**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни “Бази даних”

спеціальність 121 – Програмна інженерія

на тему: Інформаційна система матчів

(назва теми)

|  |  |  |
| --- | --- | --- |
| **Студент групи** КП-92 | **Остапенко Іван Петрович**  (ПІБ) | \_\_\_\_\_\_\_\_\_   (підпис) |
| **Викладач**  **к.т.н, доцент кафедри СПіСКС** | **Петрашенко А.В.** | \_\_\_\_\_\_\_\_\_   (підпис) |

Захищено з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2020

**Анотація**

У курсовій роботі розроблено та документально описано інформаційну систему матчів. Програмний застосунок включає в себе можливості доступу до бази даних за допомогою консольного користувацького інтерфейсу; створення статистик турнірів, команд у текстовому та графічному вигляді; виведення розкладу матчів; введення даних про команди та турніри; їх генерація, а також генерація матчів та їх результатів. Реалізовано механізми реплікації, резервування та відновлення даних.

**Зміст**

[1 Аналіз інструментарію для виконання курсової роботи 5](#_Toc59118383)

[1.1 Аналіз СУБД 5](#_Toc59118384)

[1.2 Аналіз бібліотек та фреймворків 5](#_Toc59118385)

[2 Структура бази даних 7](#_Toc59118386)

[3 Опис програмного забезпечення 8](#_Toc59118387)

[3.1 Загальна структура програмного забезпечення 8](#_Toc59118388)

[3.2 Опис модулів програмного забезпечення 8](#_Toc59118389)

[3.3 Опис основних алгоритмів роботи 9](#_Toc59118390)

[4 Аналіз функціонування засобів реплікації 10](#_Toc59118391)

[5 Аналіз функціонування засобів резервування/відновлення 10](#_Toc59118392)

[6 Аналіз результатів підвищення швидкодії запитів 11](#_Toc59118393)

[7 Опис результатів аналізу предметної галузі 12](#_Toc59118394)

[ДОДАТКИ 16](#_Toc59118395)

[ДОДАТКИ А. Консольний інтерфейс 28](#_Toc59118396)

[ДОДАТКИ Б. Створені діаграми 32](#_Toc59118397)

[ДОДАТКИ В. Фрагменти коду 35](#_Toc59118398)

**Вступ**

Починаючи з середини ХХ століття у світі зростає тенденція популяризації спорту. Як наслідок, зросла кількість турнірів. Були створені міжнародні змагання, наприклад, чемпіонати світу та повернули традицію проведення Олімпійських ігор. Також з’являлися різноманітні федерації, ліги, комітети, меценати, що організовують турніри для різних видів спорту, різних вікових категорій, різних рівнів учасників і так далі.

На сьогодні існує безліч організацій, що для проведення змагання, використовують спеціальні сервіси. Однак у країнах, що розвиваються, на змаганнях часто відсутнє програмне забезпечення, щодо обробки та збереження даних учасників, результатів матчів, тощо. Через відсутність відповідних сервісів виникають проблеми: складності отримання доступу до розкладу ігор, даних учасників; відсутність автоматизованого аналізу результатів; складність у звітуванні турнірів, створення протоколів ігор; визначення кращих гравців турніру, сезону та багато інших. Особливо ці проблеми присутні у турнірах, де відбувається реєстрація команд та формування розкладу матчів на місці події.

Вирішення цих проблем є створення інформаційного комплексу, що обслуговує змагання, в який можуть входити: сервер для зберігання та оброблення інформації, застосунки для організаторів, суддів, учасників, вболівальників.

Метою курсової роботи є набуття практичних навичок розробки та оформлення програмного забезпечення, що взаємодія з реляційними базами даних та вирішує наступні завдання: формує розклад матчів, зберігає їх результат; виводить статистику для турнірів та команд.

# 1 Аналіз інструментарію для виконання курсової роботи

## 1.1 Аналіз СУБД

Розглянемо наступні найбільш популярні реляційні СУБД [1]: Oracle, MySQL, MicrosoftSQL Server, PostgreSQL. Частина функціоналу Oracle Database[2] та MicrosoftSQL Server[3] є небезкоштовною. Для вибору РСУБД серед безкоштовних MySQL та PostgreSQL було використано метод морфологічного аналізу, що наведений у Додатку 1. Згідно з яким обрано РСУБД PostgreSQL для розробки даного програмного застосунку.

