

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

## Тема: «Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL»

Виконав: студент ІІI курсу

ФПМ групи КВ-81

Прокопчук М.О.

Перевірив:

Київ – 2020

**Завдання**

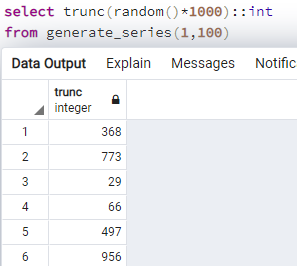
*Загальне завдання* роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

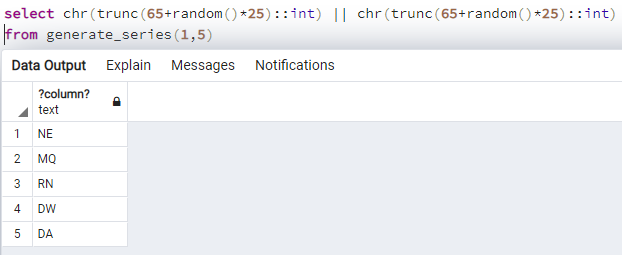
*Деталізоване завдання*:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!

Приклад генерації 100 псевдовипадкових чисел:



Приклад генерації 5 псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності (foreign key).

1. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.
2. Програмний код організувати згідно шаблону Model-View-Controller(MVC).  Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: http://initd.org/psycopg/docs/usage.html)

*Вимоги до інтерфейсу користувача*

Використовувати консольний інтерфейс користувача.

*Вимоги до інструментарію*

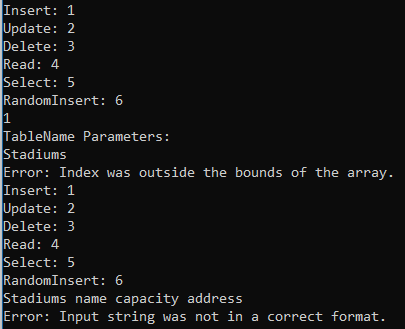
 Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

**Пункт №1**

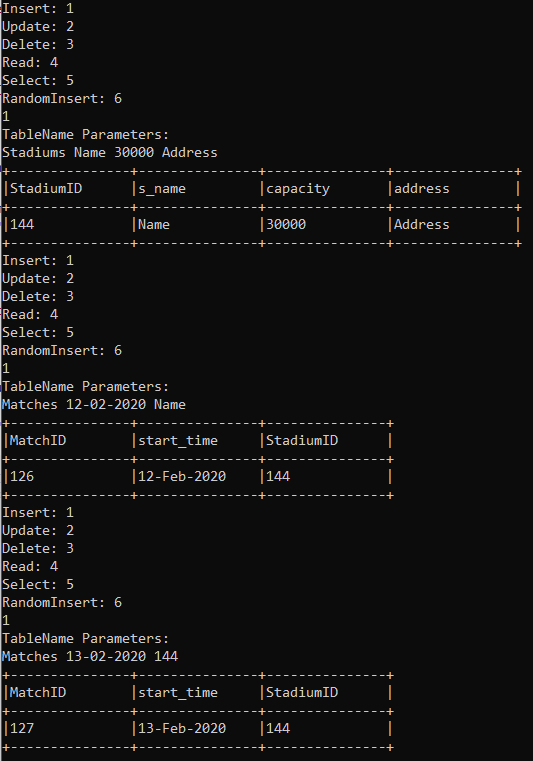
У даній програмі є два види помилок:

Перша коли недостатньо параметрів для запросу.

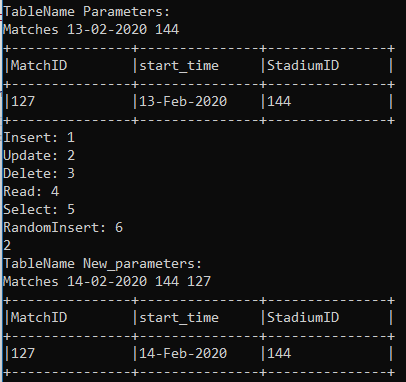
Друга коли парсер не може конвертувати string в потрібний тип.

