OR (**|**), NOT (**!**), <-> тощо.

Оператор AND вимагає, щоб всі перераховані токени входили у знайдений текст, незалежно від їх порядку слідування.

Оператор OR вимагає, щоб хоч один з перерахованих токенів входив у знайдений текст.

Оператор NOT вимагає, щоб вказаний токен не входив в знайдений текст.

Оператор <-> вимагає, щоб всі перераховані токени як цілісний вираз входили у знайдений текст.

Функції to_tsquery() та plainto_tsquery() мають вигляд

to_tsquery('мова', 'шуканий фрагмент [&'шуканий фрагмент2 …]')

to_tsquery('мова', 'шуканий фрагмент')

і працюють аналогічно функції to_tsvector() та перетворюють-нормалізують заданий користувачем текст запиту в значення tsquery, наприклад

SELECT to_tsquery ('ukraine', 'екзамен & з & програмування')

Отримаємо

'''екзам' & 'програмуванн'"

Різниця між функціями to_tsquery() та plainto_tsquery() в наступному.

Функція plainto_tsquery() використовується, якщо необхідно гарантувати, що жодні вхідні дані не можуть порушити SQL-запит. Це стається за рахунок обов’язкових пошуків.

Функція to_tsquery() використовується, якщо необхідно підтримувати складний пошук з ризиком можливого невірного SQL-запиту.

По суті функція plainto_tsquery() екранує всі символи і розміщує оператор AND між всіма токенами. Якщо використовується звичайний запит, оператори AND, NOT и <-> и не можна використовувати в пошуковому запиті, що обмежує гнучкість пошуку. Усі пошукові запити повинні містити всі передбачені токени.

Функція to_tsquery() дозволяє використовувати всі вказані умови пошуку, але якщо ви використовуєте to_tsquery(), користувачі можуть надсилати невірні дані, які спричинять помилку SQL-запиту, наприклад, шуканий фрагмент має вид '& term'. Цей початковий оператор & призведе до помилки SQL-запиту, якщо він буде надісланий безпосередньо до запиту, оскільки немає токенів перед оператором &.

6.3 Пошук простої відповідності

Повнотекстовий пошук в pgSQL реалізується на базі оператора відповідності @@, який повертає true, якщо документ tsvector відповідає запиту tsquery. Для цього оператора не важливо, який тип tsvector або tsquery записаний першим або другим.

Таким чином, запит з пошуком відповідності виглядає так

```
SELECT to_tsvector('мова', 'початковий текст') @@
          to_tsquery('мова', 'фрагмент [& фрагмент2 …]')
```

Наприклад, за запитом

```
SELECT to_tsvector('ukraine', 'Здав екзамени з програмування
                    на відмінно, вивчав англійську мову …') @@
          to_tsquery('ukraine', 'екзамен & відмінно ')
```

отримаємо true

6.4 Конфігурування пошуку відповідності

Вище розглянуто простий приклад повнотекстового пошуку. Реалізація повнотекстового пошуку pgSQL дозволяє робити крім того наступне:

а) пропускати певні слова (стоп-слова);
б) обробляти синоніми;
в) виконувати складний аналіз слів.

Ці функції управляються конфігураціями текстового пошуку. Зокрема, в попередніх прикладах використовувалася конфігурація мови пошуку - 'russian'. У разі англійської мови конфігурацію можна не вказувати, вона йде за замовчуванням.

У pgSQL є набір зумовлених конфігурацій для багатьох природніх мов, але можна створювати власні конфігурації. (Всі доступні конфігурації можна переглянути за допомогою команди \ dF).

Відповідна конфігурація для даного середовища вибирається під час установки і записується в параметрі default_text_search_config в postgresql.conf.

Більш детально питання конфігурації розглянуті в документації.

6.5 Пошук в полі таблиці

Повнотекстовий пошук можна виконати, безпосередньо в полі таблиці. Наступний запит виводить імена студентів, в текстовому полі резюме StResume яких є слова "фізика" і "відмінно":

```
      SELECT StName
      FROM Students
WHERE to_tsvector('ukraine', StResume) @@
         to_tsquery('ukraine', 'фізика & відмінно')
```

Отримаємо

Студент 1

Студент 2 …

Для прискорення текстового пошуку створюються індекси GIN або GiST по відповідному полю (див. Документацію).

6.6 Ранжирування результатів пошуку

Ранжирування документів можна розглядати як спосіб оцінки - наскільки вони релевантні заданому запиту, і впорядкувати їх так, щоб найбільш релевантні виводилися першими. У pgSQL вбудовані дві функції ранжирування ts_rank() і ts_rank_cd(), які беруть до уваги лексичну, позиційну і структурну інформацію; тобто, вони враховують, наскільки часто і наскільки близько зустрічаються в документі ключові слова і яка важливість частини документа, яка їх містить. Однак саме поняття релевантності вельми нечітке і багато в чому визначається додатком. Вбудовані функції ранжирування можна розглядати лише як приклади реалізації. Для своїх конкретних завдань користувач може розробити власні функції ранжирування і/або врахувати при обробці їх результатів додаткові чинники.

6.7 Виділення результатів пошуку

Важливою вимогою до результату пошуку є виділення тієї частини або частин документа, які відповідають запиту. Зазвичай пошукові системи показують фрагменти документу із зазначеними шуканими словами. У pgSQL для реалізації цієї можливості використовується функція ts_headline().

ts_headline([*конфігурація* regconfig,] *документ* text, *запит* tsquery
[, *параметри* text])

Функція ts_headline() приймає документ разом із запитом і повертає витяг з документа, в якій виділяються слова з запиту. Конфігурацію, що застосовується для розбору документа, можна вказати в параметрі config; якщо цей параметр опущений, застосовується конфігурація за замовчуванням

default_text_search_config.

Якщо в параметрах передається рядок options, він повинен складатися зі списку розділених комами пар параметр = значення. Параметри можуть бути наступними:

- ✓ StartSel, StopSel - рядки, які будуть розмежовувати слова запиту в документі, виділяючи їх серед інших. Якщо ці рядки містять прогалини чи коми, їх потрібно взяти в лапки;
- ✓ MaxWords, MinWords - числа, які визначають верхню і нижню межі розміру вибірки;
- ✓ ShortWord - слова зазначеної довжини (або коротше) на початку і в кінці витягу будуть відкидатися. Значення за замовчуванням, рівне три, виключає поширені англійські артиклі;
- ✓ HighlightAll - логічний прапорець; якщо він дорівнює true, витягом буде весь документ і 3 попередні параметра ігноруються;

✓ MaxFragments - максимальне число виведених текстових витягів або фрагментів. Значення за замовчуванням, рівне 0, вибирає метод створення витримки без фрагментів. При значенні більшому 0 вибирається метод з фрагментами, коли знаходяться всі фрагменти, що містять якомога більше слів запиту, а потім вони стискаються до слів запиту. Такі фрагменти можуть містити якісь ключові слова в середині і обмежуються двома шуканими словами. При цьому фрагменти можуть містити не більше MaxWords слів, а на початку і в кінці вони будуть очищені від слів довжини ShortWord і менше. Якщо в документі знайдені не всі слова запиту, виводиться один фрагмент, що включає перші MinWords слів в документі;

✓ FragmentDelimiter - якщо виводяться кілька фрагментів, вони будуть розділятися цим рядком.

Всі явно не задані параметри отримують такі значення по замовчуванню:

```
StartSel=<b>, StopSel=</b>,
MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE,
MaxFragments=0, FragmentDelimiter=" ... "
```

Приклад використання:

```
SELECT ts_headline ('ukraine', 'Здав екзамени з програмування на відмінно, а зі
    схемотехніки на добре', to_tsquery('ukraine', 'екзамен & відмінно'))
```

Отримаємо шукані слова, виділені парами тегів и

```
"Здав <b>екзамени</b> з програмування на <b>відмінно</b>, а зі
схемотехніки на добре"
```

Увага! Функція ts_headline() працює з оригінальним документом, а не з його стисненим поданням tsvector, так що вона може бути повільною і використовувати її слід обачно.

6.8 Індекси і прискорення повнотекстового пошуку

Якщо в якійсь колонці пошук виконується регулярно, зазвичай її бажано індексувати.

Для прискорення повнотекстового пошуку можна використовувати індекси 2-х видів:

1. Узагальнена дерево пошуку GiST (Generalized Search Tree);
2. Узагальнений Інвертований Індекс GIN (Generalized Inverted Index).

6.8.1 Створення та використання індексів текстових полів

Наступна команда створює індекс на базі GiST. Тут колонка може мати тип tsvector або tsquery

CREATE INDEX ***імя_індекса*** **ON** ***ім'я_таблиці*** **USING** GIST ***(колонка)***

Приклад створення індексу GiST

```
CREATE INDEX StResumIdx ON Students
```

USING GIST (to_tsvector ('ukraine', StResume));

Наступна команда створює індекс на базі GIN. Тут колонка повинна мати тип tsvector

CREATE INDEX ***імя_індекса*** **ON** ***ім'я_таблиці*** **USING** GIN ***(колонка)***

Приклад створення індексу GIN

CREATE INDEX StResumIdx_1 ON Students

USING GIN (to_tsvector ('ukraine', StResume));

Увага! Так як при створенні індексу використовувалася версія to_tsvector() з 2-ма аргументами, цей індекс буде використовуватися тільки в за-просив, де to_tsvector () викликається також з 2-ма аргументами і в першому пере-дається ім'я тієї ж конфігурації 'russian' .

SELECT StName

FROM Students

WHERE to_tsvector ('ukraine', StResume) @@

to_tsquery ('ukraine', 'фізика & відмінно')

У розглянутому вище прикладі конфігурація russian відноситься до значень всіх полів стовпця StResume. Цікавим є питання, а що якщо вміст окремих полів стовпця StResume представлено на різних мовах, тобто має різну конфігурацію - russian, english тощо. В цьому випадку в таблиці створюється додатковий стовпець текстового типу, наприклад StConfigTSvect, в якому розміщується конфігурація для кожного поля стовпця StResume. Тепер індекс можна створити наступним чином

CREATE INDEX StResumIdx_2 **ON** Students

USING GIN (to_tsvector (StConfigTSvect, StResume));

В результаті в одному індексі будуть елементи з різними конфігураціями. Запит в цьому випадку повинен використовуватися той же індекс, тобто індекс з таким же чином задається конфігурацією

SELECT StName

FROM Students

WHERE to_tsvector (StConfigTSvect, StResume) @@

to_tsquery (StConfigTSvect, 'фізика & відмінно')

Інший варіант створення індексів GiST або GIN полягає в наступному. У таблиці Students створюється стовпець типу tsvector, наприклад StResumTSvect, в якому розміщується tsvector для кожного значення стовпчика StResume. Потім слід сформувати значення поля StResumTSvect, після чого створити необхідний GiST-індекс

UPDATE Students **SET** StResumTSvect =

to_tsvector ('ukraine', StResume);

CREATE INDEX StResumIdx_3 **ON** Students

USING GIST(StResumTSvect);

Тепер можна швидко виконувати повнотекстовий пошук:

```
SELECT StName
FROM Students
WHERE StResumTSvect @@ to_tsquery ('ukraine', 'фізика & відмінно')
```

Коли вистава tsvector зберігається в окремому стовпці, необхідно створити тригер, який буде підтримувати стовпець з tsvector в актуальному стані при будь-яких змінах стовпців StName або StResume.

6.8.2 Порівняння способів індексації GiST і GIN

Індекси GiST і GIN значно розрізняються за швидкодією.

Індекс GiST допускає помилкові потрапляння і тому їх потрібно виключати додатково, звіряючи результат з фактичними даними таблиці. Імовірність помилкових влучень залежить від ряду факторів, наприклад від кількості унікальних слів.

Неточність індексу призводить до зниження продуктивності через довиконавчими звернень до записів таблиці, для яких припущення про збіг виявляється хибним. Так як довільний доступ до таблиці зазвичай не буває швидким, це обмежує застосовність індексів GiST.

Індекси GIN не є точними для стандартних запитів, але їх продуктивність логарифмічно залежить від числа унікальних слів. Слід зазначити, що індекси GIN зберігають тільки слова (лексеми) значень tsvector, але втрачають інформацію про їх ваги. Таким чином, для виконання запиту з вагами буде потрібно перевірити ще раз дані в таблиці.

Вибираючи між індексами GiST і GIN, необхідно врахувати такі їх відмінності з точки зору продуктивності:

- ✓ пошук за індексом GIN приблизно втричі швидше, ніж по GiST;
- ✓ індекси GIN будуються приблизно втричі довше, ніж GiST;
- ✓ індекси GIN оновлюються кілька повільніше, ніж GiST, але якщо відключено швидке оновлення різниця може досягати 10 разів;
- ✓ індекси GIN зазвичай в два-три рази більше індексів GiST.

Як правило, індекси GIN краще підходять для статичних даних, так як пошук з ними виконується швидше.

Для динамічних даних краще використовувати індекси GiST, так як вони швидше оновлюються. Точніше, індекси GiST ефективні для динамічних даних і працюють швидко, якщо число унікальних слів (лексем) не перевищує 100 000, а індекси GIN краще справляються з великою кількістю лексем, але оновлюватися будуть повільніше.

Є два напрями збільшення швидкості пошуку з можливістю поновлення "на льоту":

- ✓ секціонувати великі колекції документів;
- ✓ ефективно застосовувати індекси GiST і GIN.

Секціонувати дані можна як на рівні БД, з використанням наслідування таблиць, так і розподіливши документи по різних серверах і потім збирати ре-

зультати. Останній варіант можливий завдяки тому, що функції ранжирування використовують тільки локальну інформацію.

7 ПРАВА ДОСТУПУ

У PostgreSQL поняття користувач USER та роль ROLE є синонімами. pgAdmin III для управління доступом до БД використовується поняття роль.

У версії pgAdmin III.x права доступу можна надавати для таблиці БД, представлення, на виконання серверних процедур тощо [9]. У версії pgAdmin IV.x права доступу можна надавати ще й до окремих стовпців таблиці.

Процес призначення прав доступу включає 2-ва етапи:

1. Створення ролей;
2. Надання створеним ролям прав доступу.

Нижче наводиться опис виконання кожного етапу засобами pgAdmin III.

7.1 Створення ролей

Щоб створити нову роль необхідно у вікні Браузер об'єктів виокремити розділ Ролі входу, активізувати його контекстне меню і в ньому вибрати пункт Нове правило (Рисунок 7.1).

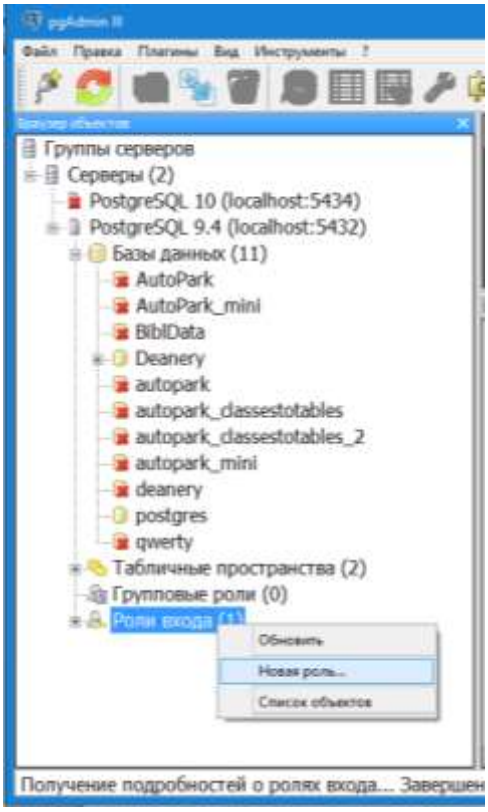


Рисунок 7.1 – Контекстне меню Ролі входу

У вікні Нове правило (Рисунок 7.2) вводиться ім'я ролі, пароль, і, якщо необхідно, дата і час закінчення дії облікового запису (аккаунта).

Для прикладу БД Deanery користувачами можуть бути декан і секретар. Для них необхідно створити відповідні ролі входу: dean і secretary. Після чого натиснути ОК.

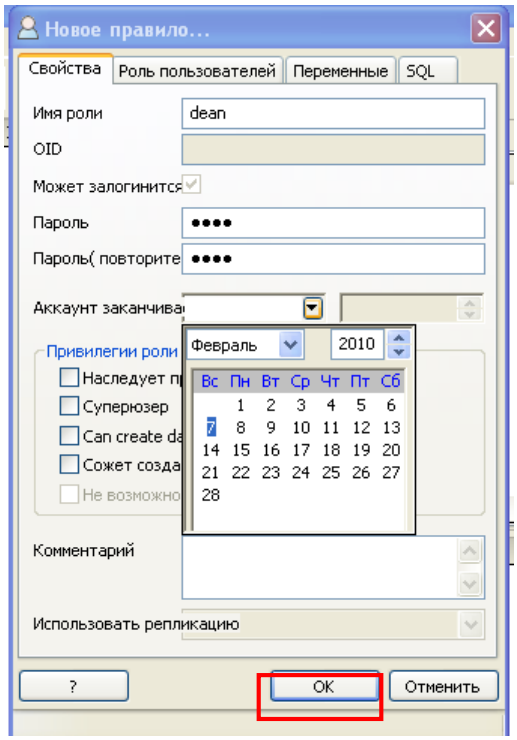


Рисунок 7.2 – Вікно створення нової ролі входу. Закладка Властивості

7.2 Призначення прав доступу

Призначити права новим ролям можна за допомогою користувацьких SQL-запитів. Нижче наведено опис способу створення SQL-запитів.

1. Вибрати будь-який об'єкт відповідної БД в вікні Браузеру об'єктів і натиснути кнопку Виконати користувацькі SQL запити (Рисунок 7.3);

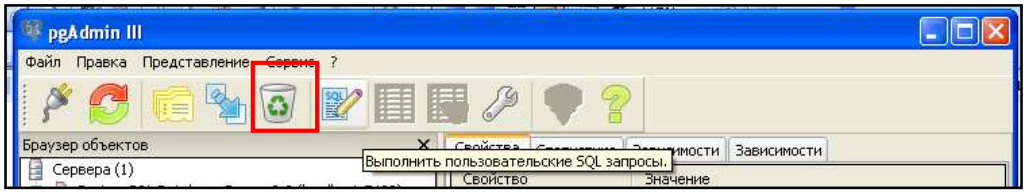


Рисунок 7.3 – Виклик вікна призначених для користувача запитів

2. У вікні Query ввести запити на надання прав доступу ролям БД.
3. Натиснути кнопку Виконати запит (Рисунок 7.4);

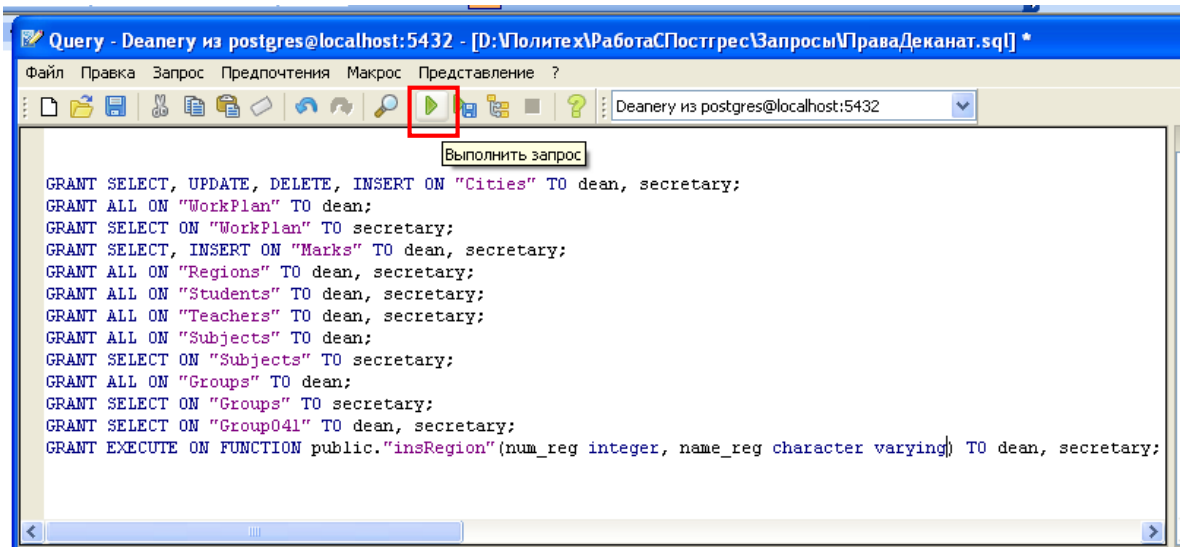


Рисунок 7.4 – Вікно введення користувацьких SQL-запитів Query

4. Якщо необхідно, виконати збереження запиту, натиснувши кнопку Зберегти файл вікна Query (Рисунок 7.5).

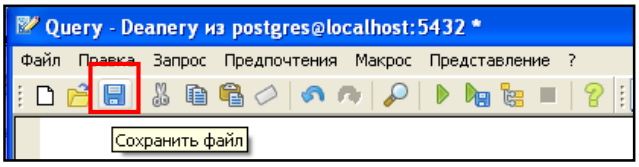


Рисунок 7.5 – Збереження користувацького SQL-запиту

7.3 Відміна прав доступу

Скасування прав доступу виконується аналогічним чином через вікно введення запитів користувача SQL.

7.4 Перевірка прав доступу

Перевірити права можна, наприклад, за допомогою утиліти SQL Explorer з середовища Delphi.

Попередня умова: Драйвер PostgreSQL ODBC повинен бути інстальований і DSN користувача або Системний DSN повинен бути створений (див. Пункт 8.1). Повинна бути створена БД "Деканат Вузу" і її псевдонім.

Нижче наведено опис способу перевірки прав доступу користувача Секретар за допомогою утиліти SQL Explorer.

1. За допомогою команди Database->Explorer з середовища Delphi запустіть утиліту SQL Explorer. Отримайте знайоме вікно (Рисунок 7.6)

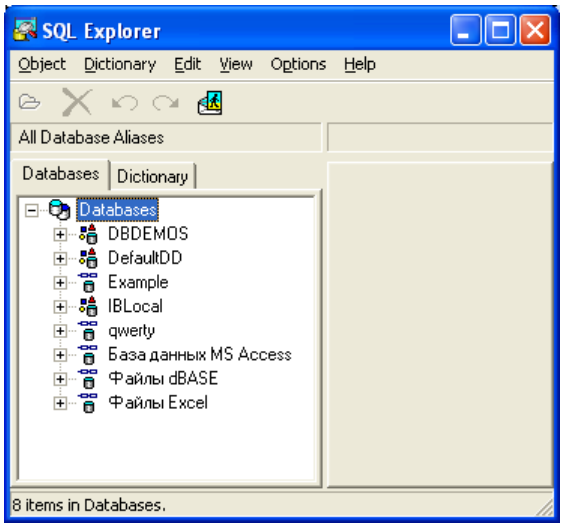


Рисунок 7.6 – SQL Explorer

2. На вкладці Databases вікна виберіть попередньо створений псевдонім до БД (для прикладу БД "Deanery" цей псевдонім називається Example). Клацніть на ньому 2-ва рази лівою кнопкою миші. В результаті відкриється вікно Database Login, в якому необхідно ввести ім'я користувача (ролі) і пароль (Рисунок 7.7).