Основні переваги PostgreSQL [4]:

* РСУБД з відкритим кодом
* наявність документації
* підтримка різноманітних індексів
* підтримка більшості основних функцій стандарту SQL (SQL:2016) [5]

1.2 Аналіз бібліотек та фреймворків

Для вибору мови програмування методом морфологічного аналізу проведений вибір з наступних альтернатив: С++, С#, Python (див. Додаток 2)

Згідно аналізу обрано мову програмування Python для розробки даного програмного застосунку.

Основні переваги Python [6]:

* відкритий код
* чистий синтаксис (для виділення блоків слід використовувати відступи)
* стандартний дистрибутив має велику кількість модулів

Модулі Python, що використовуються:

* numpy[7] — розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами.
* psycopg2[8] — адаптер бази даних PostgreSQL для мови програмування Python.
* SQLAlchemy[9] — це набір інструментів Python SQL та ORM, що надає розробникам додатків повну потужність та гнучкість SQL.
* matplotlib[10] — бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою.

# 2 Структура бази даних

База даних складається з 5 таблиць:

* tournaments — таблиця, що містить дані про турніри: назву та дату події
* teams — таблиця, що мітить дані про команди-учасників турніру: назву час реєстрації та посилання на турнір в якому беруть участь (FK)
* match\_schedule — таблиця подій запланованих матчів, що містить дані про команди-учасників (FK), запланований час початку, майданчик на якому відбудеться гра (FK)
* match\_results — таблиця результатів ігор, що має посилання на запланований матч (FK), запис у цій таблиці означає, що матч відбувся. Містить дані про фактичний початок, кінець матчу, набрані очки команд.
* рlaygrounds — таблиця переліку майданчиків на турнірі, містить назву та посилання на турнір (FK)

Особливості зв’язків:

* У кожного турніра є свій перелік майданчиків (зв’язок 1 — М між таблицями tournaments та teams)
* Одна команда не може взяти участь у декількох турнірах. Обмеження пов’язано тим, що є види турнірів, де реєстрація проходить щоразу (зв’язок 1 — М між таблицями tournaments та рlaygrounds)
* Матч може бути зіграний лише один раз, або не зіграний взагалі (зв’язок 1 — 0..1 між таблицями match\_schedule та match\_results)

Структура бази даних представлена у Додатку 3

# 3 Опис програмного забезпечення

## 3.1 Загальна структура програмного забезпечення

Під час розробки програмного забезпечення було використано наступні архітектурні рішення:

* архітектурний шаблон MVC (модель-представлення-контролер)
* міст (для “контролерів” та “сховищ” сутностей “турніру” та “команда”)
* декоратор (для заміру часу роботи методів та обробки помилок)

Діаграма класів представлена в Додатку 4.

## 3.2 Опис модулів програмного забезпечення

Розроблені модулі програмного забезпечення за архітектурою MVC можна розділити на:

* Models:, StorageEntity, GenerateStorage, SearchStorage та Database.
  + Сутності: Tournament, Team - представляють дані з якими оперують сховища, реалізовані за допомогою ORM SQLAlchemy, що спрощує синтаксично виконання CRUD-операцій
  + Сховища — представляють інтерфейс для взаємодії з таблицями в базі даних:
    - EntityStorage призначений для виконання CRUD-операцій сутностей Tournanament та Team в таблицях tournamnents та teams відповідно.
    - GenerateStorage призначений для генерації даних. Має публічний метод *generate\_tournament* для генерування турніру з заданою кількістю команд, ігрових майданчиків, матчів, зіграних матчів та приватні методи, що генерують відповідні записи в таблицях.
    - SearchStorage призначений для виконання запитів отримання статистик турніру, команд; розклад матчів для турніру і/або з використанням фільтром за часом
    - Database — контейнер для сховищ перелічних вище.
  + Представлення представлений модулем View, що надає можливості вводу/виводу даних з консолі
  + Контролери:
    - AСontroller — абстрактний клас, що забезпечує єдиний інтерфейс для контролерів
    - EntityController, GenerateController, SearchController — контролери, що пов'язують View та сховища EntityStorage, GenerateStorage, SearchStorage відповідно
    - Controller — контейнер для контролерів перелічених вище.

А також модуль bencmark, що призначений для заміру часу виконання операцій та відловлювання/обробки помилок.

## 3.3 Опис основних алгоритмів роботи

При ініціалізації програми створюється підключення до серверу.

Програмний застосунок має консольний інтерфейс і складається з множини меню, що можуть переходити одне в одне. Програма має наступні меню: “tournament menu”, “team menu”, “search menu”, “generate menu”.

Діаграма станів програми представлена у Додатку 5.

Пункти меню “tournament menu”, “team menu” призначені для виконання CRUD-операцій.