****

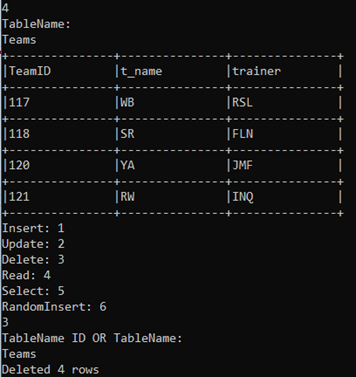
Insert:

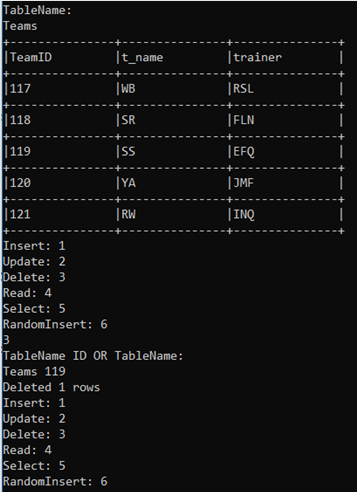
****

Update:

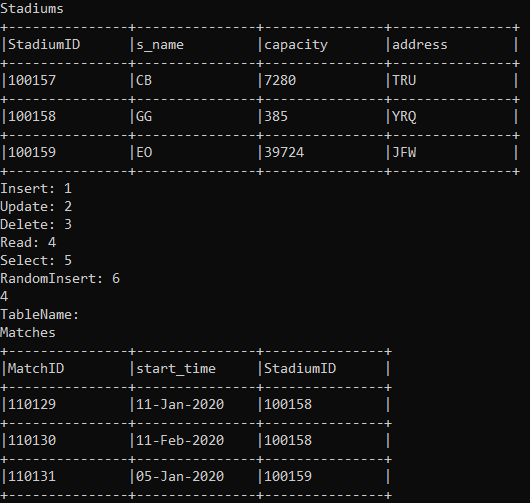
****

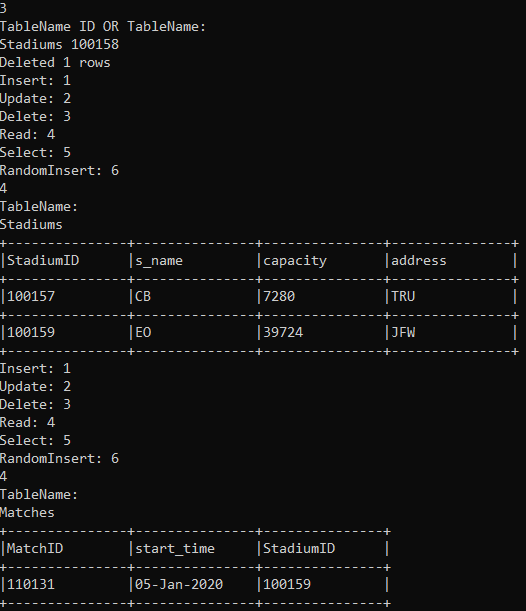
Delete & Read:

Вилучення із заданої таблиці:

****

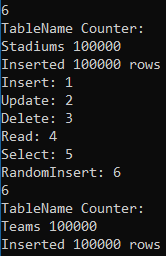
Вилучення рядку з батьківської таблиці на який посилається рядки дочірньої:

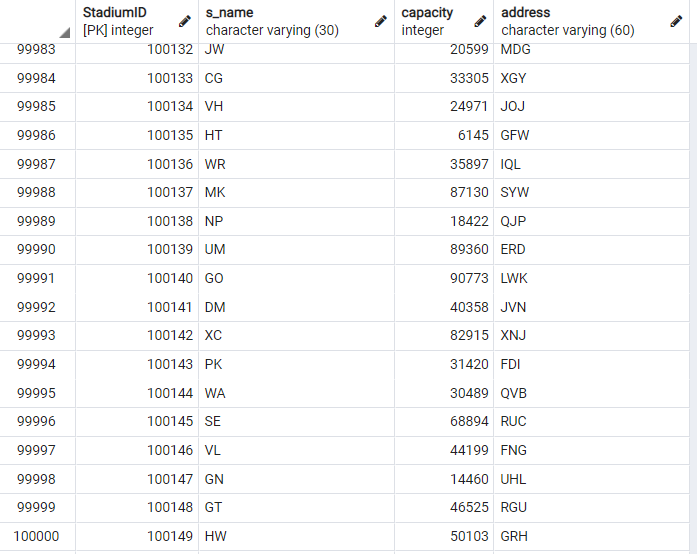
****

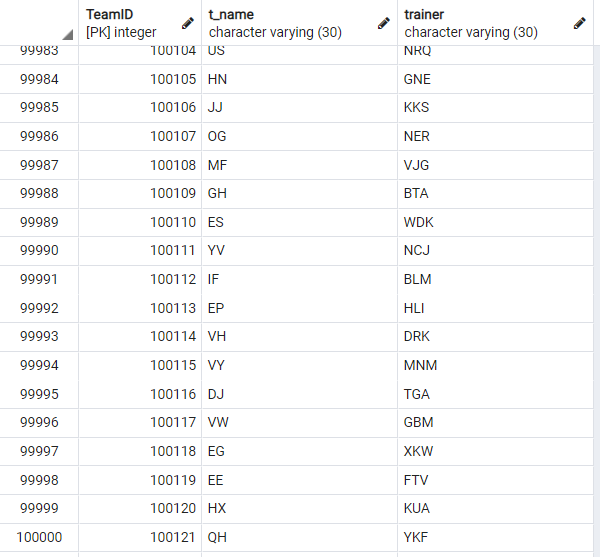
****

**Пункт №2**

Приклад роботи генерації псевдовипадкових рядків в таблицях Stadium і Team:

****

****

****

Запити:

INSERT INTO public."Stadiums" ("s\_name", "capacity", "address") SELECT chr(trunc(65 + random() \* 25) || chr(trunc(65 + random() \* 25), round(100 + random() \* 99900), chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) FROM generate\_series(1, counter)

INSERT INTO public."Teams" ("t\_name", "trainer") SELECT chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), chr(trunc(65 + random() \* 25)::int)

|| chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) FROM generate\_series(1, counter)

INSERT INTO public."Matches" ("start\_time", "StadiumID") SELECT timestamp '2020-01-10' + random() \* (timestamp '2040-01-01' - timestamp '2020-01-01'), SELECT "StadiumID" FROM public."Stadiums" ORDER BY random() limit 1) FROM generate\_series(1, counter)

INSERT INTO public."Games" ("TeamID", "MatchID") SELECT (SELECT "TeamID" FROM public."Teams" ORDER BY random() limit 1), (SELECT "MatchID" FROM public."Matches" ORDER BY random() limit 1) FROM generate\_series(1, counter)

INSERT INTO public."Goals" ("minute", "TeamID", "MatchID") SELECT round(random() \* 90), (SELECT "TeamID" FROM public."Teams" ORDER BY random() limit 1), (SELECT "MatchID" FROM public."Matches" ORDER BY random() limit 1) FROM generate\_series(1, counter)