3. Натисніть ОК

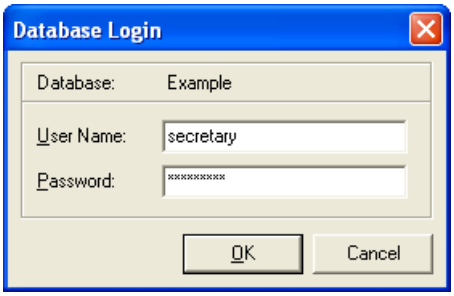


Рисунок 7.7 – Вікно Database Login

В результаті користувач secretary отримає доступ до БД (Рисунок 7.8).

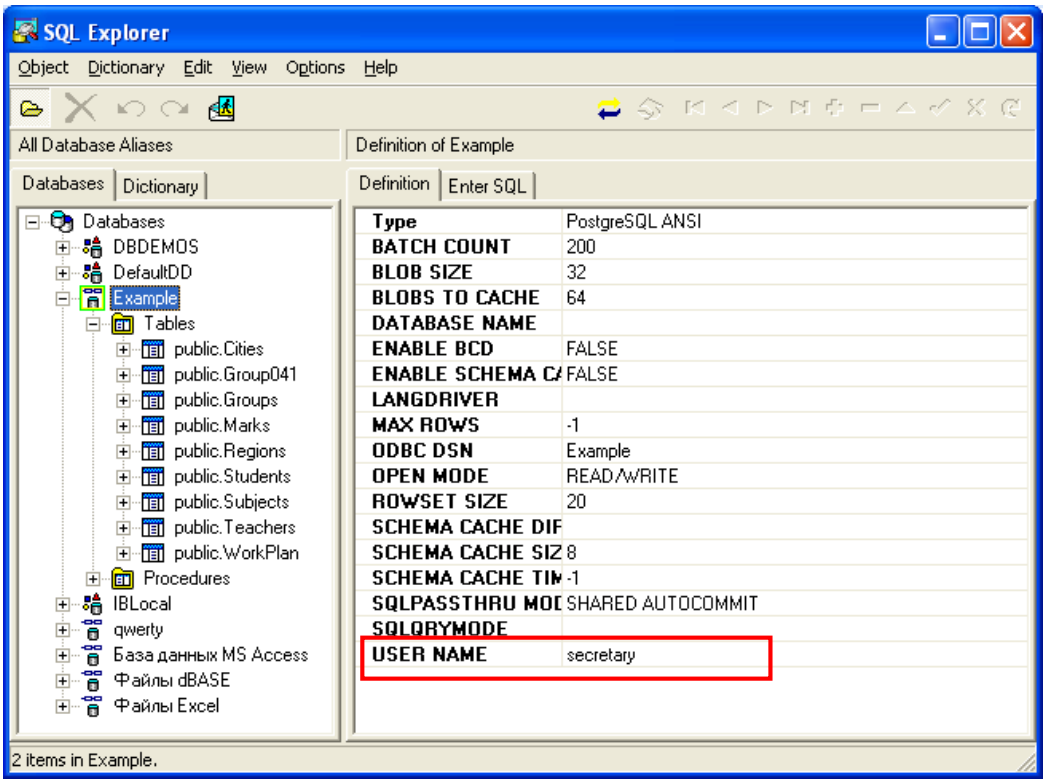


Рисунок 7.8 – SQL Explorer. Користувач/Роль secretary

Якщо даний користувач/роль спробує виконати запит на вставку або оновлення запису в таблиці Groups (закладка Enter SQL), то отримає повідомлення такого вигляду (Рисунок 7.9):

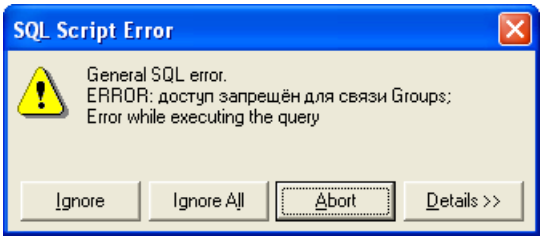


Рисунок 7.9 – Інформаційне повідомлення

Запит на вибірку даних з таблиці Groups для даного користувача буде виконаний нормально (Рисунок 7.10)

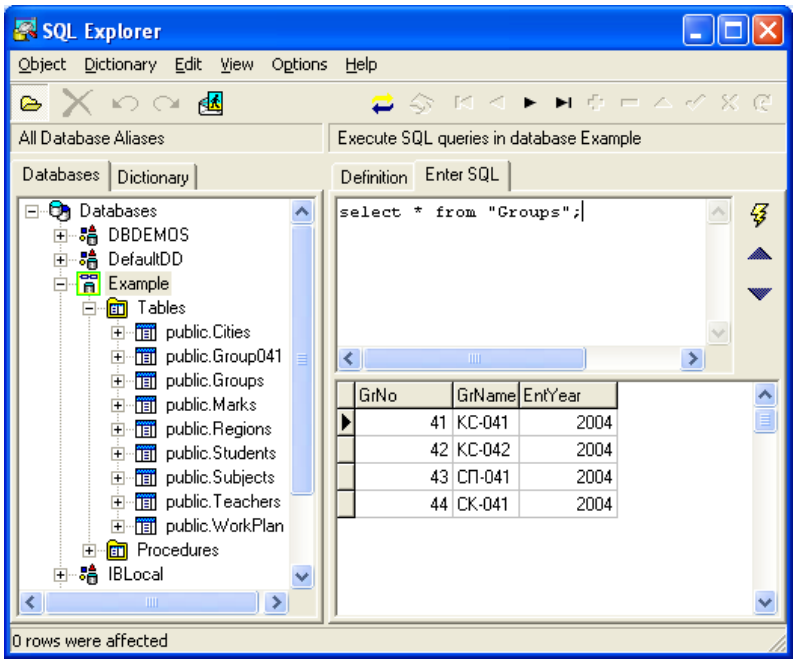


Рисунок 7.10 – Вибірка даних з таблиці Groups

8 ІЗОЛЯЦІЯ ТРАНЗАКЦІЙ

8.1 Проблеми неузгодженості даних при виконанні паралельних транзакцій

При паралельному виконанні групи транзакцій можлива ситуація, коли одночасно декілька з цих транзакцій намагаються змінити один і той же рядок таблиці, а інші в цей час намагаються читати той же рядок. Відповідно можлива ситуація, коли різні транзакції можуть отримати різні дані з одного й того ж рядка.

При паралельному виконанні транзакцій можливі наступні проблеми неузгодженості даних:

1. *Втрачене оновлення (Lost Update)* - при одночасній зміні одного блоку даних різними транзакціями втрачаються всі зміни, крім останньої;
2. *"Брудне" читання (Dirty Read)* - читання даних, доданих або змінених транзакцією, яка ще не завершена, а згодом не підтвердиться (відкотиться);
3. *Читання, що не повторюється (Non-Repeatable Read)* - при повторному читанні в рамках однієї транзакції раніше прочитані дані виявляються зміненими, тому що між читаннями інша транзакція змінила дані;
4. *Фантомне читання (Phantom Reads)* - одна транзакція в ході свого виконання кілька разів вибирає декілька рядків по одним і тим же критеріям. Інша транзакція в інтервалах між цими вибірками додає або видаляє рядки або змінює поля деяких рядків, які використовуються в умовах вибірки першої транзакції, і успішно закінчується. В результаті вийде, що одні й ті ж вибірки в першій транзакції дають різну кількість рядків;
5. *Аномалія серіалізації (Serializable anomaly*) - результат успішної фіксації групи транзакцій виявляється неузгодженим при різних варіантах виконання цих транзакцій по черзі.

8.2 Рівні ізоляції транзакцій

Під *рівнем ізоляції транзакцій* розуміється ступінь забезпечення внутрішніми механізмами СУБД (без спеціального програмування) захисту від усіх або деяких перелічених видів неузгодженостей даних, що виникають при паралельному виконанні транзакцій.

Стандарт SQL-92 визначає шкалу з 4-х рівнів ізоляції транзакцій: Read uncommitted, Read committed, Repeatable read, Serializable. Перший з них є найслабшим, останній - найсильнішим, а кожний наступний проміжний включає в себе всі попередні:

- ✓ *Read uncommitted - читання незафіксованих даних*, гарантує тільки відсутність втрачених оновлень. Якщо кілька паралельних транзакцій намагаються змінювати один і той же рядок таблиці, то в остаточному варіанті рядок буде мати значення, визначене останньою успішно виконаною транзакцією. *Увага*, при цьому можливо зчитування не тільки логічно неузгоджених даних, але і даних, зміни яких ще не зафіксовані.
- ✓ *Read committed - читання фіксованих даних*. Більшість промислових СУБД, зокрема, PostgreSQL за замовчуванням використовують цей рівень. На цьому рівні забезпечується захист від "брудного" читання, але в процесі

роботи однієї транзакції інша може бути успішно завершена і зроблені нею зміни зафіксовані. У підсумку перша транзакція буде працювати з іншим набором даних, а це вже проблема неповторюваного читання.

✓ *Repeatable read (повторюваність читання)*. Читання одного і того ж рядку чи групи рядків в транзакції дає один і той же результат. Тобто поки транзакція не закінчена, ніякі інші транзакції не можуть змінити ці дані.

✓ *Serializable (впорядкованість)*. Транзакції повністю ізолюються одна від одної. На цьому рівні результати паралельного виконання транзакцій для БД у більшості випадків можна вважати такими, що збігаються з послідовним виконанням тих же транзакцій по черзі в будь-якому порядку.

Узагальнення рівнів ізоляції транзакцій наведено в Таблиця 8.1

Таблиця 8.1 – Співвідношення рівнів ізоляції транзакцій

Рівень ізоляції	Фантомне читання	Неповторюване читання	«Брудне» читання	Втрачене обновлення
Serializable	+	+	+	+
Repeatable read	-	+	+	+
Read committed	-	-	+	+
Read uncommitted	-	-	-	+

8.3 Ізоляція транзакцій в PostgreSQL

В PostgreSQL можна заявити будь-який з 4-х рівнів ізоляції транзакцій, однак реалізовані тільки три з них – Serializable, Repeatable read, Read committed. Режим Read Uncommitted діє як Read Committed. Це викликано тим, що в PostgreSQL реалізовано архітектуру багатоверсійного управління конкурентним доступом.

Для вибору потрібного рівня ізоляції транзакцій використовується команда set transaction. Виділяють три варіанти цієї команди. Їх синтаксис має наступний вигляд

SET TRANSACTION *режим_транзакції* [, ...]

SET TRANSACTION SNAPSHOT *id_знімку*

SET SESSION CHARACTERISTICS AS TRANSACTION
режим_транзакції [, ...]

де *режим_транзакції* може бути наступним:

Isolation Level { Serializable | Repeatable read | Read committed |
Read uncommitted }

Read Write | Read only

[Not] Deferrable

Команда Set transaction встановлює характеристики поточної транзакції. На наступні транзакції вона не впливає.

Команда Set session characteristics встановлює характеристики транзакції за замовчуванням для наступних транзакцій в рамках сеансу. Задані за замовчу-

ванням характеристики транзакцій потім можна перевизначити для окремих транзакцій командою Set transaction.

До характеристик транзакції належить:

1. Рівень ізоляції транзакції (Isolation level);
2. Режим доступу транзакції - читання/запис (Read Write) або тільки читання (Read only);
3. Допустимість/недопустимість відкладання транзакції ([Not] Deferrable).

На додаток до цих характеристик можна вибрати знімок (Snapshot), але тільки для поточної транзакції, а не для сеансу за умовчанням.

Рівень ізоляції транзакції Isolation Level визначає, які дані може бачити транзакція, коли паралельно з нею виконуються інші транзакції:

- ✓ Read committed - оператор транзакції бачить тільки ті рядки, які були зафіксовані до початку його виконання. Цей рівень встановлюється за умовчанням;
- ✓ Repeatable read - всі оператори поточної транзакції бачать тільки ті рядки, які були зафіксовані перед першим запитом на вибірку або зміну даних, виконаним в цій транзакції;
- ✓ Serializable - всі оператори поточної транзакції бачать тільки ті рядки, які були зафіксовані перед першим запитом на вибірку або зміну даних, виконаним в цій транзакції. Якщо накладення операцій читання і запису паралельних Serializable транзакцій може призвести до ситуації, неможливої при послідовному їх виконанні, коли транзакції виконуються одна за одною, відбудеться відкат однієї з транзакцій з помилкою збій серіалізації serialization_failure.

Рівень ізоляції транзакції (Isolation level) не можна змінити в поточній транзакції після виконання першого запиту на вибірку або заміну даних: Select, Insert, Delete, Update, Fetch або Copy.

Режим доступу транзакції визначає, чи буде транзакція тільки читати дані (Read only) або буде і читати, і писати (Read Write) дані. За замовчуванням мається на увазі читання/запис (Read Write). У транзакції без запису (Read only) забороняються наступні команди: Insert, Update, Delete і Copy From, якщо тільки цільова таблиця не тимчасова. Також забороняються будь-які команди Create, Alter і Drop, а також Comment, Grant, Revoke, Truncate. Крім того, забороняються Explain Analyze і Execute, якщо команда, яку вони повинні виконати, відноситься до перерахованих вище.

Допустимість (Deferrable) впливає, тільки якщо транзакція також знаходиться в режимах Serializable і Read Only. Коли для транзакції встановлені всі ці 3-ри властивості, транзакція може бути заблокована при першій спробі отримати свій знімок даних, після чого вона зможе виконуватися без додаткових зусиль, звичайних для режиму Serializable, і без ризику привести до збою серіалізації або постраждати від нього. Цей режим підходить для тривалих операцій, наприклад для побудови звітів або резервного копіювання.

Команда Set Transaction Snapshot дозволяє виконати нову транзакцію зі знімком даних, який має вже існуюча транзакція. Ця раніше створена транзакція повинна експортувати цей знімок за допомогою функції pg_export_snapshot. Ця функція повертає ідентифікатор знімка, який і потрібно передати команді Set

Transaction Snapshot в якості ідентифікатора імпортованого знімка. Set Transaction Snapshot можна виконати тільки на початку транзакції, до першого запиту на вибірку або зміну даних (Select, Insert, Delete, Update, Fetch або Copy) в поточної транзакції. Більш того, для транзакції вже повинен бути встановлений рівень ізоляції Serializable або Repeatable Read. В іншому випадку знімок буде відразу ж втрачено, так як на рівні Read Committed для кожної команди робиться новий знімок. Якщо транзакція, що імпортує, працює на рівні ізоляції Serializable, то транзакція, що експортує знімок, також повинна працювати на цьому рівні. Крім того, транзакції в режимі читання/запис не можуть імпортувати знімок з транзакції в режимі "Read only".

8.4 Ізоляція транзакцій в сучасних системах програмування

В чистому вигляді команда set transaction використовувалась в процедурних мовах програмування. Сучасні системи програмування приховують цю команду в більш абстрактних конструкціях типу Session. Тому налаштування рівнів ізоляції транзакцій виконуються засобами відповідних середовищ програмування.

9 СТВОРЕННЯ ТА ВИКОНАННЯ СЕРВЕРНИХ ПРОЦЕДУР

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

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

Для створення подібної процедури можна використовувати два підходи:

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

2. Використати майстер створення функції.

9.1 Формування тексту процедури у вікні SQL-запитів користувача

Натиснути кнопку "Виконати SQL-запити користувача" (Рисунок 9.1)

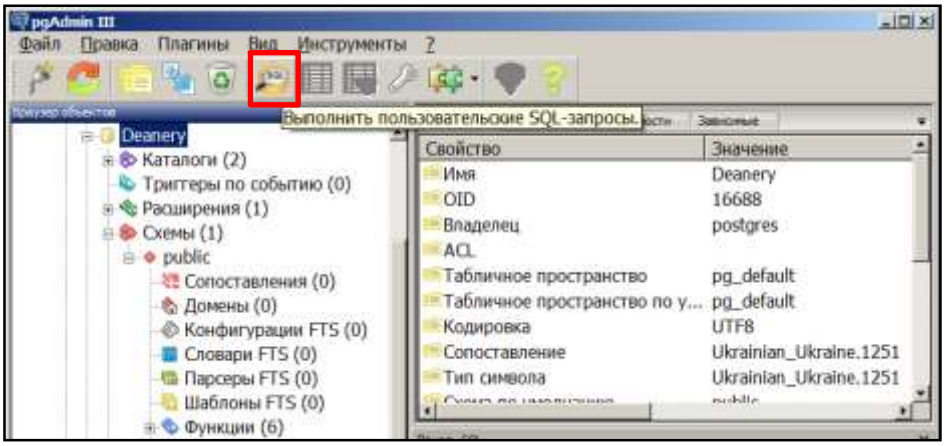


Рисунок 9.1 – Кнопка Виконати SQL-запити

У вікні записати текст функції, яка, наприклад, по назві групи повертає список її студентів (Рисунок 9.2).

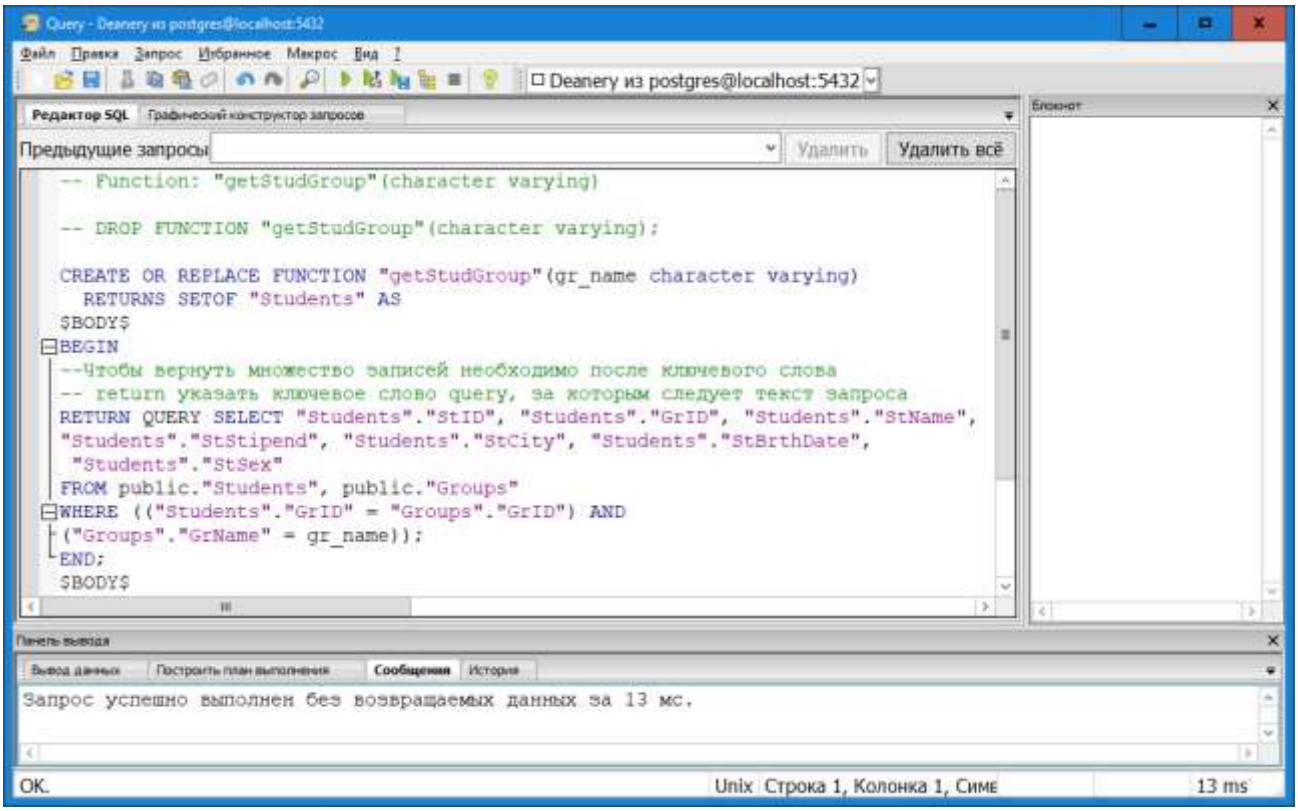


Рисунок 9.2 - Текст функції, яка по назві групи повертає список її студентів

9.2 Використання майстра створення серверної процедури

У контекстному меню елементу Функції вибрати пункт "Нова функція ..." (Рисунок 9.3).

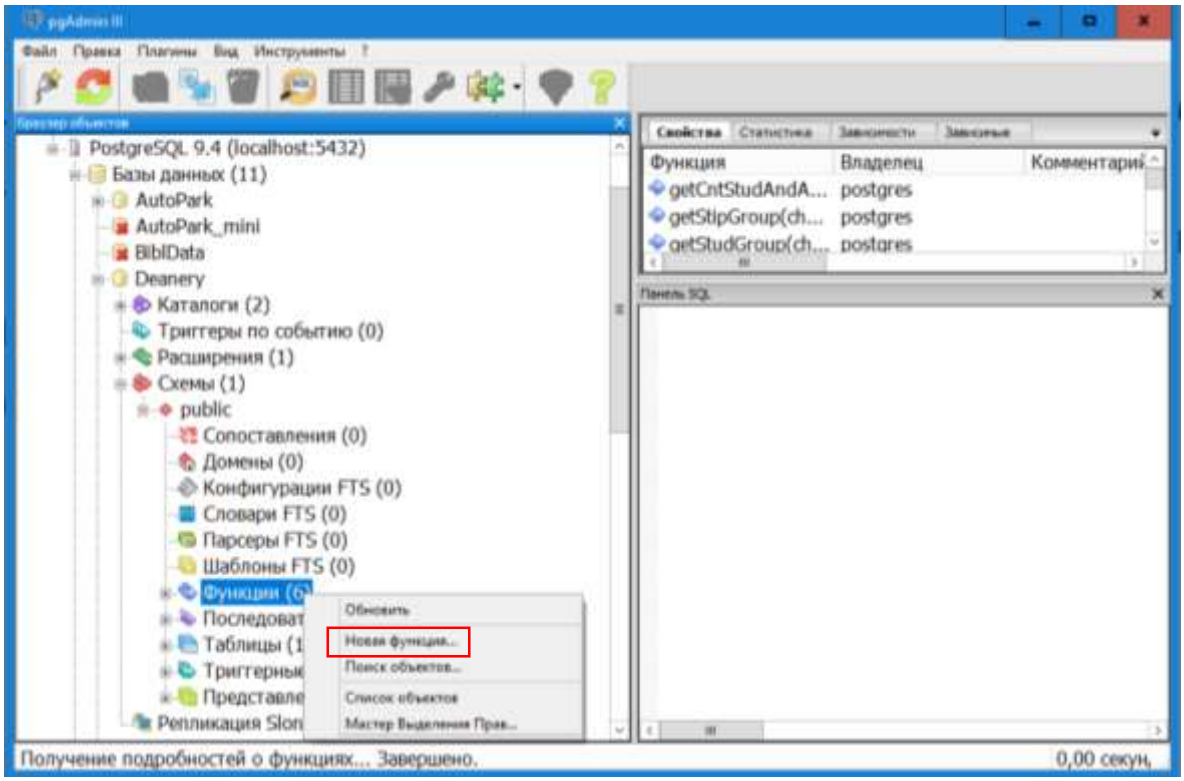


Рисунок 9.3 - Пункт Функції

Відкриється вікно визначення функції

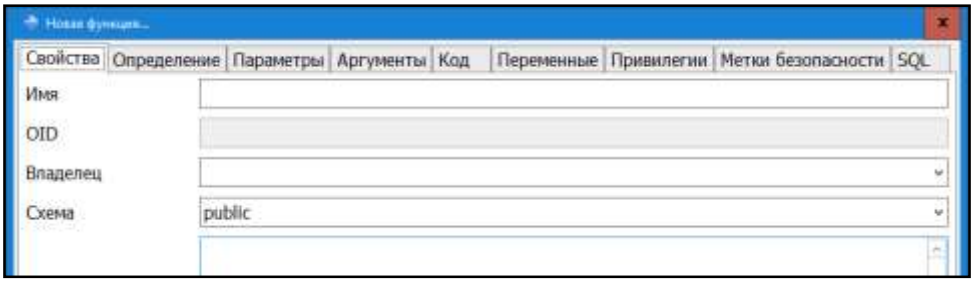


Рисунок 9.4 – Нова функція

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.