Пункти меню “search menu” призначені для отримання статистики турнірів, команд, розкладу матчів.

Пункти меню “generate menu” призначений для генерації турнірів з заданою кількістю команд-учасників, матчів та зіграних матчів з можливістю відміни операції (rollback).

При завершені програми відбувається фіксація внесених даних (commit) та підключення до серверу закривається.

# 4 Аналіз функціонування засобів реплікації

Для реалізації реплікації було створено кластер баз даних “slave” за допомогою утиліти pg\_createcluster на тому ж ПК, що і первина БД (порт за замовчуванням 5432), на порті 5434. Створено фізичну потокову реплікацію за допомогою утиліти pg\_basebackup.

Демонстрація наявності потокової реплікації на первинній БД відображено у Додатку 6

Демонстрація коректної роботи реплікації відображено у Додатку 7.

# 5 Аналіз функціонування засобів резервування/відновлення

Для резервування використовується утиліта pg\_dump.

Для автоматизації резервування було написано скрипт, що робить резервне копіювання таблиці “tournaments”:

|  |
| --- |
| **dump\_tournaments.sh** |
| timestamp=$(date +%s)  pg\_dump -U postgres tournaments > /var/my\_dir/tournaments\_dump\_$timestamp |

Даний скрипт можна використати для планувальника завдань (наприклад, для *cron*, перемістивши скрипт у папку */etc/cron.daily/*), для періодичного виклику.

Для відновлення бази даних використовується утиліта psql:

psql -U postgres -d database\_name < file\_name\_dump

де *database\_name* - назва бази даних в яку відновлятся дані з файлу *file\_name\_dump*

Наприклад:

psql -U postgres -d tournaments < /var/my\_dir/tournaments\_dump\_1607956779

Для заміру роботи часу було використана утиліта time. Створення dump розміром 1,8MiB зайняло 189ms. Відновлення 3100 ms. Демонстрація роботи представлено у Додатку 8. Результат роботи відновлення в pgAdmin4 представлено у Додатку 9.

# 6 Аналіз результатів підвищення швидкодії запитів

За специфікою предметної області існують запити щодо визначення команд, що беруть участь у турнірі. Тому було створено індекс за полем “tournaments\_id” таблиці “teams”, що прискорило запит на отримання у 3 рази (див. Додаток 10)

Для виведення статистик для всіх команд або турнірів індекси не використовуються, оскільки необхідно опрацювати всі дані таблиць.

Таблиці “match\_schedule” та “match\_results” мають зв’язок 1-1 за PK часто зв’язуються за допомогою LEFT JOIN. Створено індекси, що прискорюють представлення “all matches\_for\_teams” за допомогою Merge Join. Отримане прискорення незначне (див. Додаток 11).

Вході розробки було вдосконалено запит шляхом відсікання більшості даних, що не повинні враховуватися у кінцевому запиті, на початкових етапах обробки, що прискорило запит у 5 разів (див. Додаток 12)

Для виведення розкладу матчів необхідно фільтрувати дані таблиці «match\_schedule» за полем «tournaments\_id», тому за цим полем створено hash-індекс. Результат підвишення швидкодії наведено у Додатку 13 у вигляді графіку залежності швидкості виконання операції до розміру таблиць.

# 7 Опис результатів аналізу предметної галузі

Реалізовано консольний інтерфейс в якому:

* Для переходу між меню необхідно вводити відповідну цифру з пунктів поточного меню (див. Додаток А1)
* Ввід даних користувача перевіряється на:
* відповідність типу введеного значення
* відповідність множині значень (якщо існують обмеження)
* Приклади обробки некоректних даних користувача наведено у Додатку А2.
* Якщо запит повертає колекцію елементів (наприклад, команда get\_all для сховищ сутностей), то застосунок питає чи необхідно їх відображати на екрані (див. Додаток А3)
* Якщо запит повертає статистику, то є можливість виведення даних на екран або збереження інформації у файл у вигляді діаграми (див. Додаток А4). Також у кожної команди статистики є свій перелік діаграм (див. Додаток А5)

CRUD-операції реалізовано за допомогою ORM SQLAlchemy. Приклад використання наведено у Додатку В1.