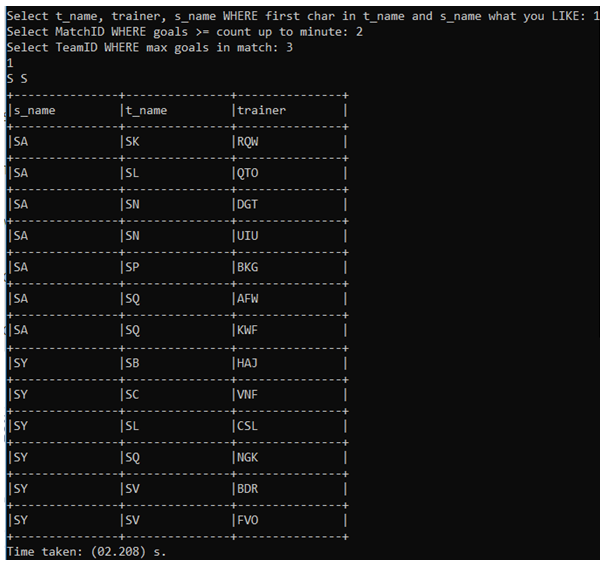
**Пункт №3**

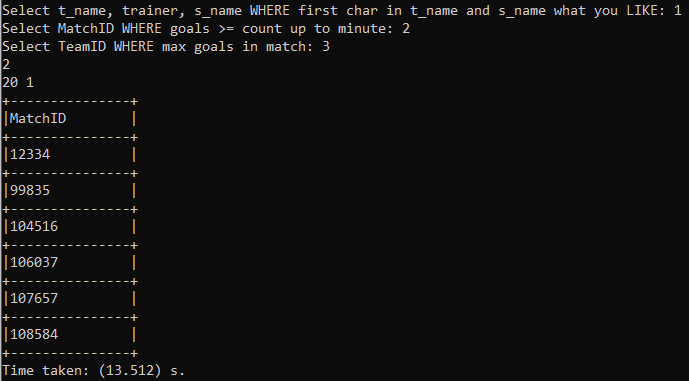
У програмі реалізовано декілька запитів пошуку:

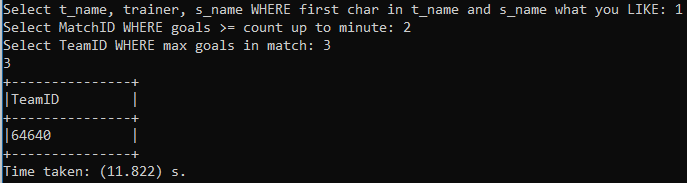
Перший запит повертає таблицю де початок імені команди і початок назви стадіону, на якій грала команда співпадають з тими що ввів користувач. (Було використано запит LIKE)

Другий запит повертає таблицю команд, котрі за відведений час у хвилинах забили відповідну кількість голів.

Третій запит повертає таблицю матчей, які мають найбільшу кількість голів за матч.







Вибірка велась з таблиць які мають 100000 записів кожна. Але для того щоб виводимі таблиці були не громісткі, був поставлений ліміт в запитах.

Запити:

SELECT DISTINCT public."Stadiums"."s\_name", public."Teams"."t\_name", public."Teams"."trainer" FROM public."Teams" INNER JOIN public."Games" ON public."Teams"."TeamID" = public."Games"."TeamID" INNER JOIN public."Matches" ON public."Matches"."MatchID" = public."Games"."MatchID" INNER JOIN public."Stadiums" ON public."Stadiums"."StadiumID"=public."Matches"."StadiumID"

WHERE "s\_name" LIKE 's\_ch%' AND "t\_name" LIKE 't\_ch%';

SELECT DISTINCT "MatchID" FROM public."Goals" AS "g"

WHERE (SELECT Count(\*) FROM public."Goals" WHERE minute < minute AND "MatchID" = "g"."MatchID") > count\_goals;

SELECT DISTINCT "TeamID" from public."Goals" AS "m" WHERE (SELECT Count(\*) FROM public."Goals" WHERE public."Goals"."MatchID" = "m"."MatchID" AND public."Goals"."TeamID" = "m"."TeamID") = SELECT max(y.m) from (select count(\*) AS m

FROM public."Goals" group by public."Goals"."MatchID",

public."Goals"."TeamID") AS y)

**Пункт №4**

Код програми можна знайти за посиланням [тут](https://github.com/FaceandControl/DataBaseLabs) (репозиторій github).