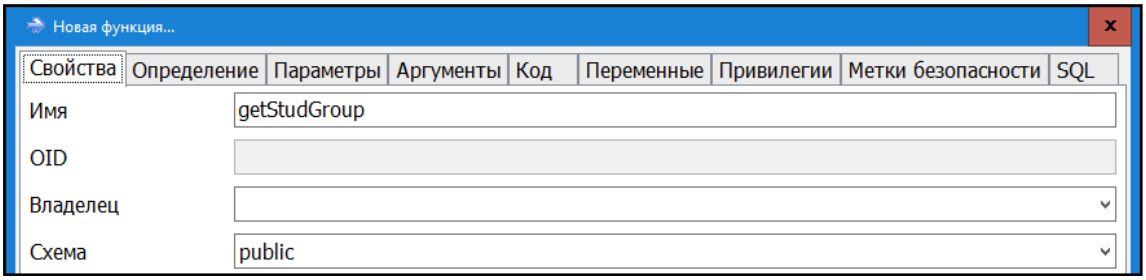


Рисунок 9.5 – Ім'я функції

2. Вказати допустимий в PostgreSQL тип значення, що повертається.

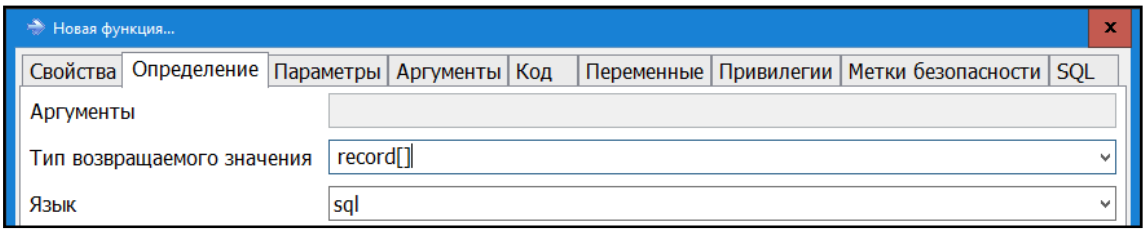
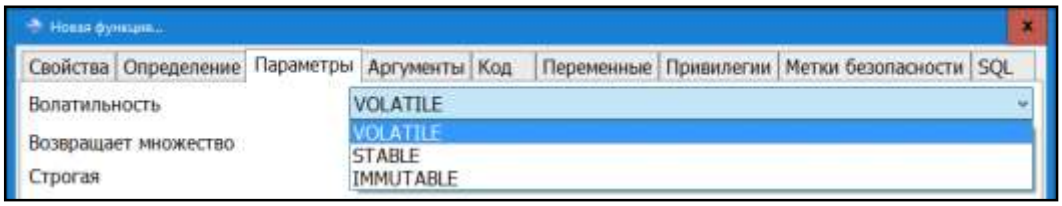


Рисунок 9.6 - Тип значення, що повертається

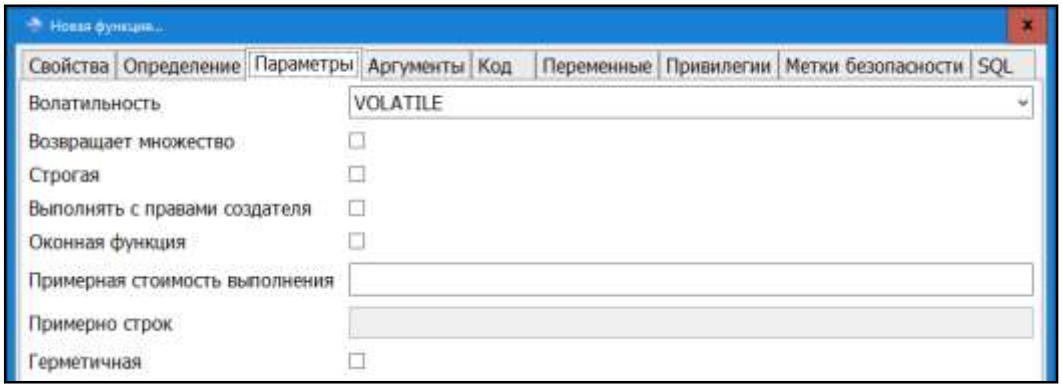
Слід підкреслити, що синтаксис PL/pgsql має більш виразні можливості типу результату, що повертається, але, в будь-якому випадку, ці типи зводяться до допустимих у PostgreSQL типів значень.

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 9.7);



а) – Тип волатильності



б) - Значення волатильності

Рисунок 9.7 – Тип та значення волатильності

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

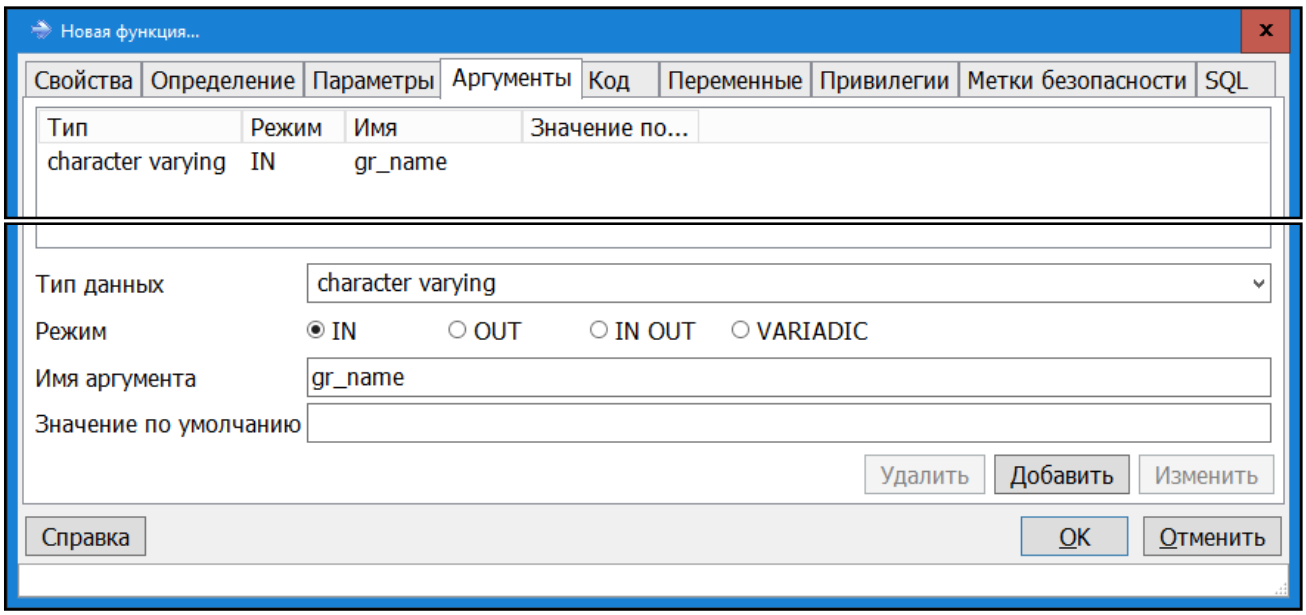


Рисунок 9.8 - аргументи функції

4. Задати код

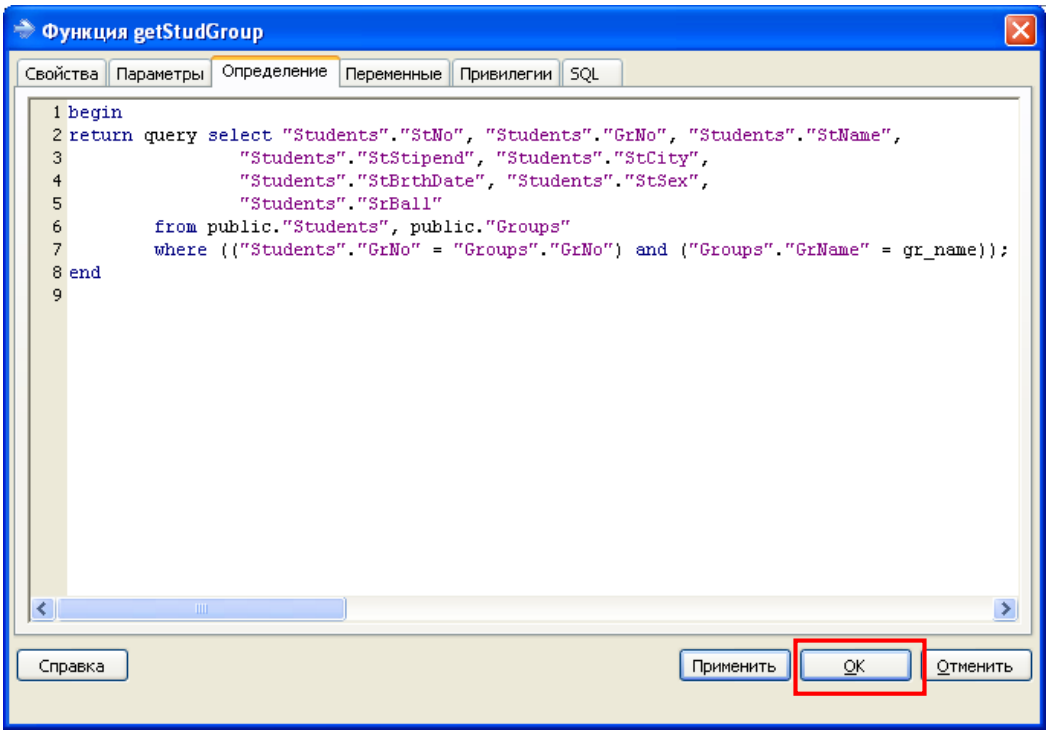


Рисунок 9.9 – Код функції

5. Отриманий текст функції можна побачити на закладці SQL

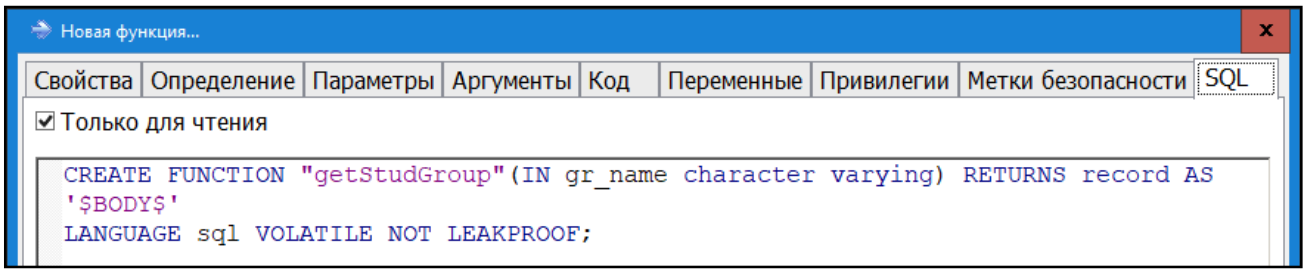


Рисунок 9.10 - Текст функції

9.3 Редагування серверної процедури

Для редагування серверної функції слід в контекстному меню функції вибрати пункт "Скрипт CREATE" (Рисунок 9.11).

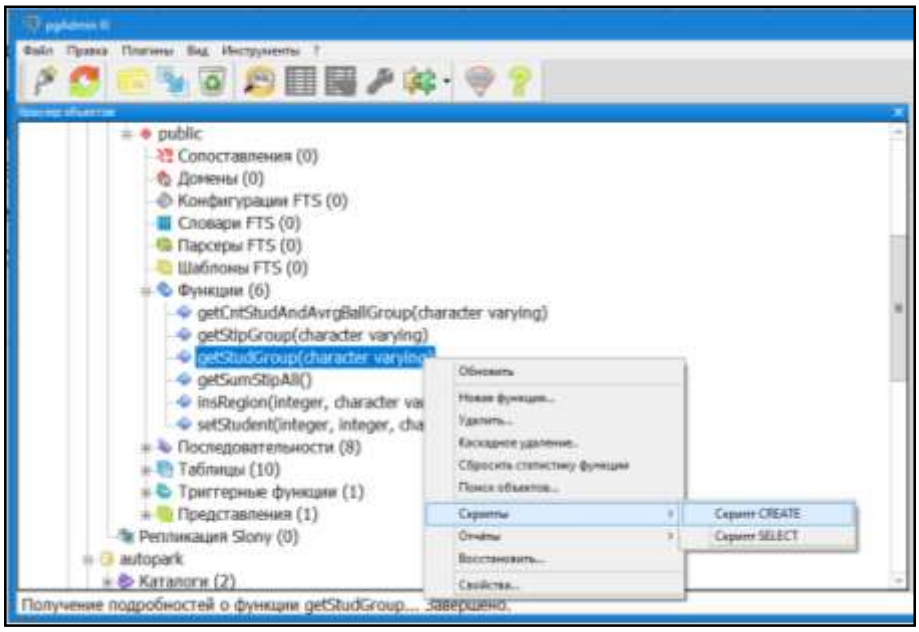


Рисунок 9.11 – Вибір скрипту

Тепер в екранному редакторі можна остаточно сформувати текст функції і відправити її на виконання (Рисунок 9.12 та Лістинг 9.1)

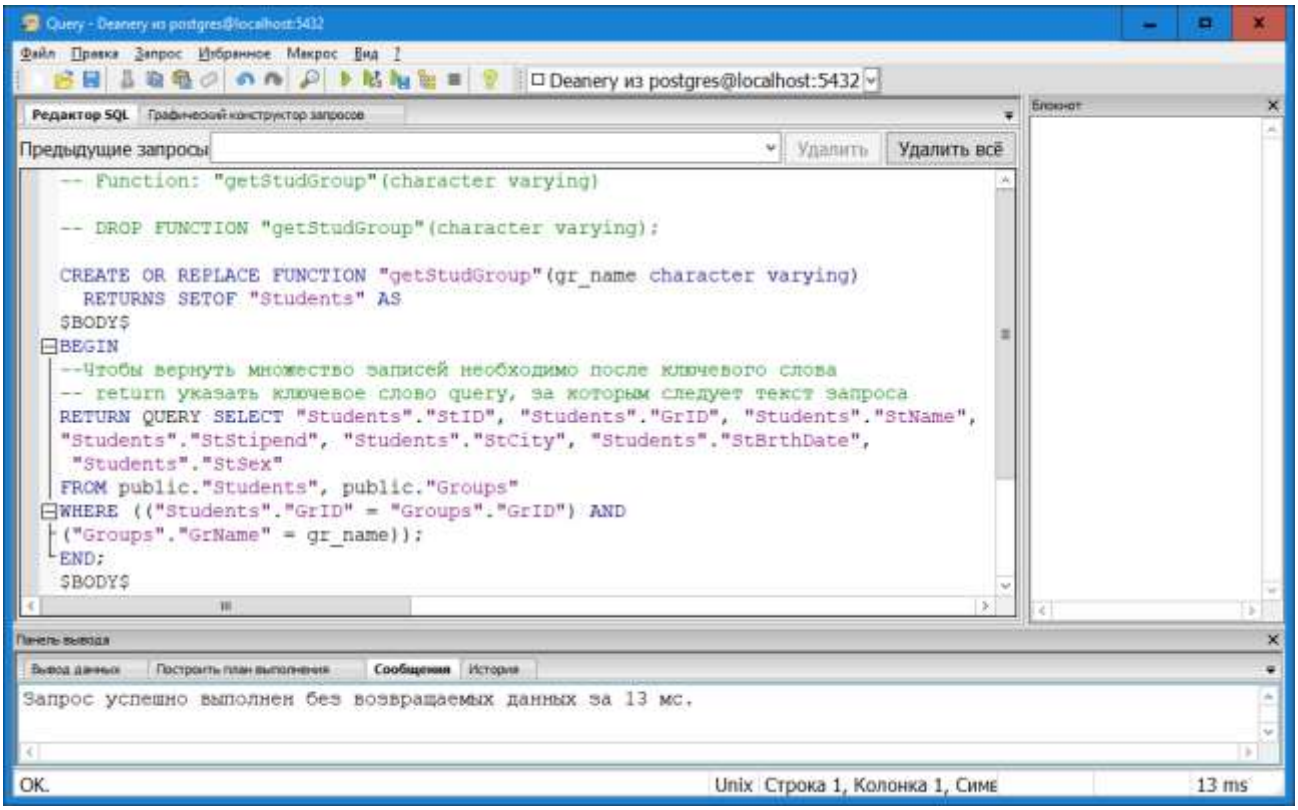


Рисунок 9.12 – Відредагований текст функції

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;
```

Лістинг 9.1 – Відредагований текст функції

9.4 Виклик серверних процедур в pgAdmin і прикладній програмі

Узагальнений виклик функції мови PL/pgSQL виглядає наступним чином:

SELECT * **FROM** public."Назва функції"([параметр, …]);

Так для наведеного вище прикладу виклик функції має вигляд:

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

9.4.1 Виклик серверної процедури в pgAdmin

Для виклику функції необхідно у вікні SQL-команд ввести відповідну команду і, якщо потрібно, параметри та відправити її на виконання (Рисунок 9.13)

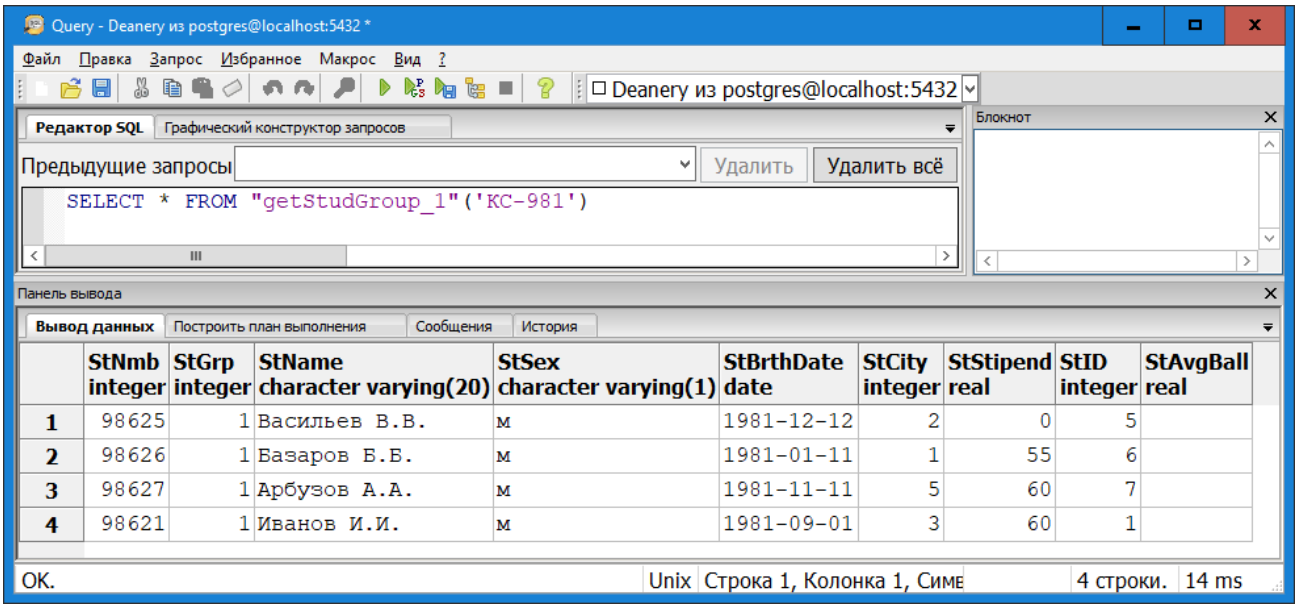


Рисунок 9.13 - Виклик серверної процедури

9.4.2 Виклик серверної процедури в прикладній програмі

Можливі два варіанта виклику серверної 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"('KC-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 з серверними процедурами, підготовки їх вхідних параметрів і виконання процедури.

Після виконання процедури необхідно виділити по черзі значення полів, що повертаються, оператора Select.

9.4.3 Виклик серверних процедур в Java засобами JDBC

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

JDBC підтримує виклик серверних 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 процедури і символи '?', які визначають необхідні параметри. Можливі два варіанти специфікації 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 один рядок або тільки одне значення.

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

Розглянемо варіанти виклику серверних процедур, наведених в Рисунок 9.11.

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 встановлення значення типу int першому параметру і setString(2, "Іванов") для встановлення значення типу String другому параметру функції insStudent().

Приклад 2. 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. Функція «підрахувати суму витрат на стипендію всіх студентів групи X».

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 два параметра. Перший параметр вихідний OUT - результат функції. Другий параметр - вхідний IN. Для його встановлення використовується метод setString(2, "КС-071"), де значення 2 вказує, що другому параметру серверної 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 серверної процедури три параметра. Перший і другий параметри вихідні OUT. 3-ий параметр - вхідний IN. Для його встановлення використовується метод setString(3, "КС-981"), де значення 3 вказує, що 3-му параметру 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("Студент " + name + " стипендия " + stip);
}
```

```
rs.close();
```

Результатом роботи 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 процедури необхідно провести перетворення до типу ResultSet. Далі використовується стандартний спосіб обходу записів об'єкта ResultSet.