Генерація даних та отримання статистик і розкладу матчів відбувається шляхом виклику відповідних функцій або представлень у СУБД за допомогою використання адапетра для роботи з PostgreSQL — psycopg2.

|  |  |
| --- | --- |
| Назва функції/представлення | Призначення |
| generate\_tournament | Генерування запису турніру |
| generate\_playground\_in\_tournament(  tournamnets\_id) | Генерування майданчика для турніру |
| generate\_team\_in\_tournament(  tournamnets\_id) | Генерування команди для турніру |
| generate\_match\_schedule\_record\_in\_tournament(  tournamnets\_id) | Генерування запису заплановоного матчу для турніру |
| generate\_match\_results\_record\_in\_tournament(  tournamnets\_id) | Генерування запису зіграного матчу |
| tournaments\_statistics | Повертає статистику всіх турнірів |
| teams\_statistics\_in\_tournament(  tournamnets\_id) | Повертає статистику команди в турнірі |
| tournaments\_schedule(  tournamnets\_id) | Повертає розклад матчів для турніру |

Код яких наведено у Додатку B2.

Після генерації даних є можливість відміни операції. Для тестування виконання транзакцій було запущено 2 екземпляри програми, що перший – після генерації виконав операцію rollback; другий – commit (див. Додаток 14). Операція відміни транзакції (rollback) у Postgresql створює «мертві» кортежі (dead tuples), тому виникає необхідність періодично очищувати їх за допомогою команди VACUUM [11]. Налаштовано postgresql.conf для автоматичного виконання команди VACUUM (див. Додаток 15)

За для забезпечення коректності даних таблиці “match\_schedule”, було створено тригер “check\_valid\_data\_match\_schedule” перед вставленням та редагуванням записів, що у випадку коли команда грає сама собою або команда, матч, майданчик прив’язані до різних турнірів, то генерує помилку (див. Додаток 16).

Код тригерної функції наведено у Додатку B3.

Формування графічних діаграм статистик турнірів, турніру, команд в турнірі реалізовано за допомогою використання бібліотек numpy та matplotlib. Приклад використання наведено в Додатку В4. Результати формування наведено в Додатку Б.

**Висновки**

Під час виконання курсової роботи отримано досвід розробки інформаційної системи, що взаємодіє з базами даних; документування програмного застосунку; набуто навички налаштування реплікації, механізмів резервування/відновлення, прискорення запитів.

В результаті виконання курсової роботи:

* Розроблено консольний застосунок в якому:
  + виконуються CRUD операцій для турнірів та команд
  + генеруються дані турнірів, команд, матчів з можливістю відміни операції
  + формуються графічні діаграми статистик турнірів, турніру, команд в турнірі
* Налаштовано фізичну потокову реплікацію
* Налаштовано механізм резервування та відновлення

Розроблена система може бути розширена обробкою даних учасників і/або використовуватися як окремий продукт для обслуговування турнірів.

**Література**

1 DB-Engines Ranking [Електронний ресурс] – Режим доступу до ресурсу: https://db-engines.com/en/ranking.

2 Oracle Database [Електронний ресурс] – Режим доступу до ресурсу: <https://www.oracle.com/ru/database/>

3 SQL Server 2019 [Електронний ресурс] – Режим доступу до ресурсу: <https://www.microsoft.com/ru-ru/sql-server/sql-server-2019>

4 Царьов О. PostgreSQL vs MySQL [Електронний ресурс] / Олег Царьов – Режим доступу до ресурсу: https://habr.com/ru/company/mailru/blog/248845/.

5 PostgreSQL: Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/12/features.html.

6 Python: простое лучше, чем сложное [Електронний ресурс] – Режим доступу до ресурсу: http://suhorukov.com/news\_akademy/python-prostoe-luchshe-chem-slozhnoe.

7. Overview - NumPy v1.19 [Електронний ресурс] – Режим доступу до ресурсу: <https://numpy.org/doc/stable/>.

8. psycopg2: Project description [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/psycopg2/.

9. SQLAlchemy -Object Relational Mapper [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlalchemy.org/.

10. Matplotlib: Python plotting [Електронний ресурс] – Режим доступу до ресурсу: <https://matplotlib.org/>.

11. Hussain S. PostgreSQL VACUUM and ANALYZE Best Practice Tips [Електронний ресурс] / Sadequl Hussain – Режим доступу до ресурсу: https://www.2ndquadrant.com/en/blog/postgresql-vacuum-and-analyze-best-practice-tips/.