9.4.4 Виклик серверних процедур або функцій у Python

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

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

Модуль psycopg2 підтримує виклик серверних процедур за допомогою звернення

cursor.callproc()

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

1. Імпорт в файл зі зверненням

```
import psycopg2
```

2. Створення об'єкту з'єднання з БД

```
ps_connection = psycopg2.connect(Connection Arguments)
```

3. Створення об'єкту курсор

```
cursor = ps_connection.cursor()
```

4. Виклик та виконання серверної процедури або функції, використовуючи метод cursor.callproc(). Тут необхідно знати назву серверної 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;

5. 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 цієї 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 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")
```

10 СТВОРЕННЯ ТА ВИКОНАНЯ ТРИГЕРІВ

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

10.1 Синтаксис визначення тригера в PostgreSQL

Синтаксис визначення тригера в PostgreSQL наступний::

CREATE TRIGGER ім'я_тригера 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:

1. По одному разу щодо кожного рядка, який буде підданий впливу події тригера (FOR EACH ROW);
2. 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 вставлено або оновлено.

Більш наочно використання цих змінних наведено в таблиці 10.1.

Таблиця 10.1 – Використання змінних OLD та NEW

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

10.2 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 10.1).

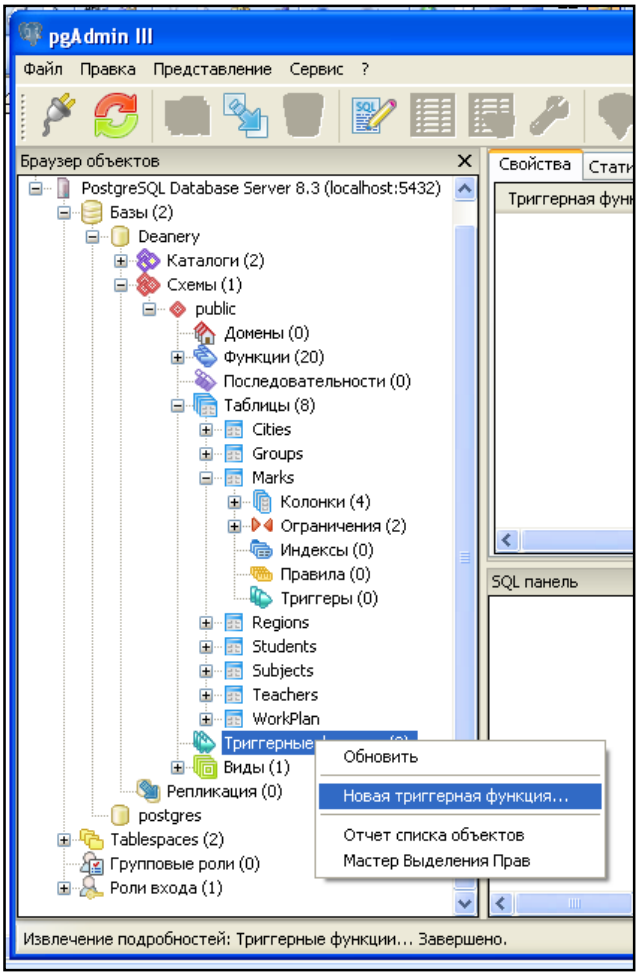


Рисунок 10.1 – Контекстне меню розділу Тригерні функції

В результаті відкриється вікно створення нової критичної функції (Рисунок 10.2).

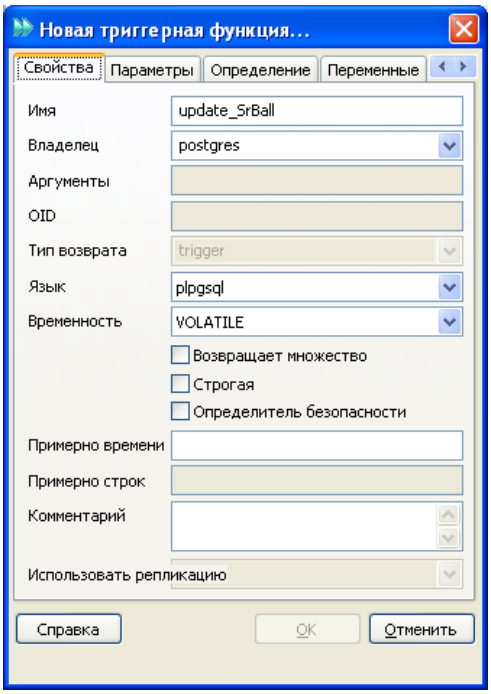


Рисунок 10.2 – Вікно Нова тригерна функція. Закладка Властивості

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 10.3):

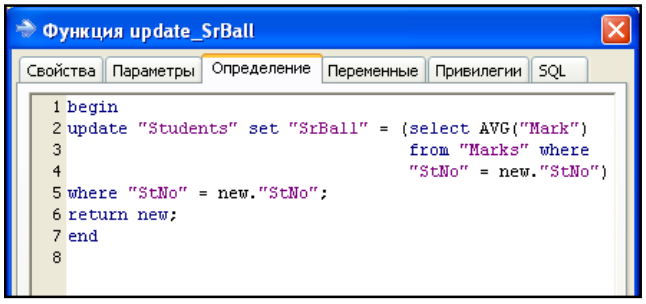


Рисунок 10.3 – Закладка Визначення вікна Нова тригерна функція

Повний код критичної функції можна переглянути на закладці SQL (Рисунок 10.4).

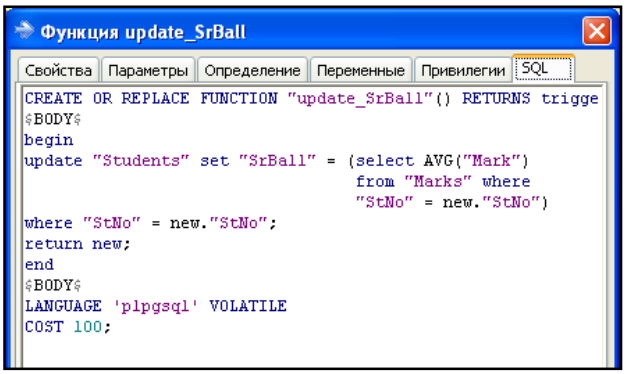


Рисунок 10.4 – Закладка SQL вікна Нова тригерна функція

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 10.5).

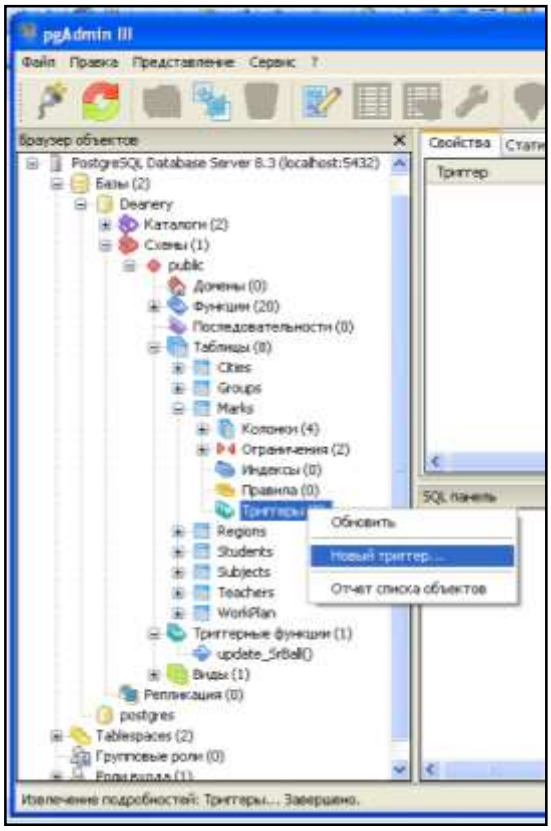


Рисунок 10.5 – Контекстне меню розділу Тригери

В результаті відкривається вікно створення нового тригеру (Рисунок 10.6).

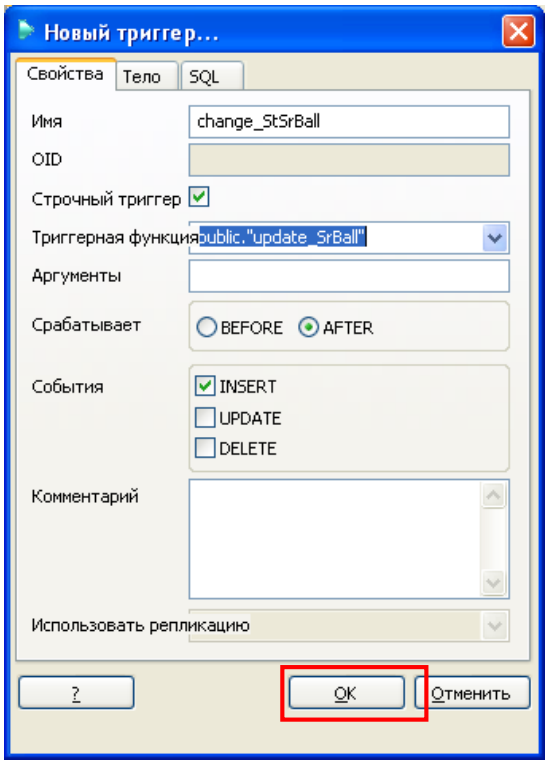


Рисунок 10.6 – Вікно Новий тригер. Закладка Властивості

У властивостях нового тригера необхідно вказати наступне:

- Ім'я - ім'я нового тригера;

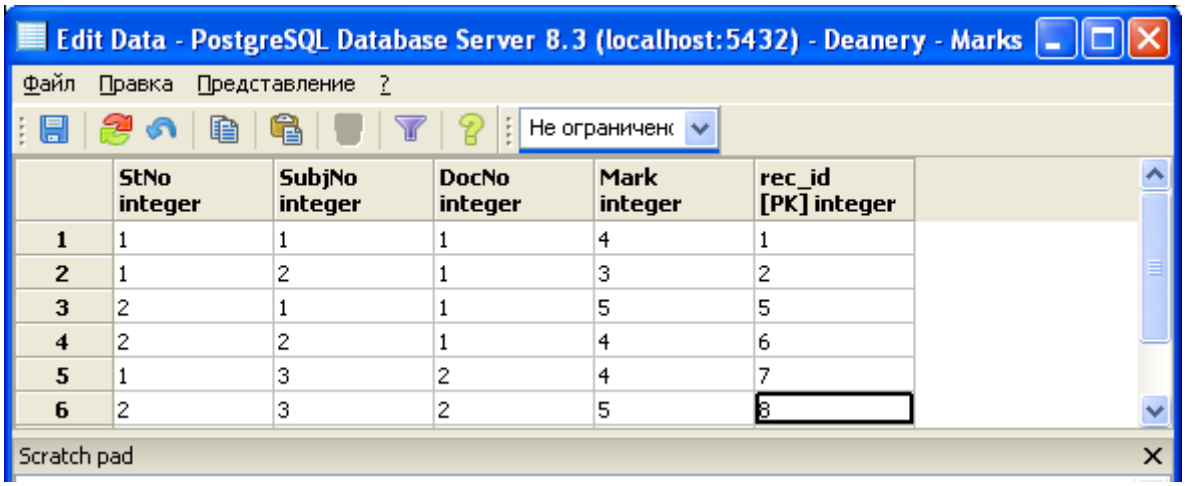
- властивість Рядковий тригер задає рівень тригера: FOR EACH ROW (властивість вибрано) або FOR EACH STATEMENT (властивість не вибрано).

- 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);

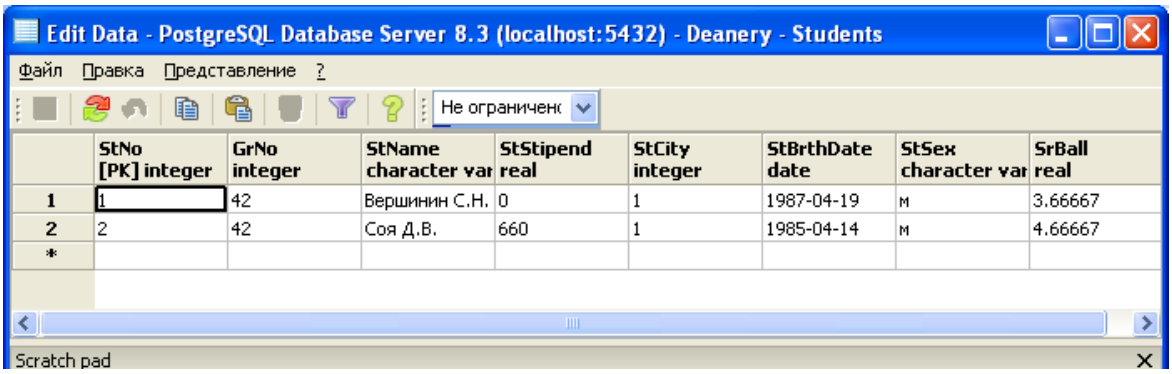
- Події - вказується подія, за яким ініціюється тригер. Важливо підкреслити, що для одного тригера можна вибрати одне або кілька подій.

Перевірити роботу тригера можна при додаванні нового запису в таблицю Marks (Рисунок 10.7). Автоматично буде змінено значення колонки SrBall таблиці Students для заданого студента (Рисунок 10.8).



	StNo integer	SubjNo integer	DocNo integer	Mark integer	rec_id [PK] integer
1	1	1	1	4	1
2	1	2	1	3	2
3	2	1	1	5	5
4	2	2	1	4	6
5	1	3	2	4	7
6	2	3	2	5	8

Рисунок 10.7 – Вміст таблиці Marks



	StNo [PK] integer	GrNo integer	StName character var	StStipend real	StCity integer	StBrthDate date	StSex character var	SrBall real
1	1	42	Вершинин С.Н.	0	1	1987-04-19	м	3.66667
2	2	42	Соя Д.В.	660	1	1985-04-14	м	4.66667
*								

Рисунок 10.8 – Вміст таблиці Students

11 РЕЗЕРВНЕ КОПІЮВАННЯ ТА ВІДНОВЛЕННЯ БД

11.1 BackUp - Резервне копіювання БД

До складу PostgreSQL входить 2-вІ утиліти резервного копіювання BackUp. Це pg_dumpall, що виконує резервну копію всіх БД (включаючи системну) і pg_dump, яка виконує резервне копіювання однієї БД. Обидві ці утиліти вміють зберігати копію БД як в текстовому так і в бінарному вигляді.

Розглянемо процес створення резервної копії засобами pgAdmin III, в якому підтримується можливість створювати резервну копію (BackUp) окремої таблиці БД, схеми БД, всієї БД або всього сервера БД.

Нижче наводиться опис процесу створення резервної копії БД Deanery.

1. У вікні Браузер об'єктів необхідно вибрати певну БД, активізувати її контекстне меню і в ньому вибрати пункт Резервна копія (Рисунок 11.1);
2. У вікні, Резервна копія (Рисунок 11.2) потрібно вказати ім'я та шлях до файлу резервної копії, вибрати формат і додаткові опції. Якщо обрана опція ББО (Blobs) (задається за замовчуванням), то в файл резервної копії БД будуть збережені об'єкти типу blobs;
3. Після чого натиснути кнопку ОК;
4. На закладці Повідомлення вікна Backup База буде виведений лістинг процесу створення резервної копії БД (Рисунок 11.3).

Для виконання стандартного резервного копіювання і відновлення рекомендується використовувати варіанти COMPRESS або TAR в стандартному режимі, запропонованому системою за замовчуванням. У цьому випадку файл резервної копії має розширення .backup.

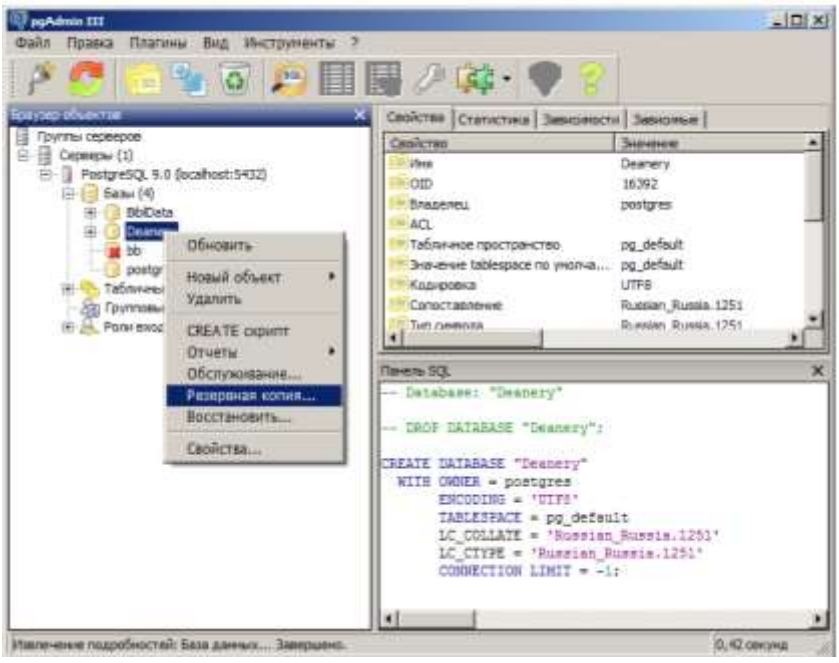


Рисунок 11.1 – Контекстне меню об'єкта БД

З'явиться вікно, на закладці Файл якого можна задати ім'я файлу для збереження копії. Достатньо вказати каталог, в якому буде створено файл резервної копії з розширенням .bakup.

Інші параметри цієї закладки слід залишити незмінними (Рисунок 11.2).

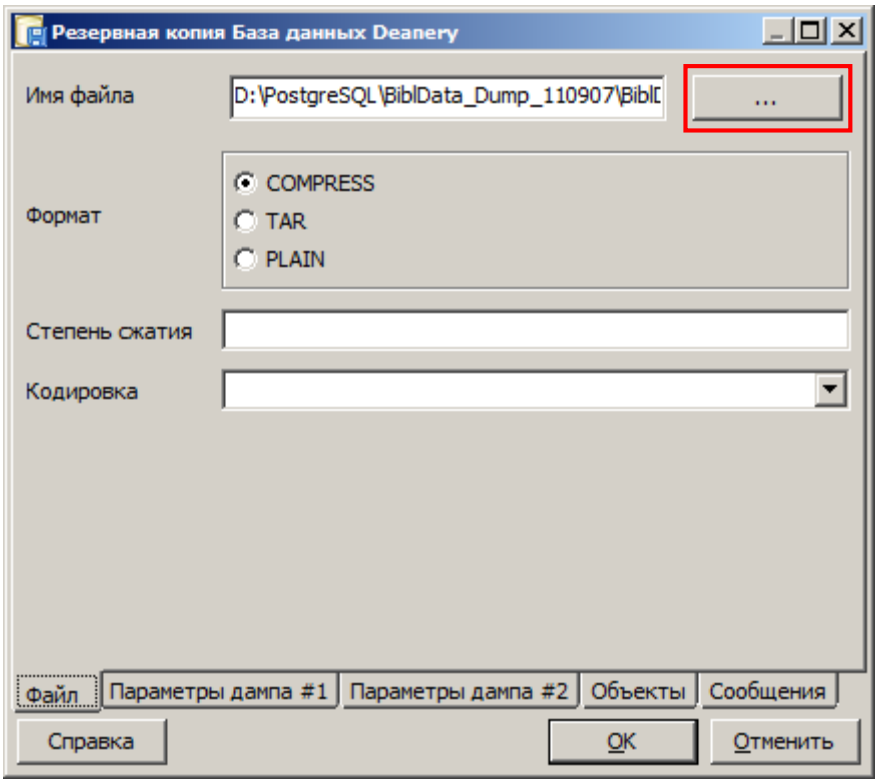


Рисунок 11.2 – Вікно введення параметрів резервного копіювання БД

Вибір пункту Використовувати команди INSERT на закладці Параметри дампа №1 або Параметри дампа №2, використовується для створення копії (дампа), яка може бути відновлена в СУБД інших виробників. Файл дампа в цьому випадку має те ж розширення .backup.

Примітка. Копія з таким налаштуваннями може бути відновлена і PostgreSQL, але відновлення відбувається значно повільніше, ніж в звичайному режимі.

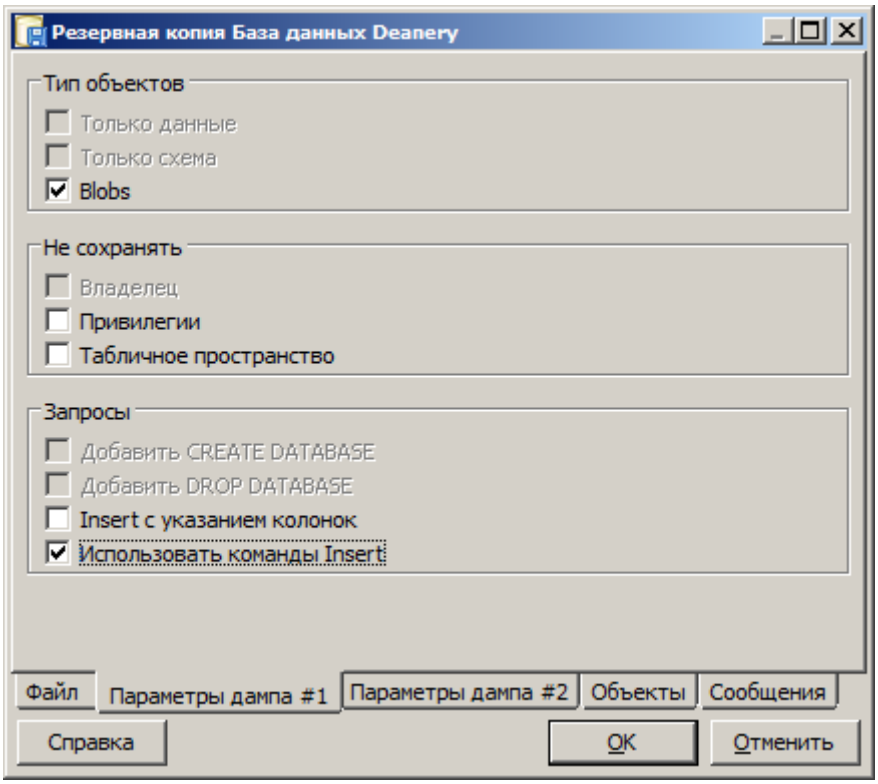


Рисунок 11.3 – Вікно вводу додаткових параметрів резервного копіювання БД

Якщо копіювання пройшло успішно отримаємо повідомлення "Процес повернув код виходу 0" (Рисунок 11.4)

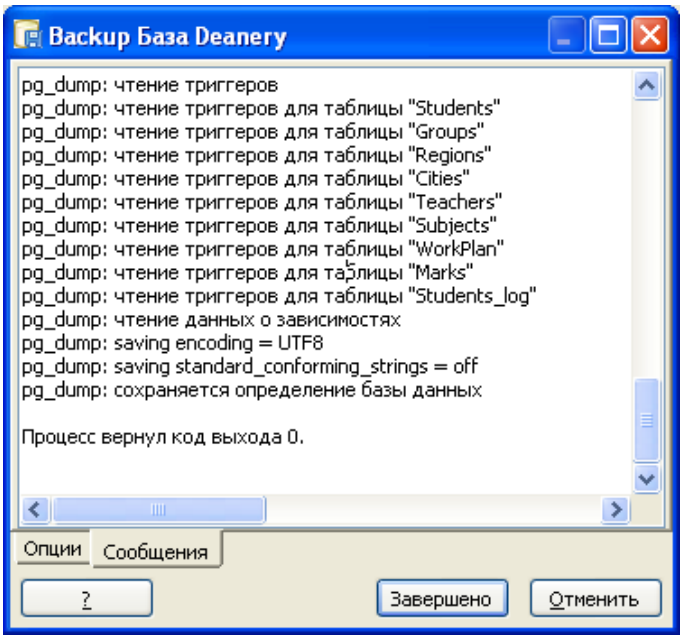


Рисунок 11.4 – Закладка Повідомлення вікна Backup База

11.2 Restore - Відновлення БД

Нижче наведено опис способу відновлення (Restore) БД в pgAdmin III.

Попередня умова: Повинна бути створена нова *БД,* наприклад Deanery_Restore.

1. Вибрати у вікні Браузер об'єктів нову БД, активізувати її контекстне меню і в ньому вибрати пункт Відновити (Рисунок 11.5);

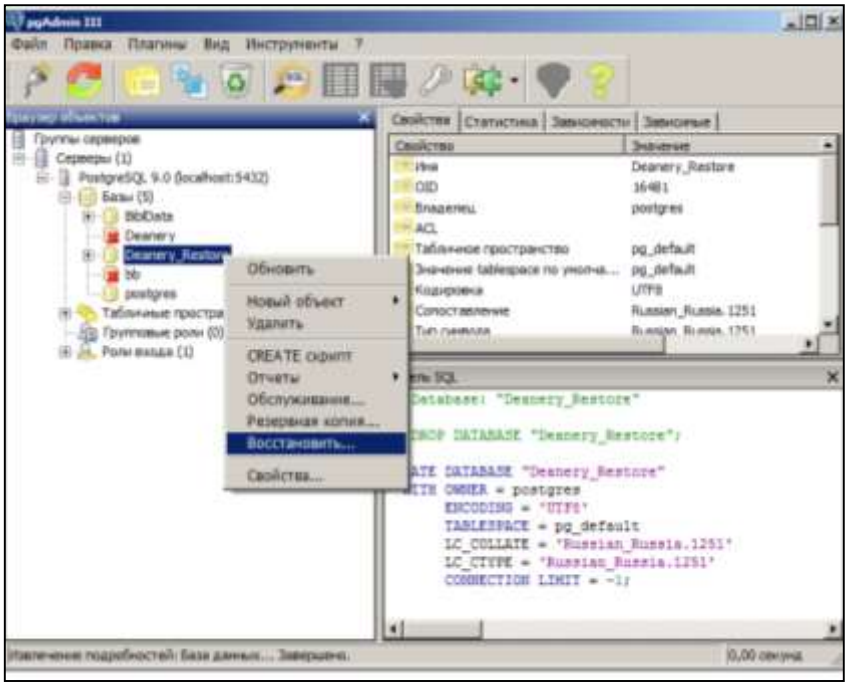


Рисунок 11.5 – Контекстне меню об'єкта БД

2. У вікні Відновлення Бази (Рисунок 11.6) на вкладці Файл слід вказати файл, який містить резервну копію БД;
3. Після чого натиснути кнопку ОК

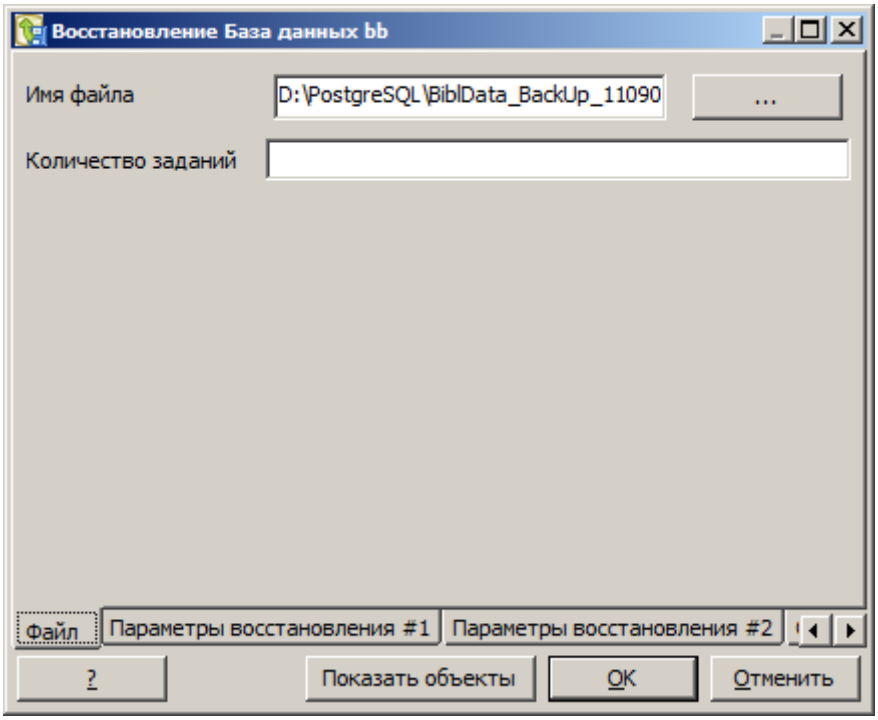


Рисунок 11.6 – Вікно вводу параметрів відновлення БД

Якщо буде потрібно, то можна задати додаткові опції відновлення (Рисунок 11.7). На вкладці Параметри відновлення №1 можна зокрема вимагати:

✓ Додати CREATE DATABASE - Створити нову БД;

✓ Очистити перед відновленням - перед відновленням очистити існуючу БД.

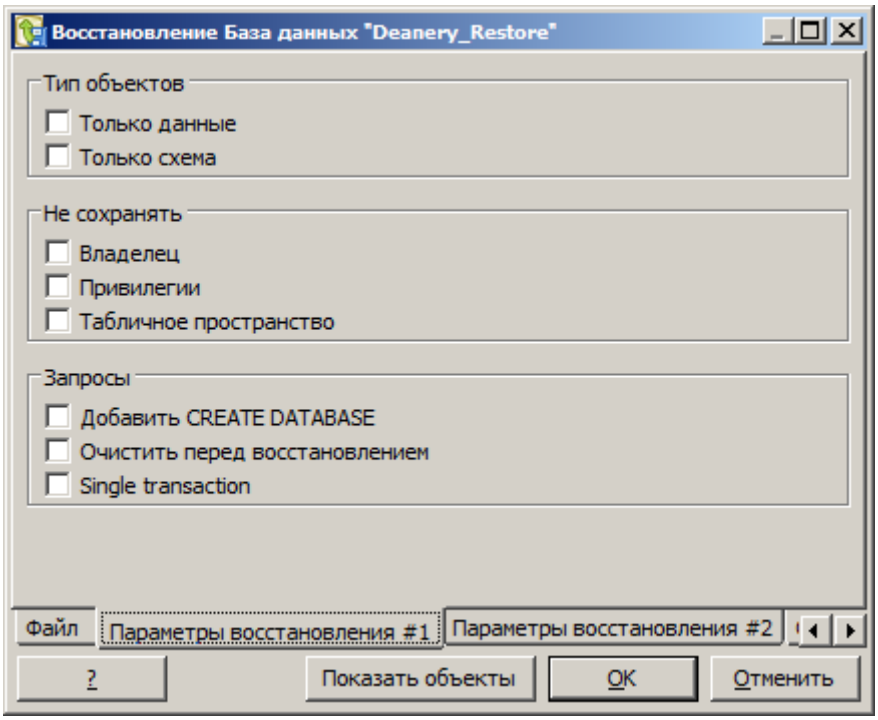


Рисунок 11.7 – Вікно вводу додаткових параметрів відновлення БД

При успішному відновленні з'явиться повідомлення виду "Процес повернув код виходу 0" (Рисунок 11.8)

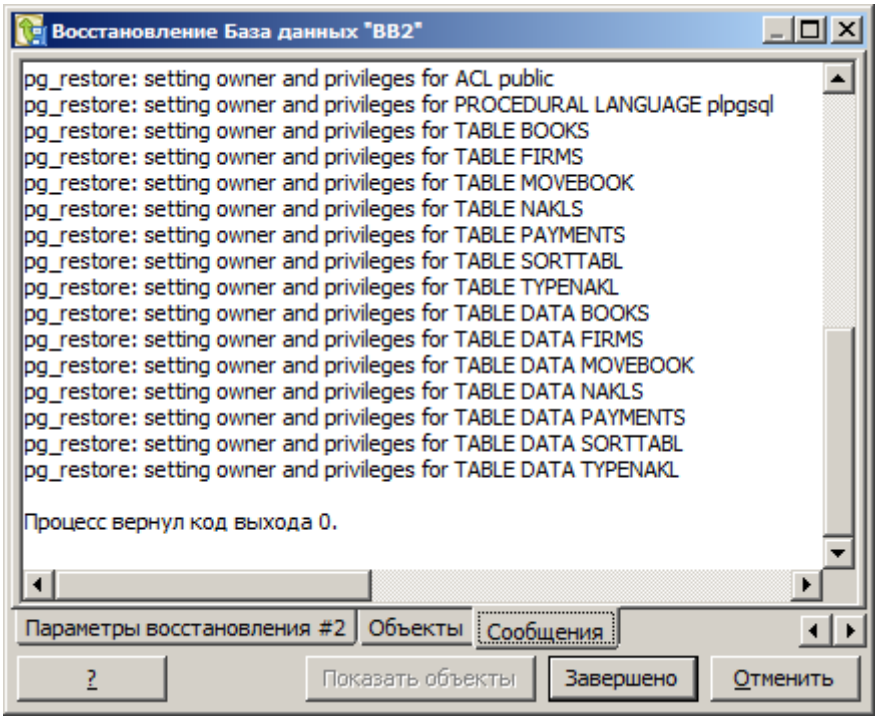


Рисунок 11.8 – Закладка Повідомлення вікна Відновлення БД

11.3 Створення SQL-дампа БД

Ідея створення SQL-дампа БД полягає в створенні текстового файлу з командами SQL INSERT, за допомогою яких на сервері можна відтворити БД в реляційної СУБД будь-якого виробника, що забезпечує перенесення БД між різними СУБД.

Розглянемо процес створення резервної копії (BackUp) засобами pgAdmin III, в якому підтримується можливість створювати резервну копію окремої таблиці БД, схеми БД, всієї БД або всього сервера БД.

Нижче наводиться опис процесу створення SQL-дампа БД Deanery:

1. У вікні Браузер об'єктів необхідно вибрати певну БД, активізувати її контекстне меню і в ньому вибрати пункт Резервна копія (Рисунок 11.9);

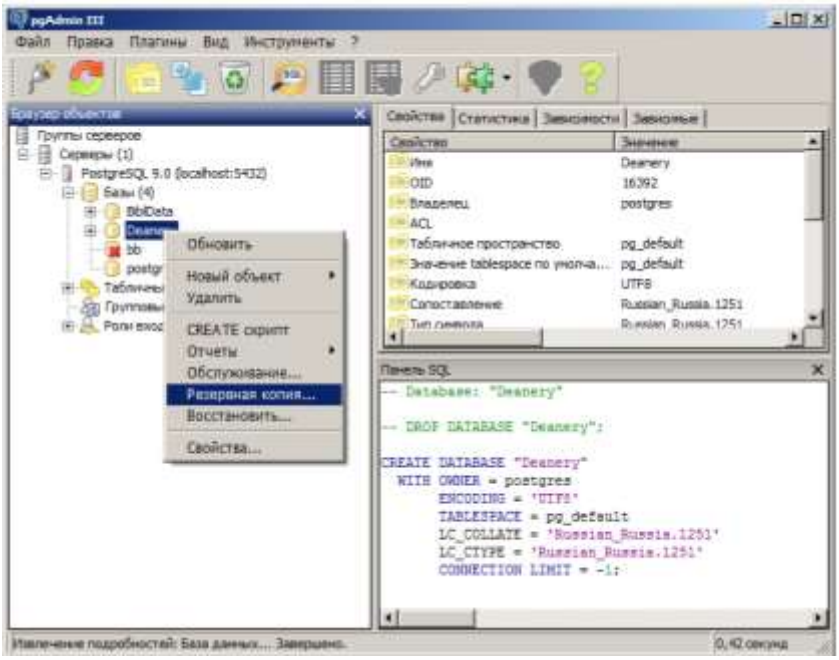


Рисунок 11.9 – Контекстне меню об’єкта БД

2. У вікні (Рисунок 11.10.а) Резервна копія Бази даних потрібно:

а) У полі ім'я файлу вказати ім'я та шлях до файлу SQL-дампа;

б) У полі Формат слід вибрати Простий (Plain), що і призведе до створення SQL-дампа;

3. На вкладці Параметри дампа #1 (Рисунок 11.10.б) обрати додаткові опції. Якщо опція вибрана Blobs (задається за замовчуванням), то в файл SQL-дампа БД будуть збережені об'єкти типу blobs;

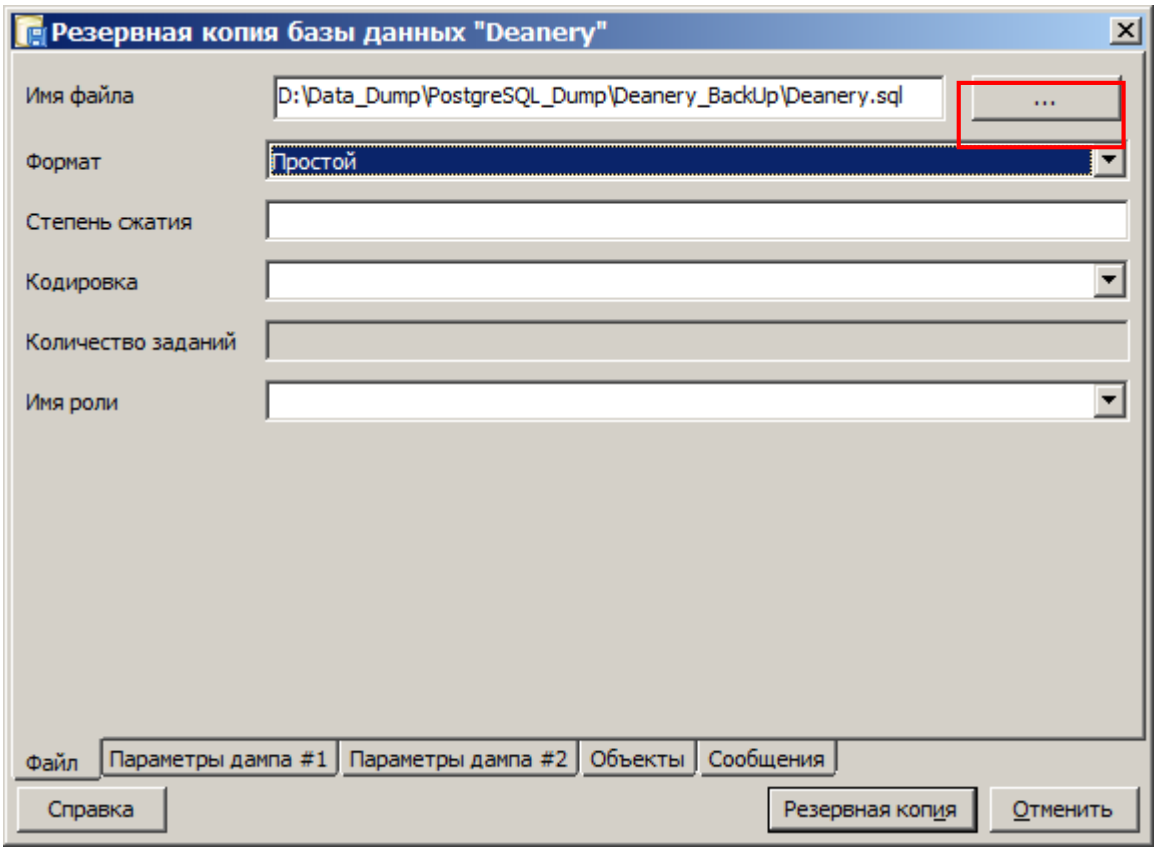
4. На вкладці Параметри дампа #2 (Рисунок 11.10.в) обрати опцію Використовувати команду Insert.

5. Натиснути кнопку Резервна копія;

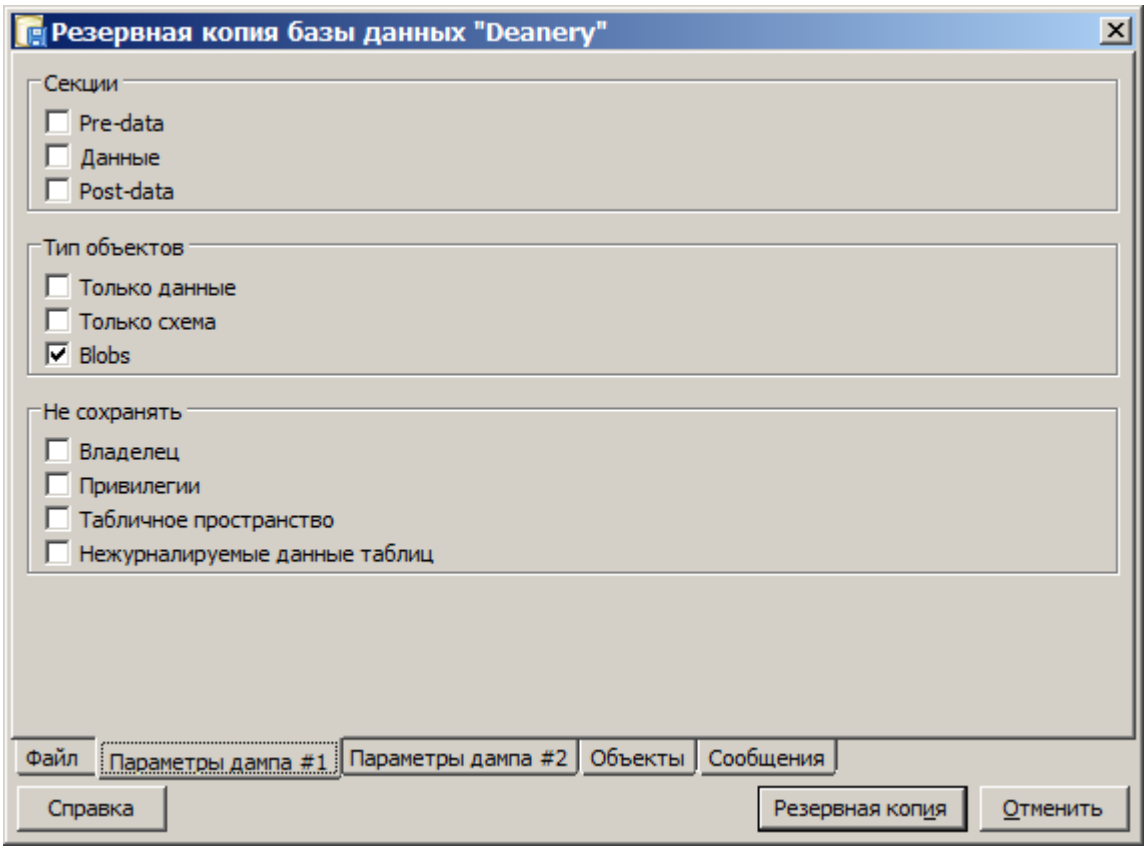
6. На вкладці Повідомлення (Рисунок 11.10.г) буде виведений лістинг процесу створення резервної копії БД.

З'явиться вікно, на закладці Файл якого можна задати ім'я файлу для збереження sql-дампа. Досить вказати каталог, в якому буде створено файл дампа з розширенням .sql.

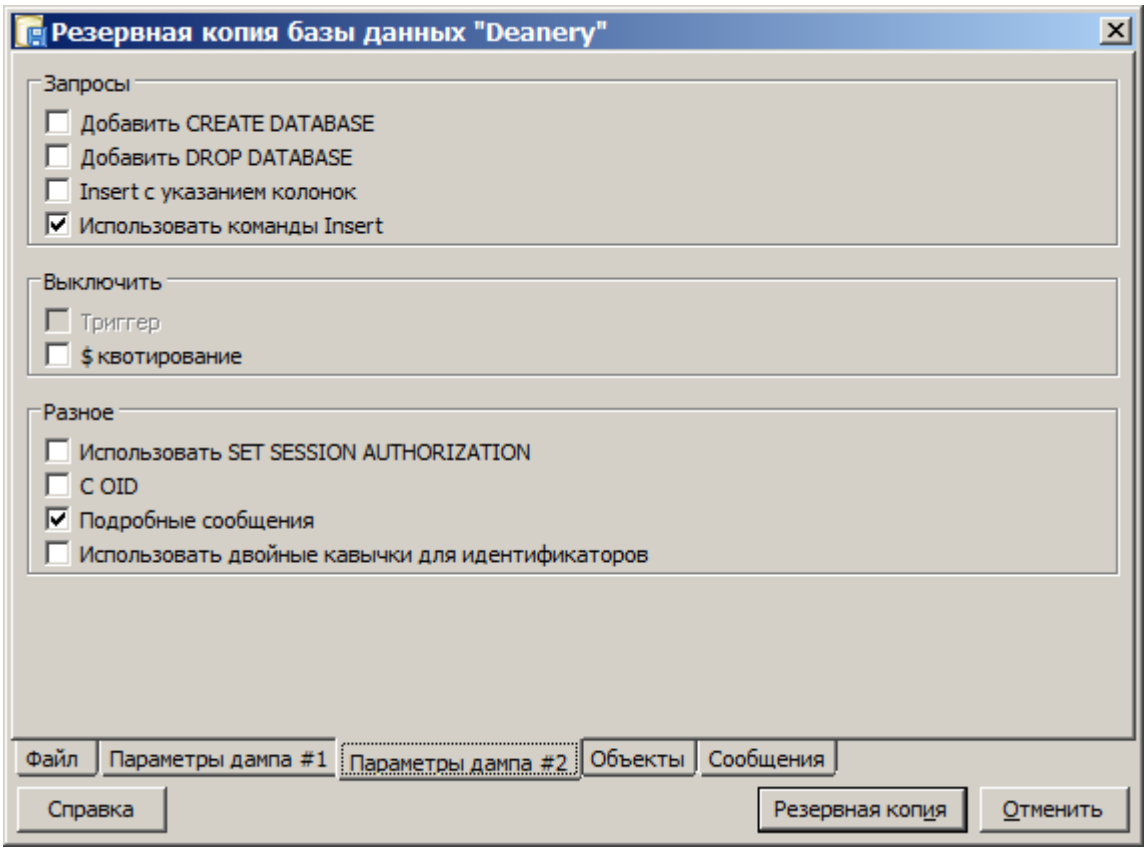
Інші параметри цієї закладки слід залишити незмінними.



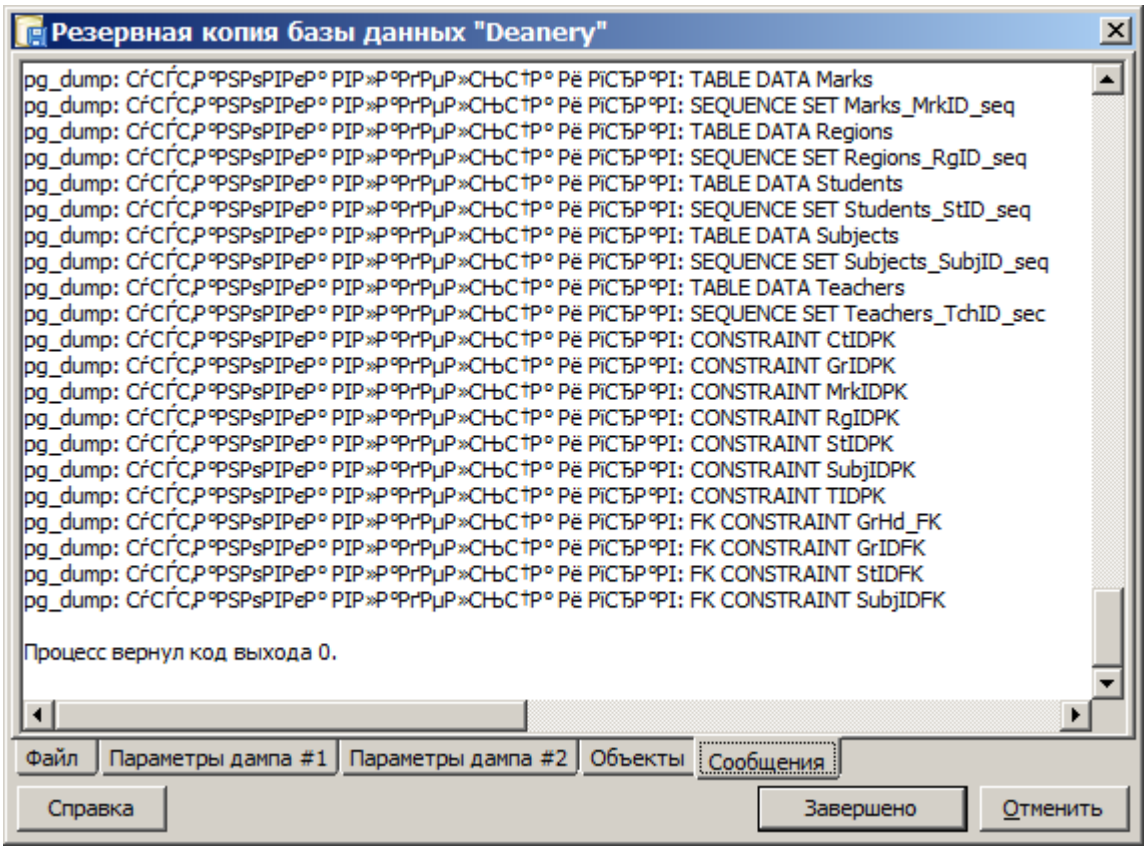
а) - вкладка Файл



б) – вкладка Параметри дампа #1



в) - вкладка Параметри дампа #2



г) вкладка Повідомлення

Рисунок 11.10 – Вікна уведення параметрів SQL-дампа БД

Засоби інших виробників, наприклад менеджер PgMaestro, добре виконують роботу по створенню і відновленню SQL-дампа, який відновлюється і pgAdmin III.

11.4 Відновлення SQL-дампа засобами pgAdmin III

Викликати вікно SQL–запиту (Рисунок 11.11).

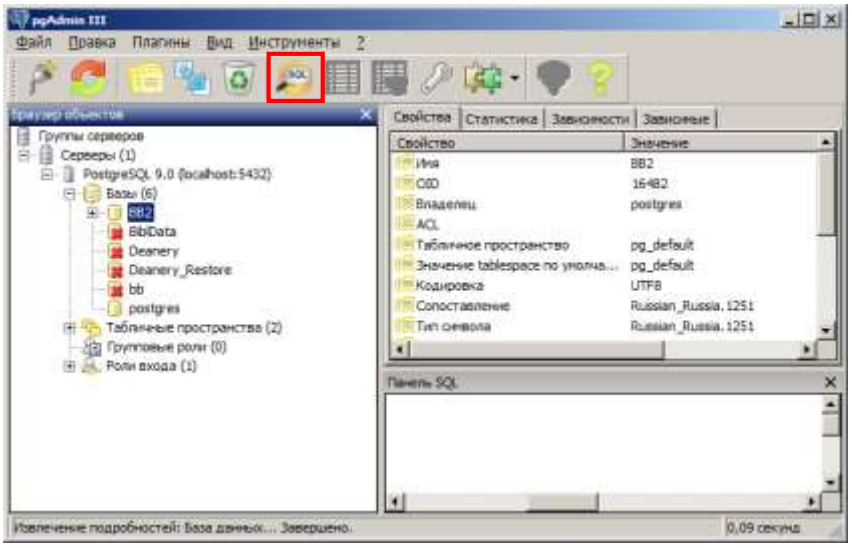


Рисунок 11.11 – Виклик вікна SQL–запиту

Виконати пошук файлу з SQL-дампом

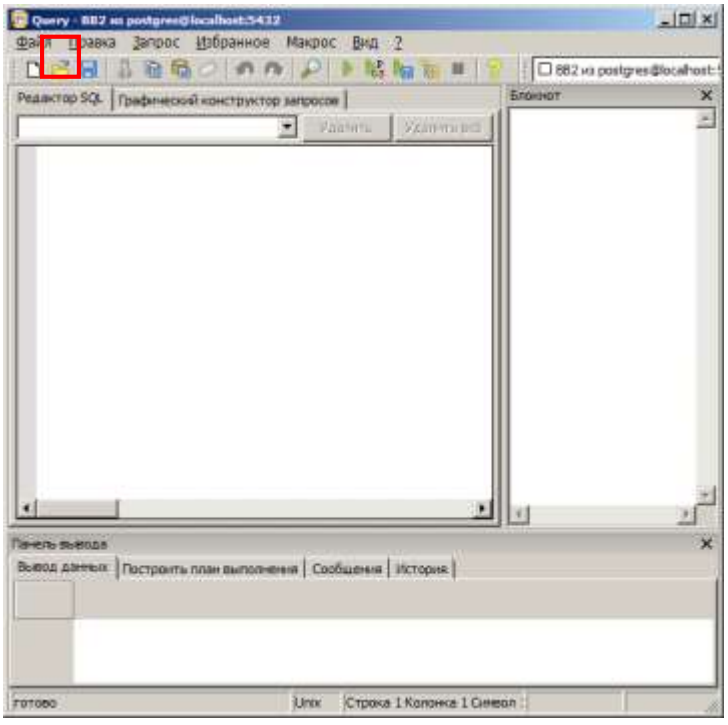


Рисунок 11.12 – Вибір файлу SQL-дампа

Відкрити файл з SQL-дампом (Рисунок 11.13)