# ДОДАТКИ

ДОДАТОК 1 Таблиці морфологічного аналізу вибору РСУБД

Таблиця 1 – Морфологічний аналіз вибору РСУБД

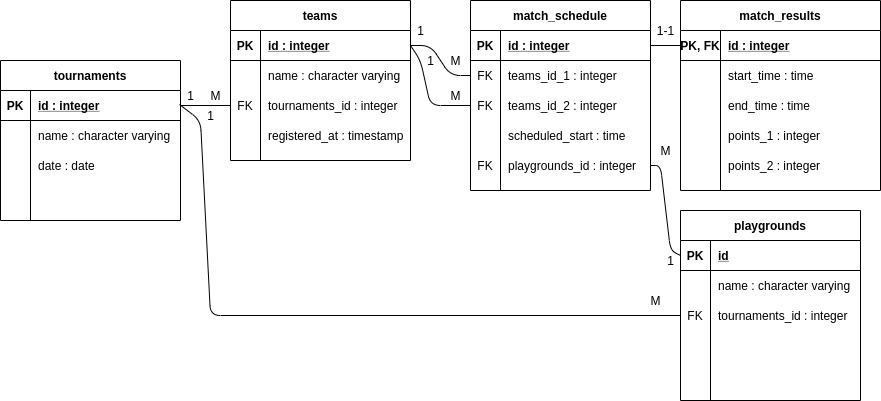
|  |  |  |  |
| --- | --- | --- | --- |
| Критерій | Вага критерію | MySQL | PostgreSQL |
| Підтримка стандартів SQL | 1 | 2 | 1 |
| Складність освоєння | 1 | 1 | 2 |
| Набутий досвід використання | 3 | 2 | 1 |
| Швидкодія | 2 | 2 | 1 |
| Типи даних | 1 | 2 | 1 |
| Складність реплікації | 2 | 2 | 1 |
| Якість документації | 2 | 2 | 1 |
| Сумарний пріоритет | - | 23 | 13 |

ДОДАТОК 2 Таблиці морфологічного аналізу вибору мови програмування

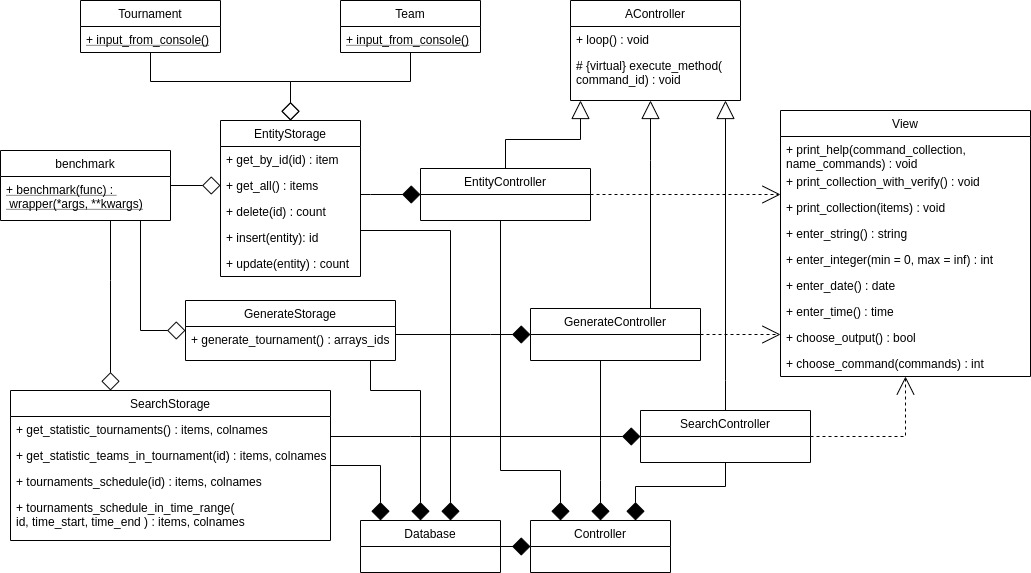
Таблиця 2 – Морфологічний аналіз вибору мови програмування

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерій | Вага критерію | C++ | C# | Python |
| Складність освоєння | 3 | 3 | 2 | 1 |
| Набутий досвід | 2 | 1 | 3 | 2 |
| Швидкодія роботи програми | 1 | 1 | 2 | 3 |
| Швидкодія розробки | 3 | 3 | 2 | 1 |
| Робота з діаграмами | 3 | 3 | 2 | 1 |
| Сумарний пріоритет | - | 30 | 26 | 16 |