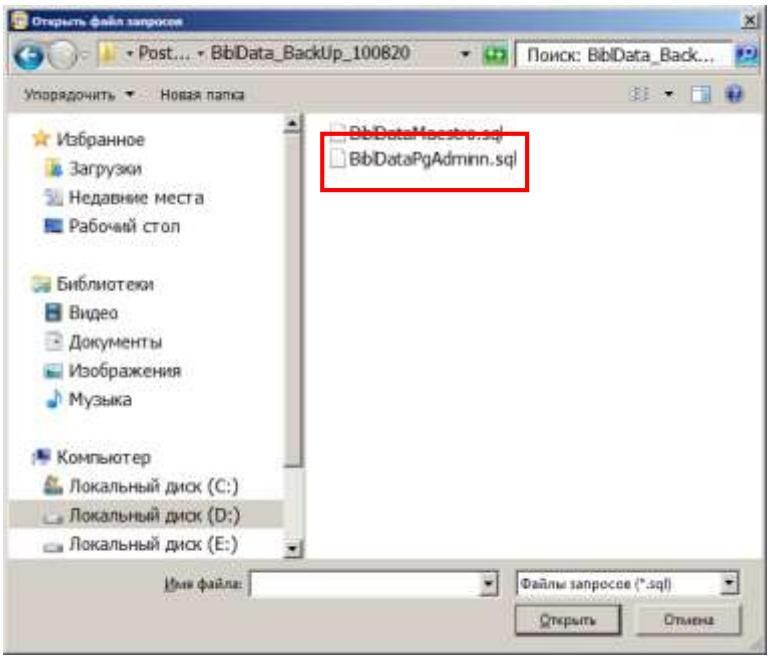


Рисунок 11.13 – Відкриття файлу SQL-дампа

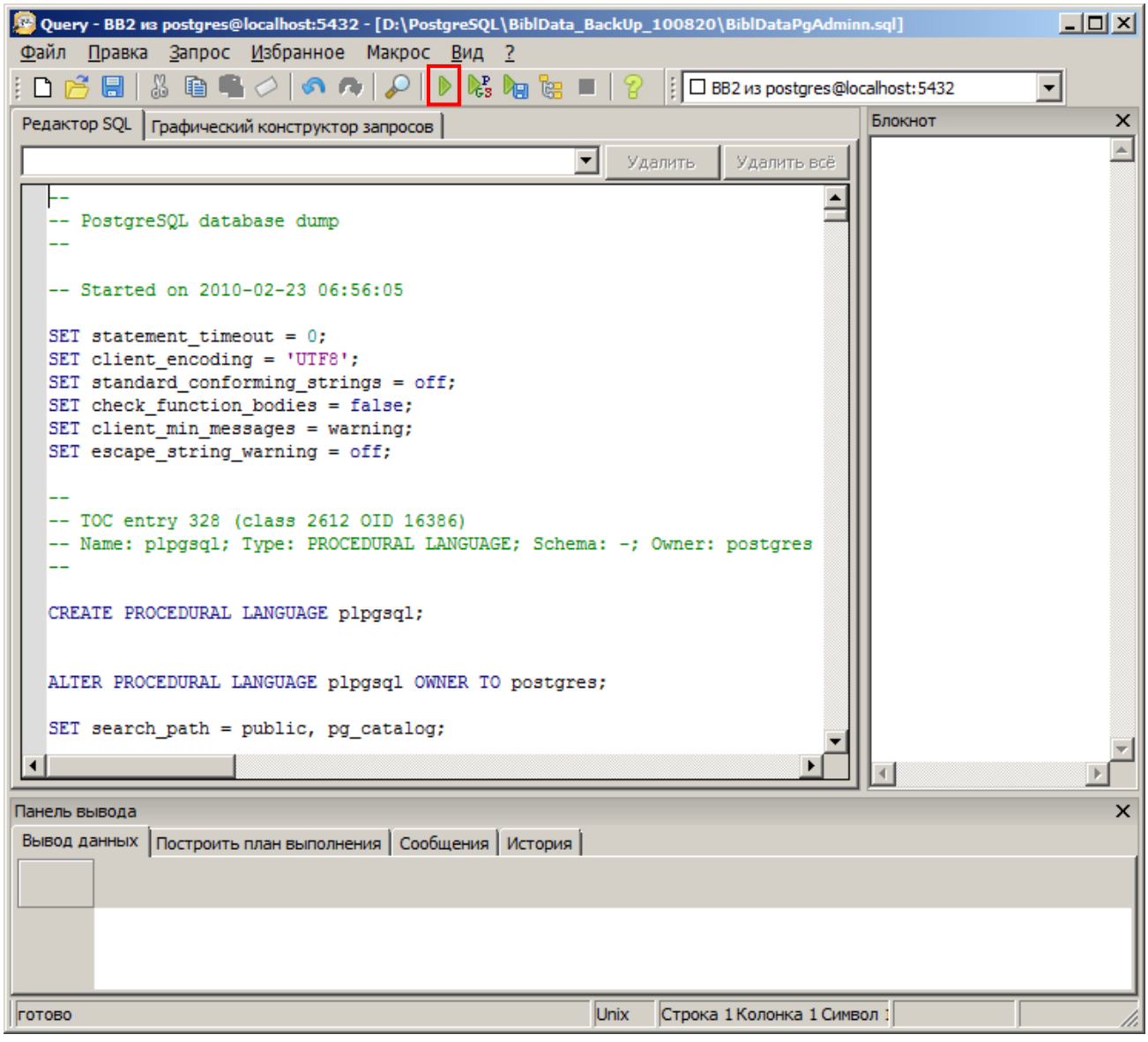


Рисунок 11.14 – Відновлення SQL-дампа

11.5 Створення дампа БД за допомогою утиліт pg_dump та pg_dumpall

З метою демонстрації можливостей по створенню / відновленню SQL-дампа по глобальним мережам при встановленні жорстких обмежень на доступ до серверів тільки через SSL порт *(Secure Sockets Layer - Протокол захищених сокетів*) далі для створення SQL-дампа БД в PostgreSQL буде застосована утиліта pg_dump.

Утиліта pg_dump являє собою звичайне клієнтську програму PostgreSQL. Приклад її використання може виглядати наступним чином:

pg_dump dbname > outfile,

де

dbname – ім'я БД на сервері,

outfile – ім'я нового файлу, який буде вміщувати SQL-дамп БД.

Примітка 2. В якості outfile можна вказувати повний шлях до файлу. За замовчуванням новий файл буде створений в папці \bin каталогу, куди було встановлено PostgreSQL (наприклад, C:\Program Files\PostgreSQL\ 9.5\bin\).

Утиліта pg_dump дозволяє робити резервне копіювання БД з будь-якого віддаленого комп'ютера, який має доступ до цієї БД. Однак, слід пам'ятати, що pg_dump не працює з особливо налаштованими правами. А саме, користувач, який запускає цю утиліту, повинен мати права на читання всіх таблиць, для яких йому потрібно створити резервні копії, тому, як правило, найкраще запускати pg_dump з-під суперкористувача БД.

Щоб вказати, до якого сервера слід звернутися pg_dump, в командному рядку використовуються опції -h host і -p port. В якості хосту за замовчуванням використовується локальний комп'ютер або те, що зазначено в змінної оточення PGHOST. Аналогічно, порт за замовчуванням визначається змінною оточення PGPORT або, якщо це не вдалося, значенням за замовчуванням, зазначеним при компіляції.

Як і будь-яке інше клієнтське додаток PostgreSQL, pg_dump за замовчуванням буде встановлювати з'єднання з ім'ям користувача БД, відповідним імені поточного користувача операційної системи. Це можна змінити за допомогою опції -U або встановленням змінної оточення PGUSER. Слід також пам'ятати, що з'єднання pg_dump підкоряються звичайним механізмам аутентифікації клієнтів.

Дамп, створюваний за допомогою pg_dump, володіє внутрішньою зв'язністю, оскільки дамп являє собою "знімок БД" в стані на момент запуску pg_dump. Під час своєї роботи pg_dump не блокує виконання інших операцій. (Винятки становлять операції, що вимагають повного блокування, серед яких майже всі види ALTER TABLE).

Примітка 3. Якщо схема БД ґрунтується на використанні OID (наприклад, в якості зовнішніх ключів), необхідно дати pg_dump вказівки включити OID в дамп. Для цього використовується опція командного рядка -o. Більш докладно про опції pg_dump див. [11].

Як приклад наведемо опис процесу створення SQL-дампа для БД «Deanery» за допомогою командного рядка.

1. Запустити командний рядок PostgreSQL (Пуск–>Програми–> PostgreSQL–>Командний рядок). Або в Пуск–>Виконати самостійно набрати команду або її через Огляд

C:\Program Files\PostgreSQL\9.0\bin\pg_dump.exe

В результаті буде отримано вікно (Рисунок 11.15);

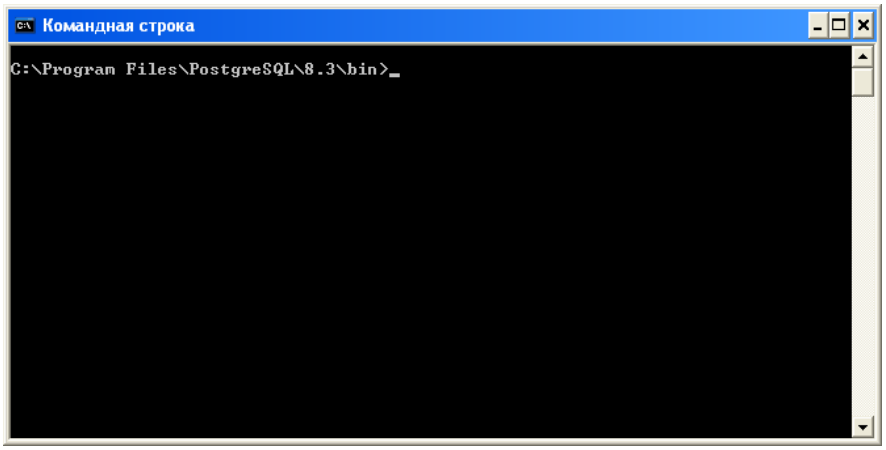


Рисунок 11.15 – Командний рядок PostgreSQL

2. Написати в командному рядку наступний рядок, після чого натиснути Enter (Рисунок 11.16):

pg_dump –U postgres Deanery > d:\BackUpDb\DeaneryBkUp

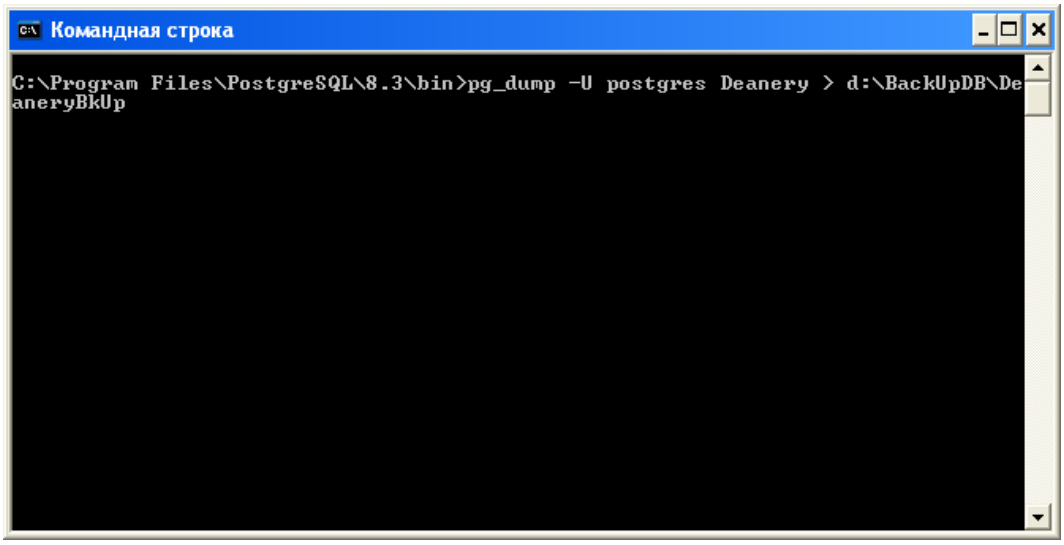


Рисунок 11.16 – Командний рядок PostgreSQL

Примітка: Як було сказано вище, pg_dump за замовчуванням буде встановлювати з'єднання з ім'ям користувача БД, відповідним імені поточного користувача операційної системи, який може не бути суперкористувачем цієї БД. Тому необхідно явно вказувати суперкористувача за допомогою опції -U.

3. Потім ввести пароль суперкористувача БД.

11.6 Відновлення SQL-дампа БД, створеного утилітою pg_dump

Текстові файли, створювані утилітою pg_dump, призначені для читання програмою psql. Відновлення дампа проводиться командою, загальний вигляд якої представлений нижче:

psql dbname < infile

де infile – це ім'я файлу (або шлях до нього), який вказувався в якості параметра outfile команди pg_dump.

Зауваження: При запуску цієї команди не створюється БД dbname, тому її необхідно створити заздалегідь з template0, перш ніж запускати утиліту psql (наприклад, createdb -T template0 dbname).

Для утиліти psql подібно pg_dump можна задати сервер БД, з яким слід встановлювати з'єднання, і ім'я користувача, під яким з'єднання буде встановлено.

Перед відновленням SQL-дампа, необхідно, щоб існували всі облікові записи користувачів, які є власниками об'єктів або мають права на об'єкти відновлюваної БД. Якщо їх не буде, неможливо буде відновити об'єкти з початковою приналежністю і / або правами на них.

За замовчуванням, якщо скрипт psql зустріне SQL-помилку, він все одно продовжить роботу. При бажанні, це можна змінити, додавши в самий верх скрипта наступний рядок:

\set ON_ERROR_STOP

Після цього, при виникненні SQL-помилки, утиліта psql буде переривати роботу і виходити зі статусом 3.