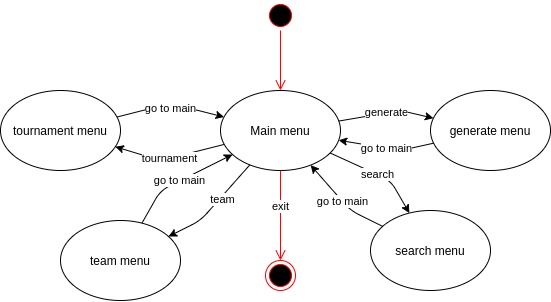
ДОДАТОК 3 Схема баз даних



ДОДАТОК 4 Діаграма класів

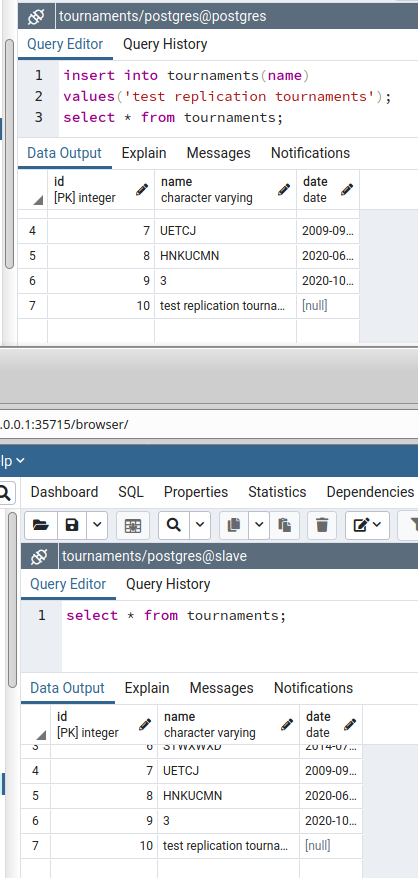
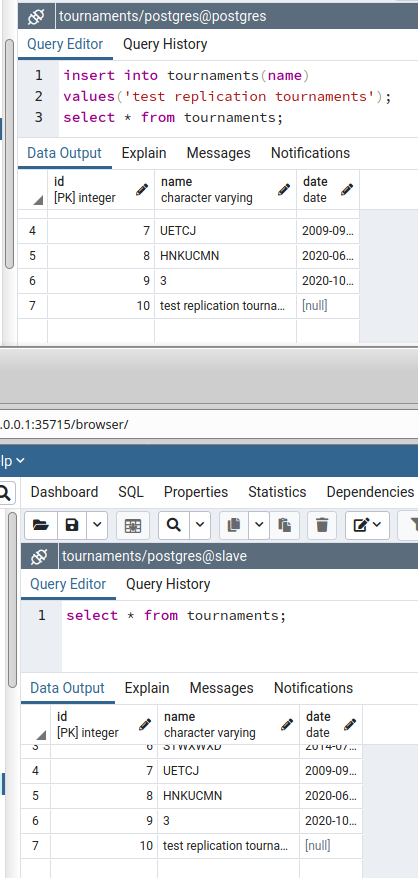


ДОДАТОК 5 Діаграма станів



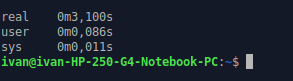
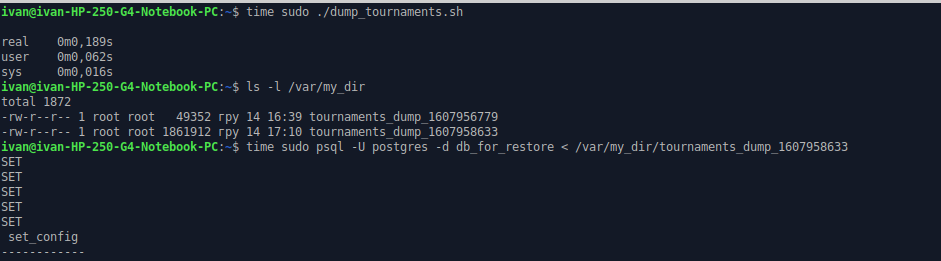
ДОДАТОК 6 Демонстрація наявності потокової реплікації на первинній БД

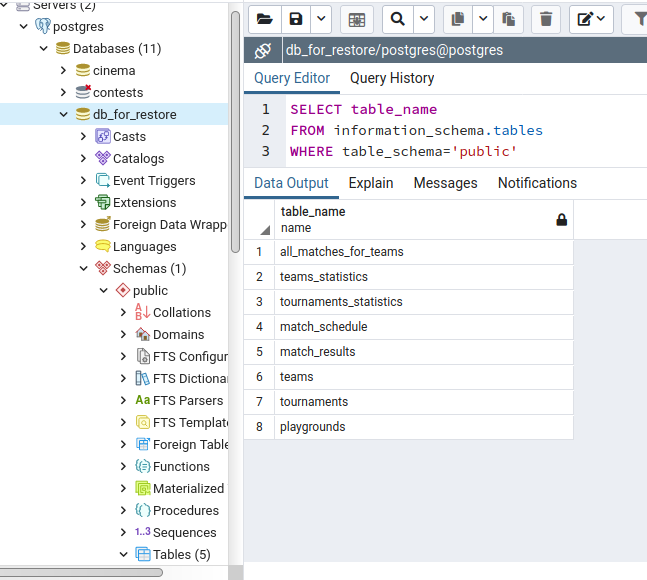
|  |  |
| --- | --- |
| При відключеному slave | При запущеному slave |
|  |  |
|  |  |

ДОДАТОК 7 Демонстрація коректної роботи реплікації

В первинній базі даних (ліворуч, @postgres) було створено новий запис. За допомогою запиту в базі даних реплікації (@slave) перевіряємо наявність відповідного запиту

ДОДАТОК 8 Демонстрація роботи резервування та відновлення



ДОДАТОК 9 Результат роботи відновлення в pgAdmin4

ДОДАТОК 10 Індексування поля “tournaments\_id” таблиці “teams”

|  |  |
| --- | --- |
| Без індексів: | З BTree-індексом: |
|  |  |

ДОДАТОК 11 Оптимізація з’єднання таблиць

|  |  |
| --- | --- |
| Hash Join — 375ms | Merge Join — 343 ms |
|  |  |

ДОДАТОК 12 Оптимізація запиту

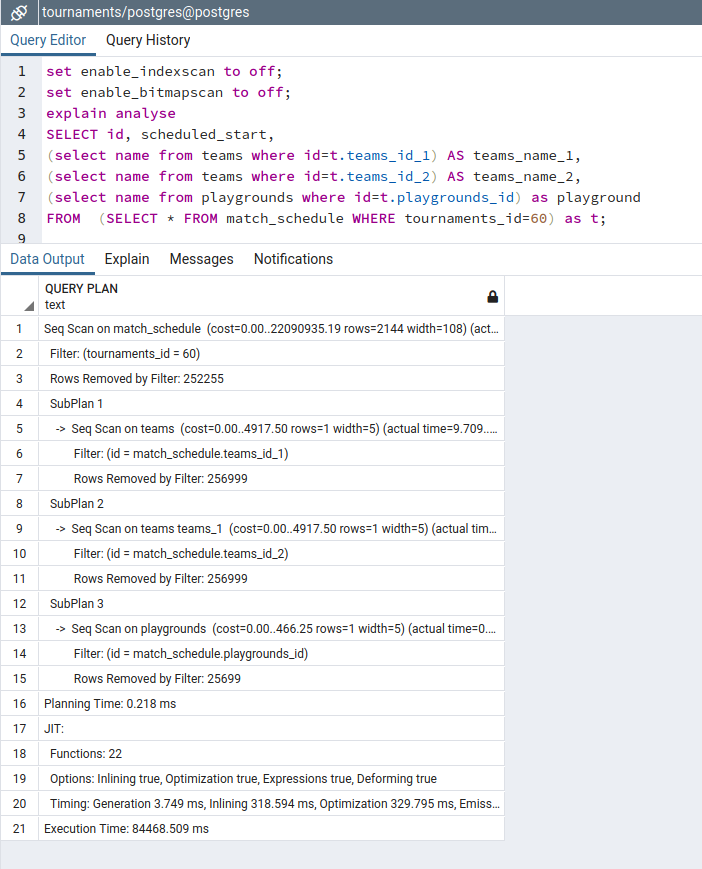
|  |  |
| --- | --- |
| Не оптимізований | Оптимізований |
|  |  |