Нижче наводиться опис процесу відновлення SQL-дампа БД «Deanery».

1. У командному рядку PostgreSQL за допомогою команди createdb -U username restoreDeanery створюється нова БД (Рисунок 11.7). В результаті вікно Браузер об'єктів pgAdmin III набуде вигляду, показаному на Рисунок 11.8.

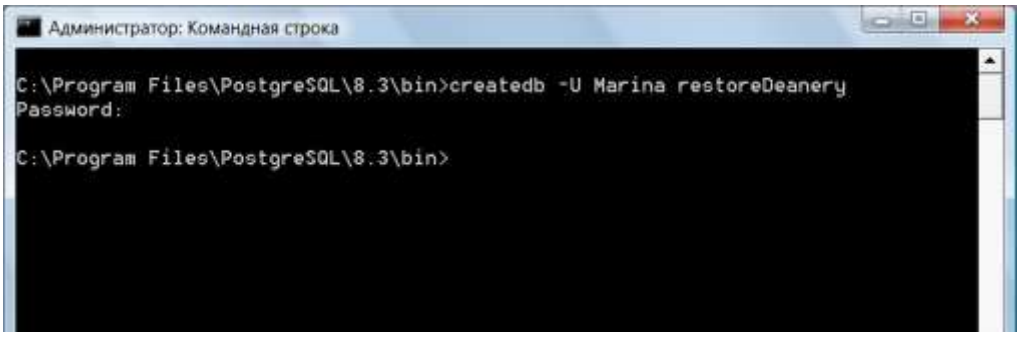


Рисунок 11.17 – Створення нової БД

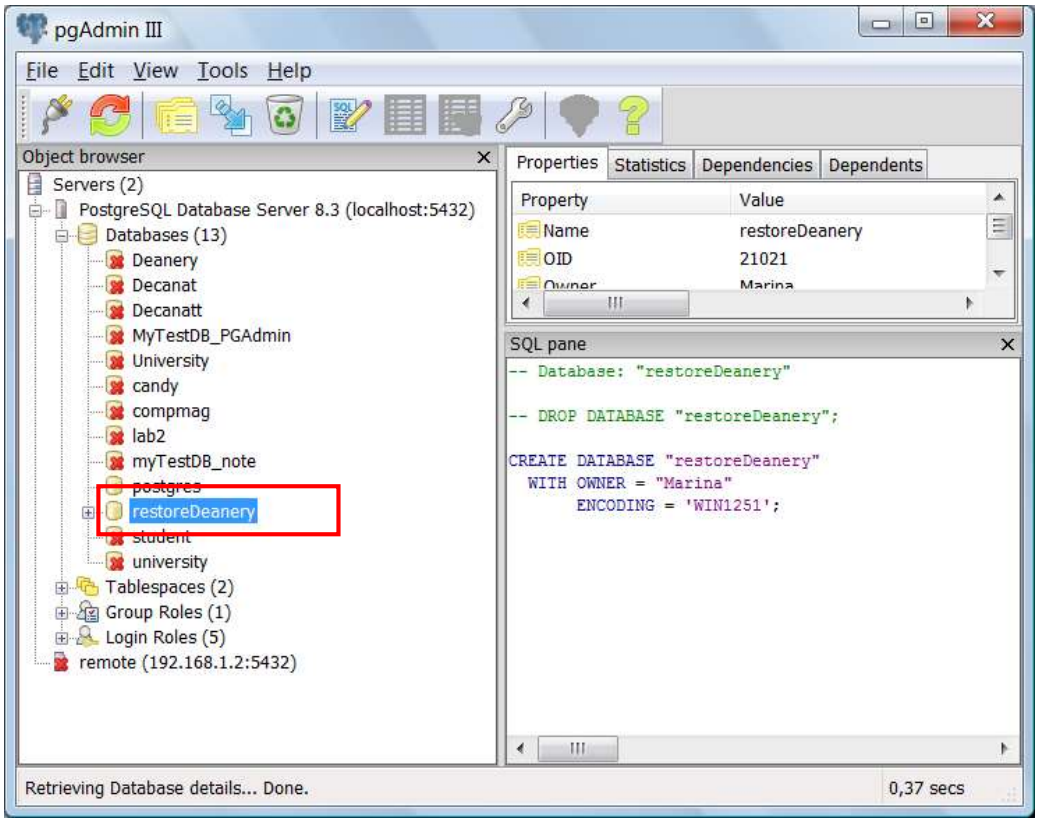


Рисунок 11.18 – Вікно Браузер об'єктів з відновленою БД

2. Потім необхідно створити ролі для користувачів секретар і декан (Рисунок 11.19, Рисунок 11.20).

3. Після цього виконується команда відновлення SQL-дампа БД (Рисунок 11.21).

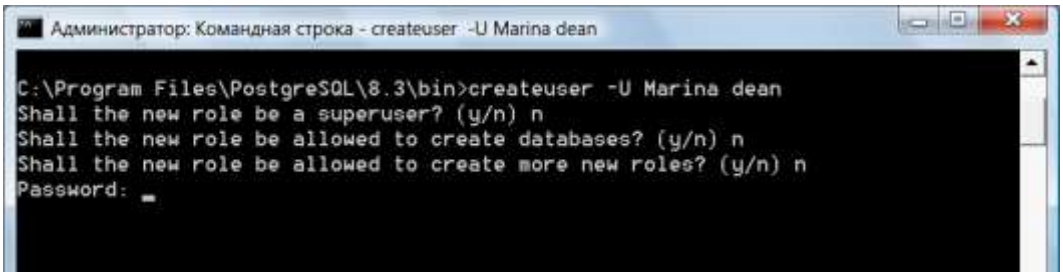


Рисунок 11.19 – Створення ролі Декан (dean)

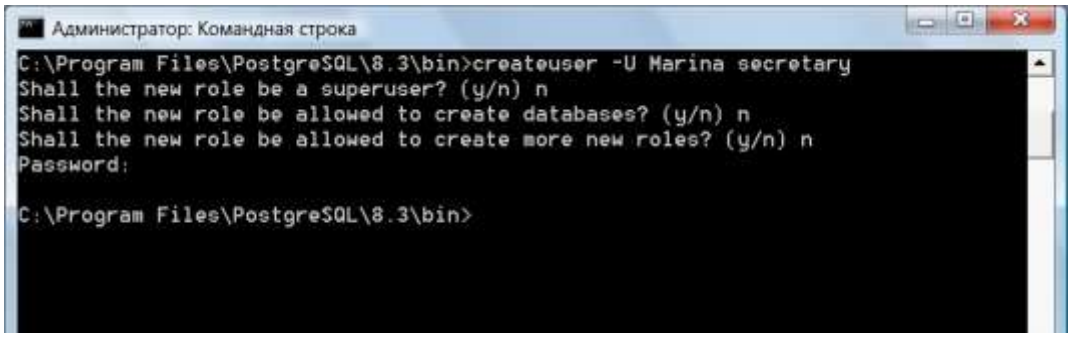


Рисунок 11.20 – Створення ролі Секретар (secretary)

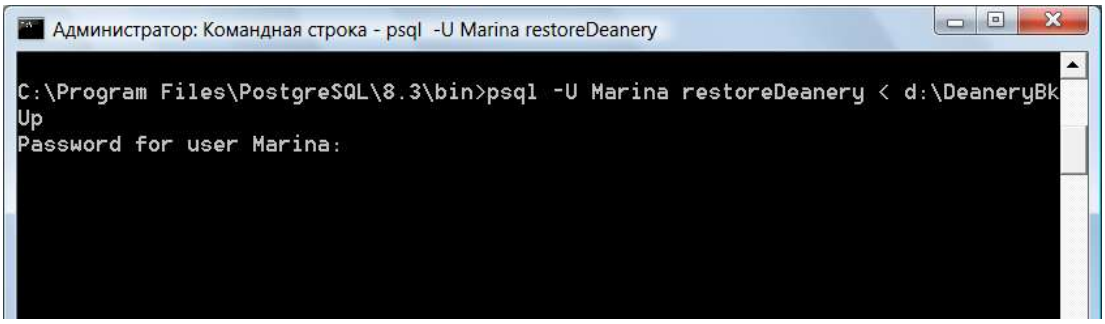


Рисунок 11.21 – Відновлення SQL-дампа БД

12 ОСОБЛИВОСТІ ВЗАЄМОДІЇ СУБД Access ТА PostgreSQL

Нижче наведено опис зручного способу копіювання таблиць Access в таблиці PostgreSQL, а також можливості підключення таблиць PostgreSQL до таблиць Access.

Попередня умова: Драйвер PostgreSQL ODBC повинен бути інстальований.

12.1 Підготовка з'єднання PostgreSQL з БД в Access

1. Вибрати Пуск -> Налаштування -> Панель управління -> Адміністрування -> Джерела даних (ODBC) (Рисунок 12.1).

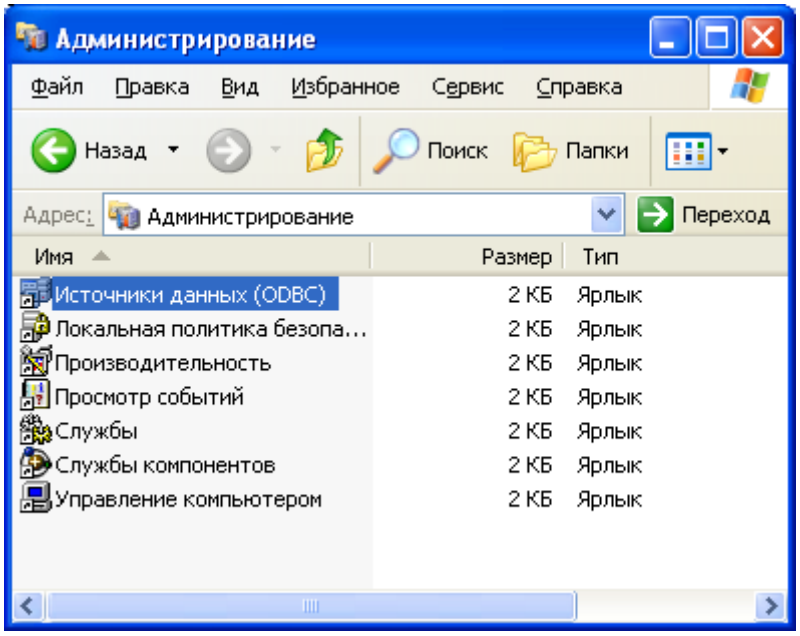


Рисунок 12.1 – Вікно Адміністрування

2. Створити користувальницький або системний DSN.

Порада: Для індивідуального комп'ютера можна вибрати користувальницький DSN або Системний DSN, а для комп'ютера в мережі організації краще вибрати Системний DSN, тому що він доступний всім користувачам даного комп'ютера та інших комп'ютерів мережі (Рисунок 12.2).

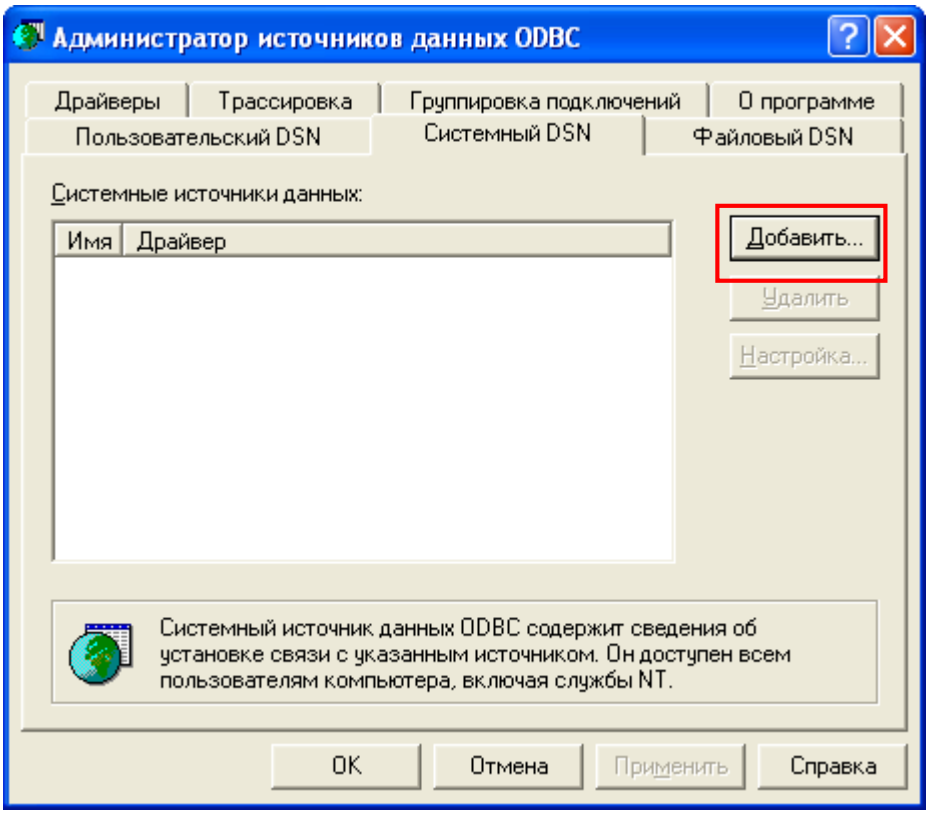


Рисунок 12.2 – Вікно Адміністрування джерела даних ODBC

3. Натиснути кнопку Додати і вибрати ім'я драйвера PostgreSQL (Рисунок 12.3).

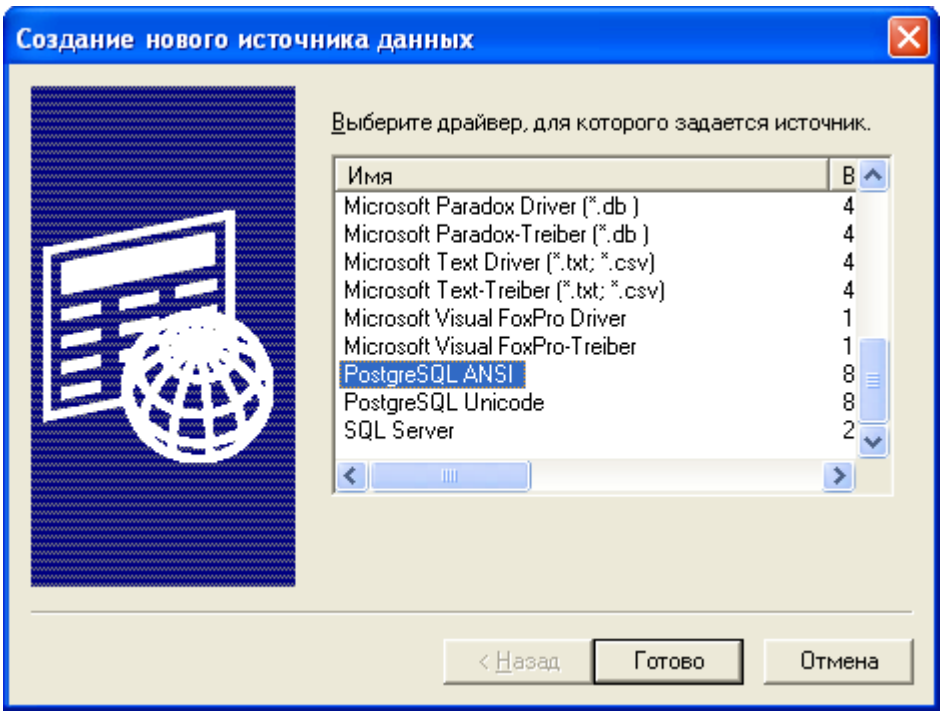


Рисунок 12.3 – Створення нового джерела даних

Примітка. Попередньо потрібно встановити ODBC драйвери для PostgreSQL.

4. Отримаємо вікно створення з’єднання, яке належить заповнити подібним чином (Рисунок 12.4).

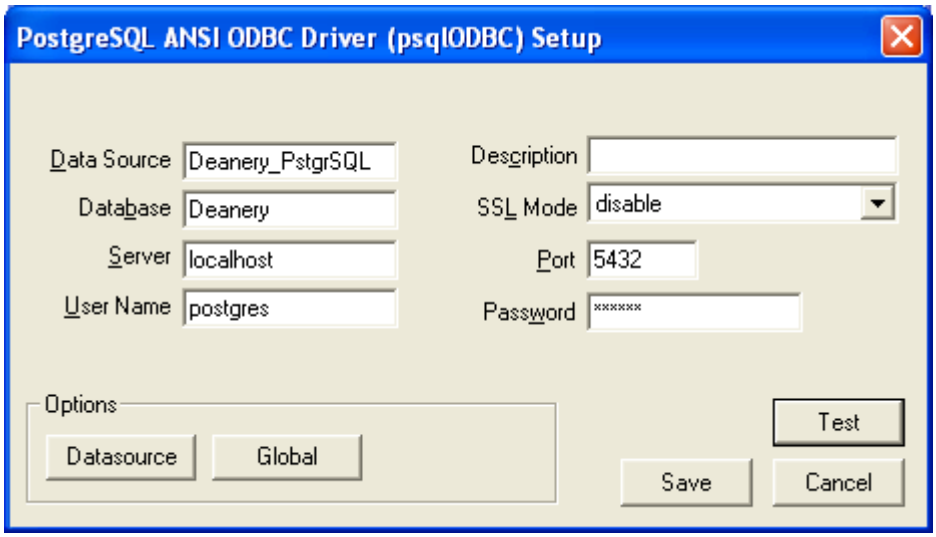


Рисунок 12.4 – Параметри підключення

5. Отримано новий Системний DSN з іменем Deanery_PstgrSQL (Рисунок 12.5)

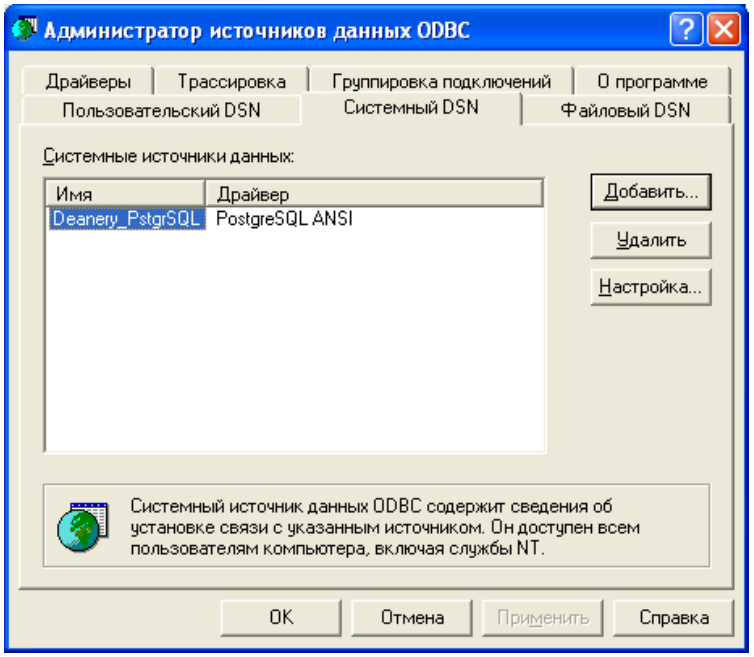


Рисунок 12.5 – Вікно Адміністратор джерел даних ODBC

Після чого натиснути OK.

12.2 Копіювання таблиць Access в таблиці PostgreSQL

СУБД Access спільно з драйвером ODBC надає зручні засоби експорту своїх таблиць в БД, створені в інших СУБД. Покажемо це на прикладі СУБД PostgreSQL

Попередні умови:

1. Драйвер PostgreSQL ODBC повинен бути інстальований і має бути створений для DSN користувача або Системний DSN;

2. В PostgreSQL повинна бути створена БД decanat, в яку будуть копіюватися таблиці

Необхідні дії наступні:

1. Перейти в Access и вибрати таблицю, що копіюється, наприклад, Students_log (Рисунок 12.6)

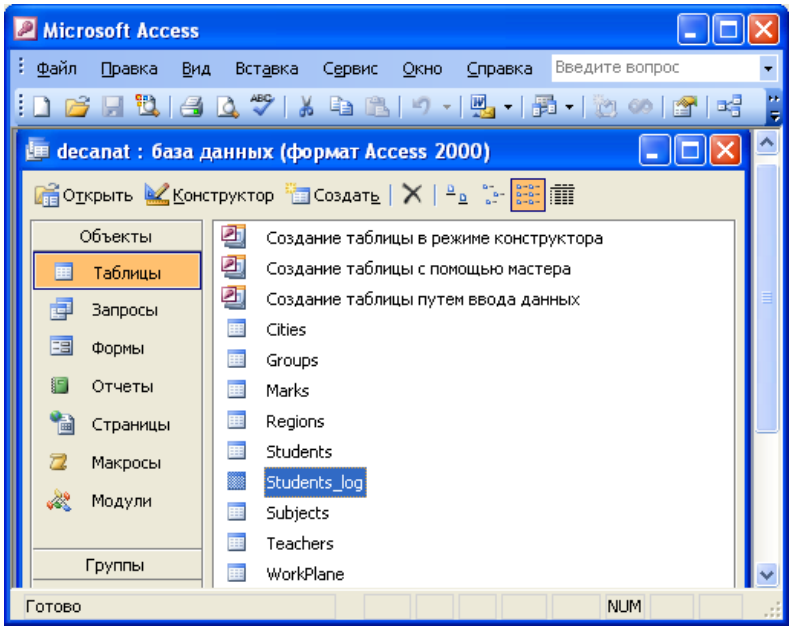


Рисунок 12.6 – БД decanat в Access

2. У головному меню виконати Файл->Експорт і в списку, що розкривається, Тип файлу вибрати База даних ODBC (ODBC DataBases). Тим самим пропонується експорт в БД, що мають з'єднання з ODBC драйвером (Рисунок 12.7).

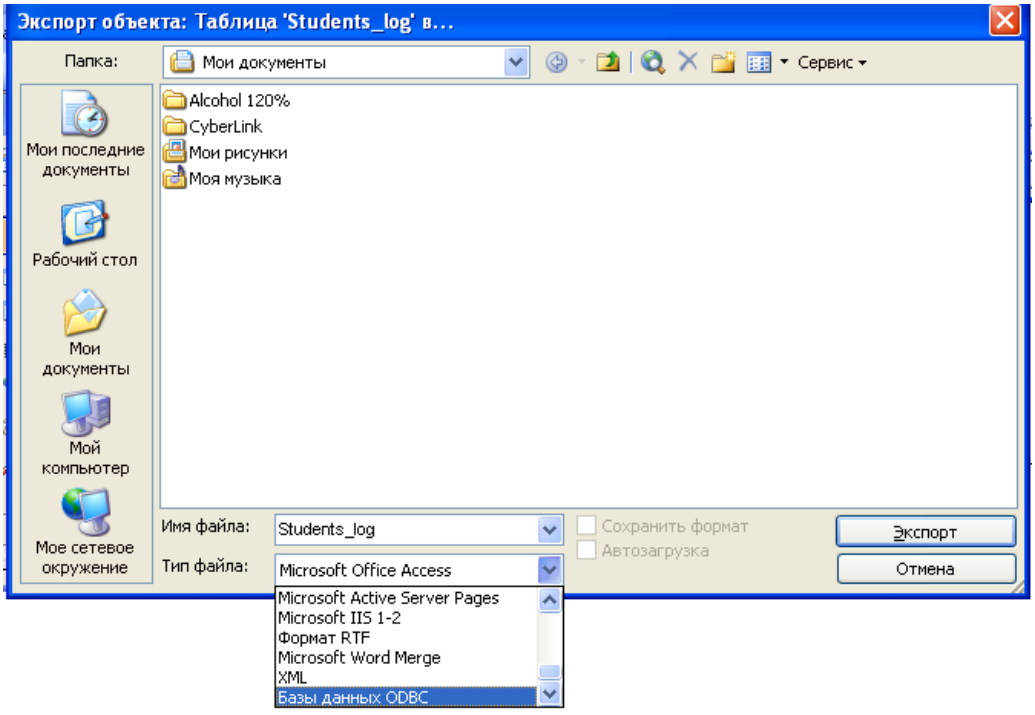


Рисунок 12.7 – Вікно експорту об'єктів

3. В отриманому вікні Експорт натиснути OK (Рисунок 12.8).

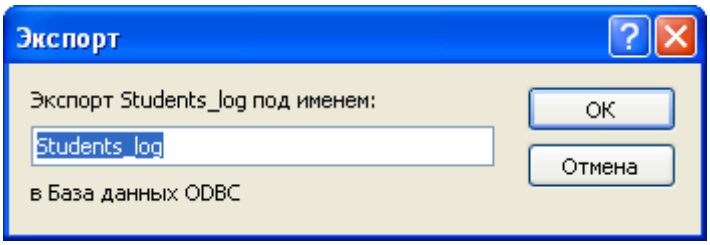


Рисунок 12.8 – Вікно Експорт

4. У відновленому вікні Вибір джерела даних виберіть Deanery_PstgrSQL і натисніть OK (Рисунок 12.9).

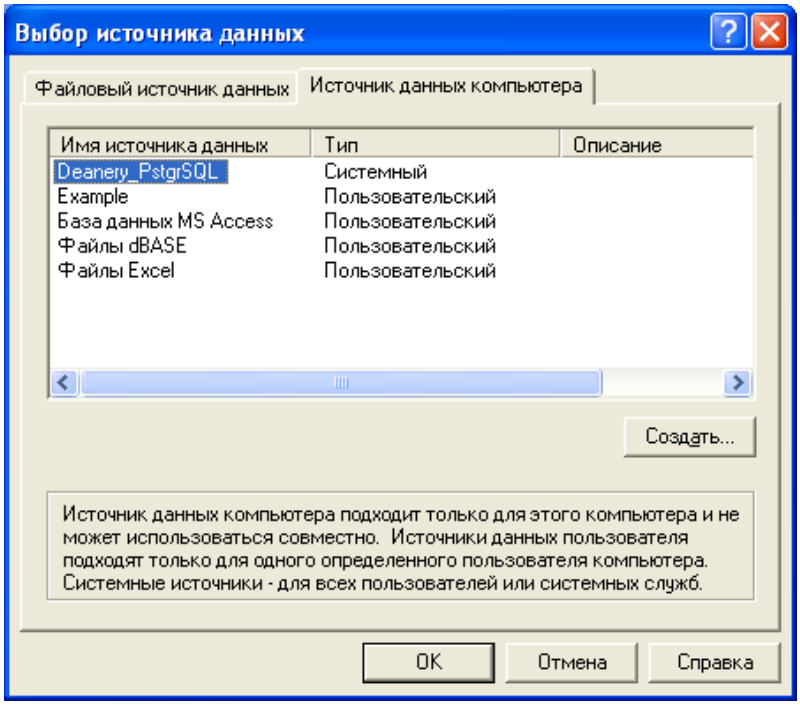


Рисунок 12.9 – Вікно вибору джерела даних

5. В PostgreSQL в БД Deanery буде створена таблиця Students_log.

Після чого можна продовжити подальшу роботу в PostgreSQL з отриманою БД Deanery і таблицею Students_log (Рисунок 12.10).

Примітка. Слід звернути увагу, що при такому копіюванні таблиць з БД Access, втрачається інформація про ключі і зв'язки між таблицями. Однак їх можна відновити засобами PostgreSQL. (Див. вище розділ 2.2)

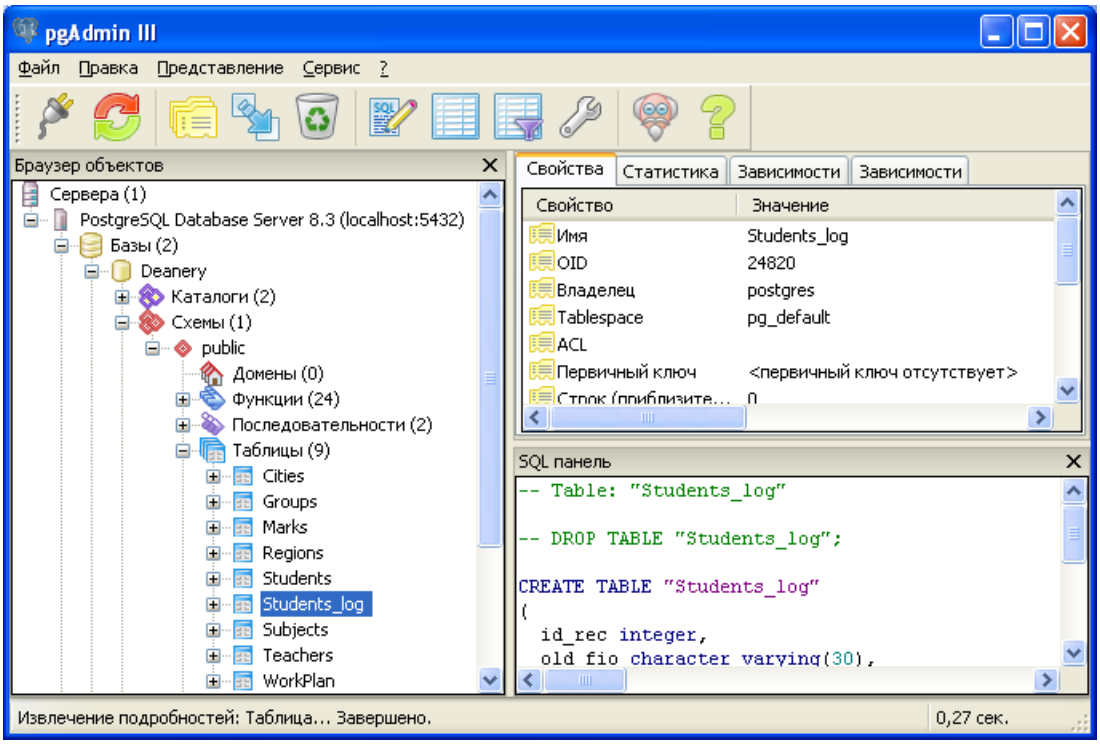


Рисунок 12.10 – Вікно Браузер об'єктів

12.3 Підключення таблиць PostgreSQL к БД в Access

СУБД Access спільно з драйвером ODBC надає зручні засоби підключити в БД Access таблиць, створені в інших СУБД. Покажемо це на прикладі С1УБД PostgreSQL.

Для підключення к БД Access таблиць PostgreSQL потрібно:

1. Виконати Файл->Зовнішні дані->Зв'язок з таблицями (Рисунок 12.11).

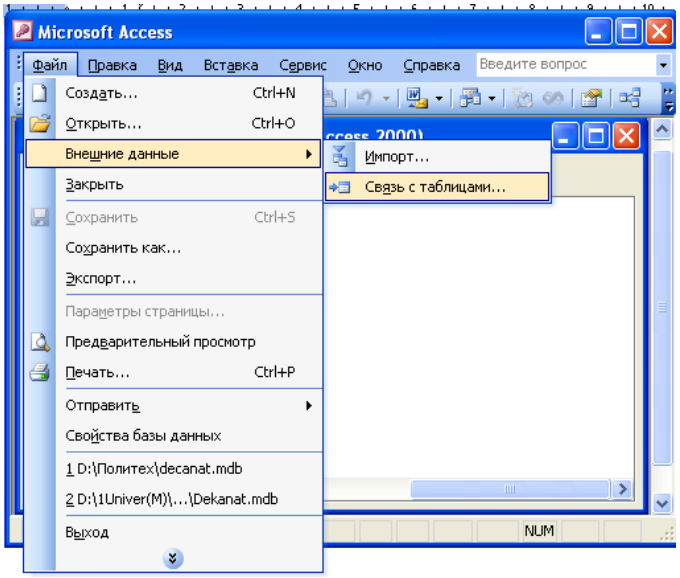


Рисунок 12.11 – Меню Файл Access

2. У списку Тип файлу оберіть База даних ODBC (ODBC DataBases) (Рисунок 12.12).

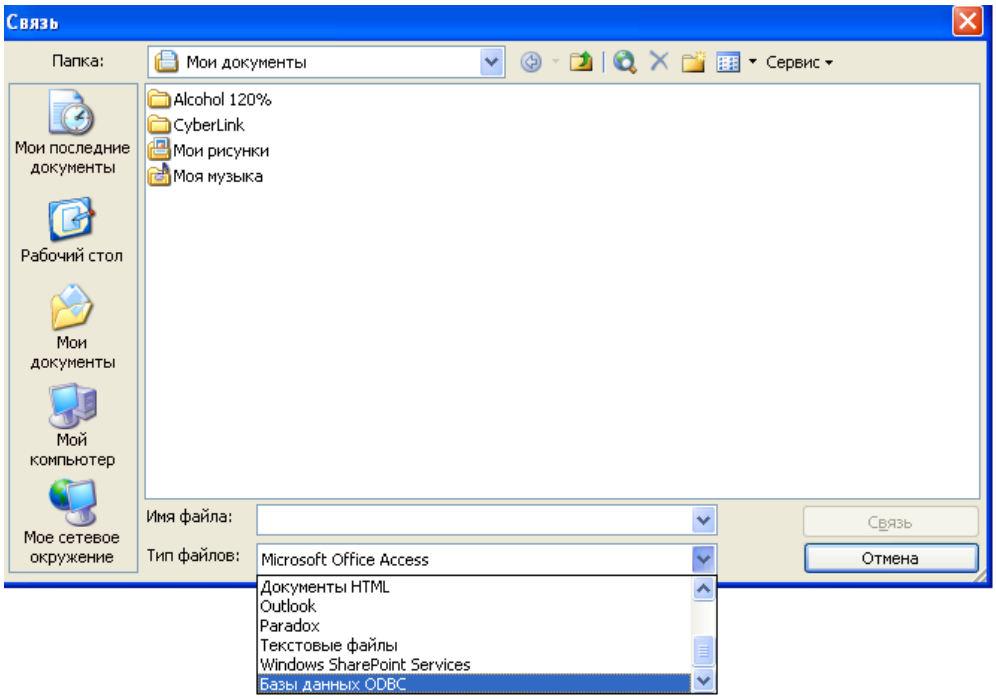


Рисунок 12.12 – Вікно Зв'язок

3. Оберіть джерело даних в PostgreSQL (Рисунок 12.13).

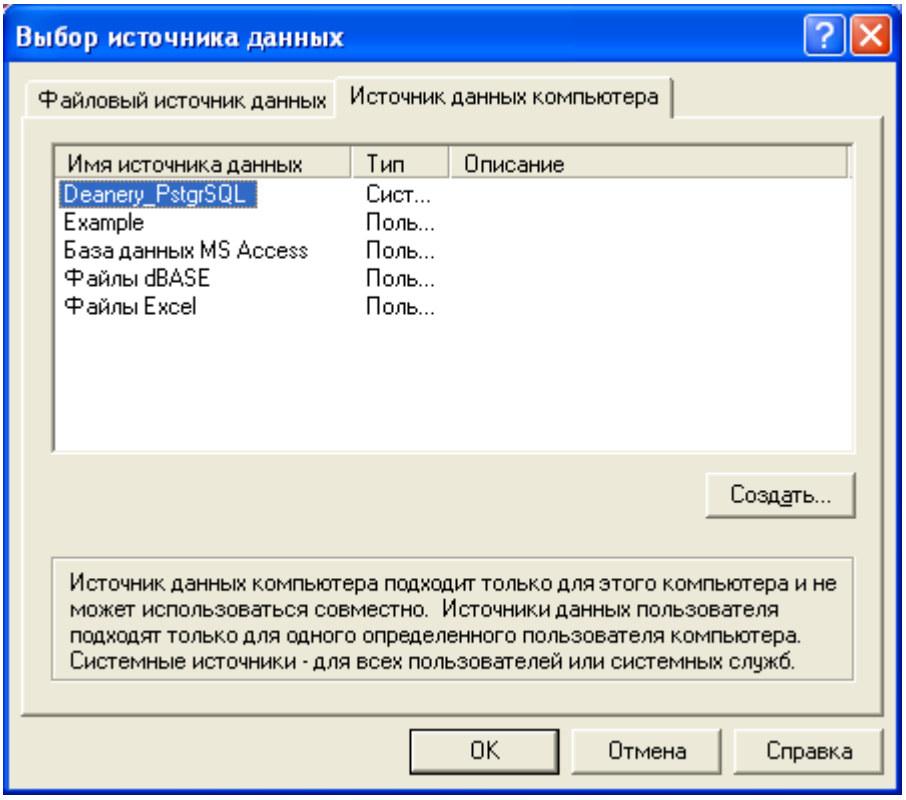


Рисунок 12.13 – Вікно вибору джерела даних

4. Оберіть потрібну таблицю, наприклад Work_Plan (Рисунок 12.14).

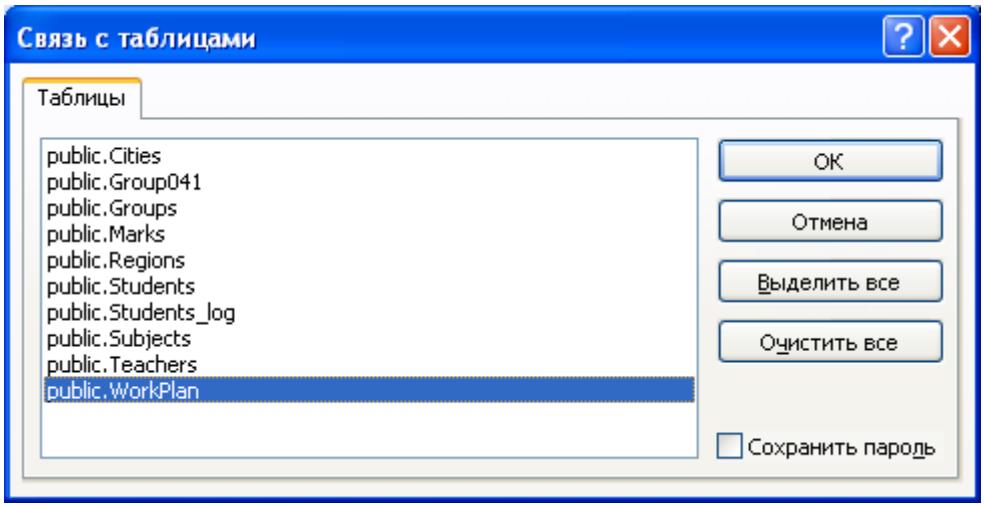


Рисунок 12.14 – Вікно вибору таблицю БД PostgreSQL

Примітка. Якщо для таблиці не вказано ключове поле, то з'явиться наступне вікно, в якому слід вказати ключове поле (однозначний індекс) таблиці, що під'єднується. Це поле id_rec (Рисунок 12.15).

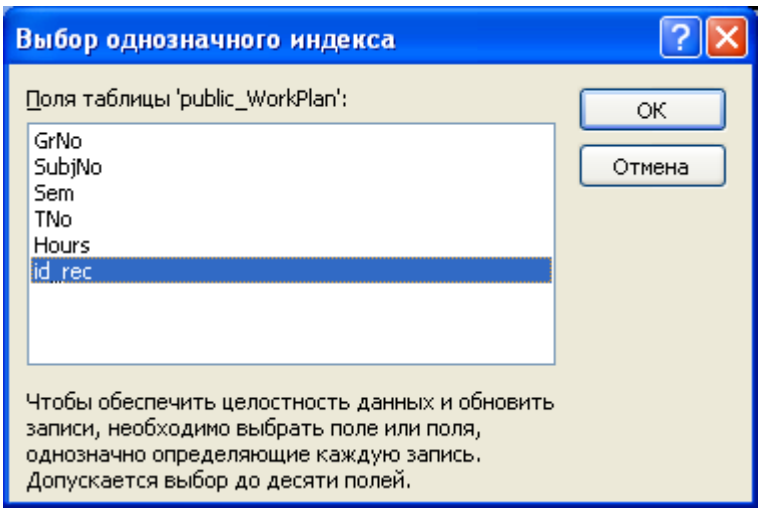


Рисунок 12.15 – Вікно вибору первинного ключа таблиці

5. В результаті в переліку таблиць проекту з'явиться приєднана (зв'язана) таблиця public_WorkPlan (Рисунок 12.16). Вона може бути використана для цілей проекту, як і власна таблиця проекту.

Примітка. Слід звернути увагу на особливості імені таблиці public_WorkPlan. Це викликано тим, що в БД decanat вже існує своя таблиця WorkPlan.

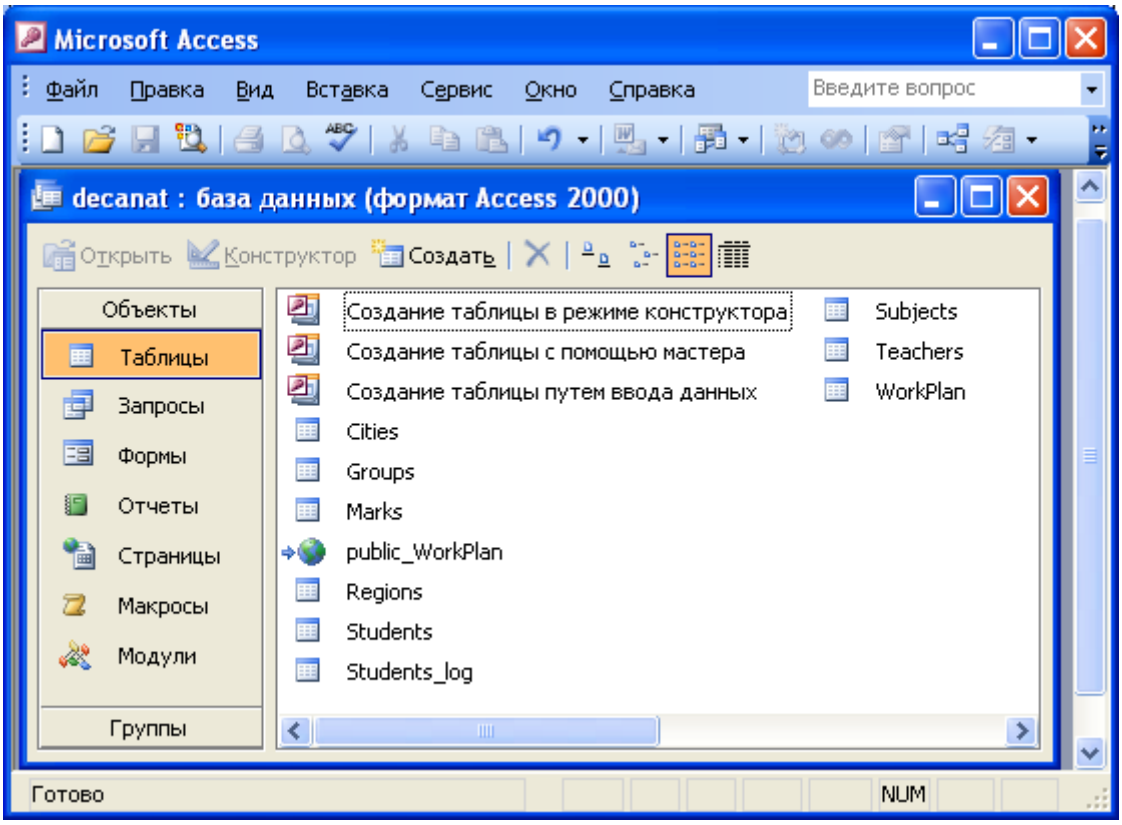


Рисунок 12.16 – БД decanat з приєднаної таблиці public_WorkPlan

13 ХАРАКТЕРИСТИКИ PostgreSQL

http://www.scribd.com/doc/4951927/PostgreSQL-7
http://postgresql.ru.net/docs/overview.html#langs

13.1 Розмір бази даних

Максимальний розмір БД	не обмежено
Максимальний розмір таблиці	32 TB
Максимальний розмір запису (рядка) таблиці	1.6 TB
Максимальний розмір поля в запису (рядку)	1 GB
Максимальна кількість записів (рядків) в таблиці	не обмежено
Максимальна кількість полів (колонок) в таблиці	250 - 1600 в залежності від типу даних в колонці
Максимальна кількість індексів на таблицю	не обмежено

13.2 Підтримувані вбудовані типи даних

Числові типи

smallint	коротке 2-х байтове ціле
integer	звичайне 4-х байтове ціле
bigint	велике 8-ми байтове ціле
decimal	дійсне з фіксованою точкою
numeric	дійсне з фіксованою точкою
real	дійсне з плаваючою точкою
double precision	дійсне з плаваючою точкою та подвійною точністю
serial	ціле с автозбільшенням (автоінкрементом)
bigserial	велике ціле с автозбільшенням (автоінкрементом)

Грошові типи

money	для зберігання грошових значень

Символьні типи

character varying(n), varchar(n)	рядок змінної довжини з обмеженням
character(n), char(n)	рядок фіксованої довжини
text	рядок змінної необмеженої довжини

Бінарні (двійкові) типи

bytea	бінарний рядок змінної довжини

Логічні типи

boolean	TRUE або FALSE

Дата та час

timestamp [(p)] [без часового поясу]	дата та час
timestamp [(p)] з часовим поясом	дата та час з часовим поясом
interval [(p)]	інтервал часу
date	тільки дата
time [(p)] [без часового поясу]	тільки час
time [(p)] з часовим поясом	тільки час з часовим поясом

Геометричні типи

point	Точка на площині (x,y)
line	Невидима лінія
lseg	Видимий відрізок ((x1,y1),(x2,y2))
box	Чотирикутник ((x1,y1),(x2,y2))
path	Замкнутий багатокутник (схожий на полігон) ((x1,y1),...)
path	Ломана лінія [(x1,y1),...]
polygon	Полігон (схожий на замкнутий багатокутник) ((x1,y1),...)
circle	Коло (x,y),r (центр та радіус)

Типи для адрес комп'ютерних мереж

cidr	IPv4 або IPv6 мережа
inet	IPv4 або IPv6 хост та мережа
macaddr	MAC адрес

Бітові рядки

bit [(n)]	бітовий рядок фіксованої довжини
bit varying [(n)]	бітова рядок змінної довжини

Типи для пошуку тексту

tsquery	запит на пошук тексту
tsvector	список для пошуку тексту

UUID тип

uuid	універсальний унікальний ідентифікатор

XML типи

xml	дані XML

Крім цього набору типів, PostgreSQL надає можливість створення списків (тип ENUM), масивів типів, складових типів на зразок структур в мові C, а також має типи для унікальної ідентифікації об'єктів (OID) і псевдотипів для серверних процедур.

Типи даних, створені користувачем

За допомогою команди CREATE TYPE користувачі можуть створювати нові типи даних для своїх потреб.

14 АДМІНІСТРУВАННЯ PostgreSQL

За замовчуванням PostgreSQL автоматично запускається разом із запуском операційної системи і споживає значний ресурс оперативної пам'яті. Якщо PostgreSQL використовується зрідка, то має сенс дезактивувати цю службу або сервіс і активувати її в міру необхідності.

В 9-ой версії PostgreSQL.9 і далі цю службу можна активізувати та деактивувати, використовуючи Диспетчер задач (Рисунок 14.1).

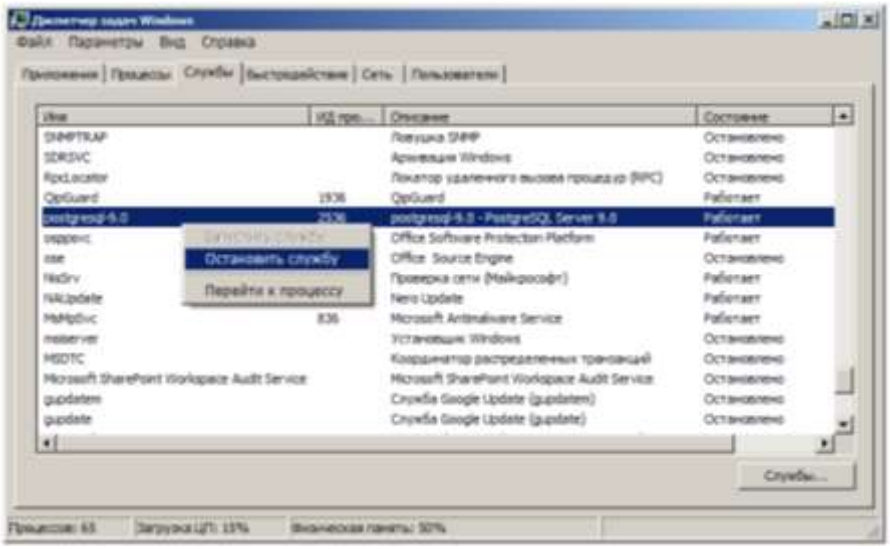


Рисунок 14.1 – Зупинка служби PostgreSQL

З різних питань адміністрування і, зокрема, управління користувачами див. [11].

ЛІТЕРАТУРА

1. http://postgresql.ru.net/manual/tutorial-createdb.html
2. http://postgresql.ru.net/postgis/ch04_5.html
3. http://postgresql.ru.net/docs/overview.html#index
4. http://www.scribd.com/doc/4846380/-GIN-GiST-PostgreSQL
5. http://postgresql.ru.net/manual/gist.html
6. http://postgresql.ru.net/manual/gin.html
7. http://postgresql.ru.net/manual/sql-createfunction.html
8. http://www.postgresql.org/docs/8.3/static/plpgsql.html
9. http://www.postgresql.org/docs/8.1/static/sql-grant.html
10. http://www.sai.msu.su/~megera/postgres/talks/gist_tutorial.html
11. http://www.opennet.ru/links/info/685.shtml
12. http://www.citforum.ru/database/postgres/what_is/
13. http://www.nestor.minsk.by/kg/2003/48/kg34802.html
14. http://ru.wikipedia.org/wiki/PostgreSQL#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D0.B8
15. Native Queries – How to call native SQL queries with JPA https://www.thoughts-on-java.org/jpa-native-queries/
16.