ДОДАТОК 13 Швидкодія виконання операції

отримання розкладу матчів турніру

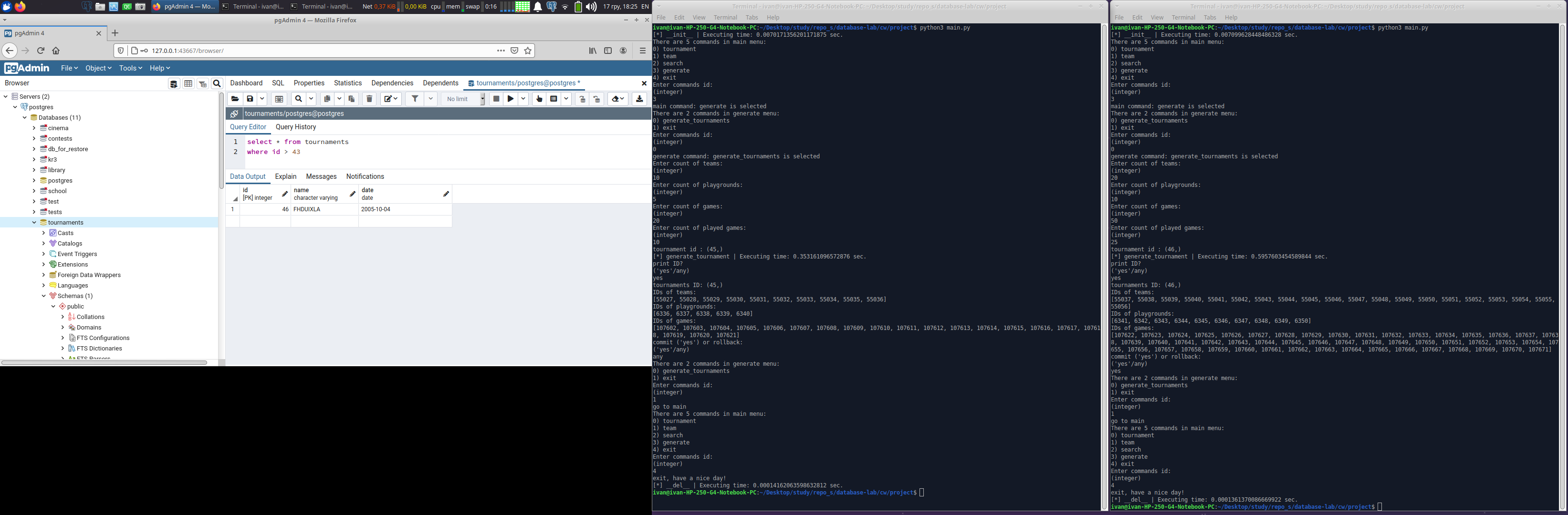
ДОДАТОК 13.1 Графік залежності швидкості виконання операції tournament\_schedule до розміру таблиць

ДОДАТОК 13.2 Послідовне сканування при розміру таблиць 250000 рядків

в

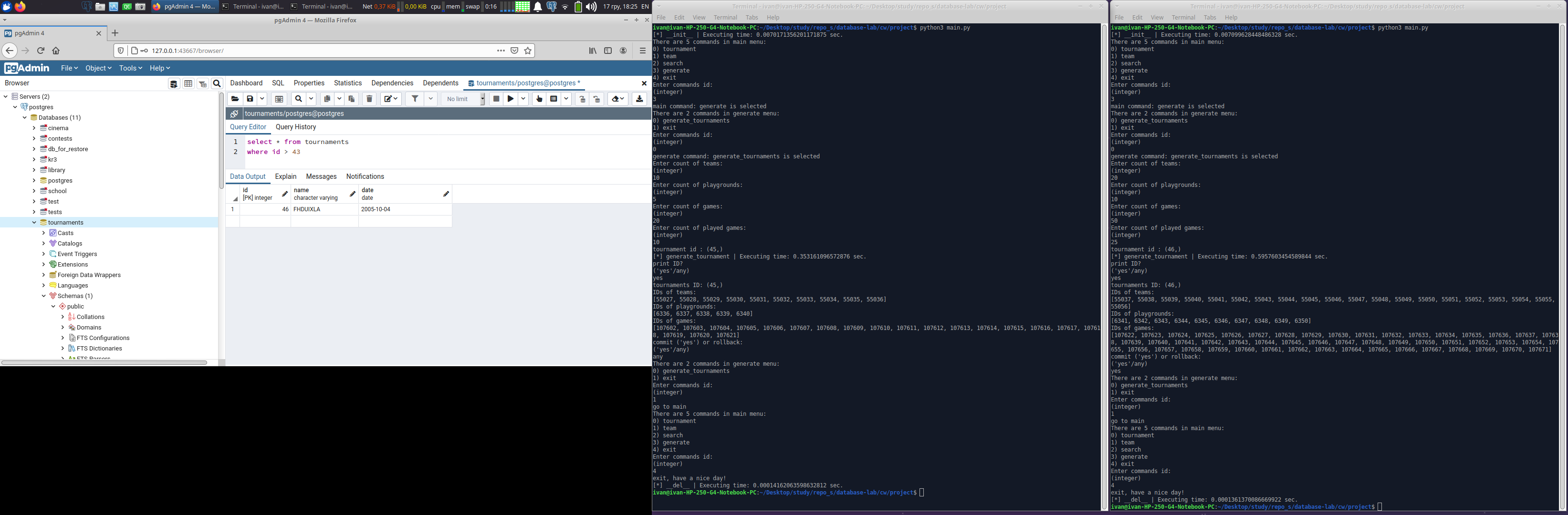
ДОДАТОК 14 Демонстрація роботи commit та rollback

ДОДАТОК 14.1 rollback генерації турніра з id 45



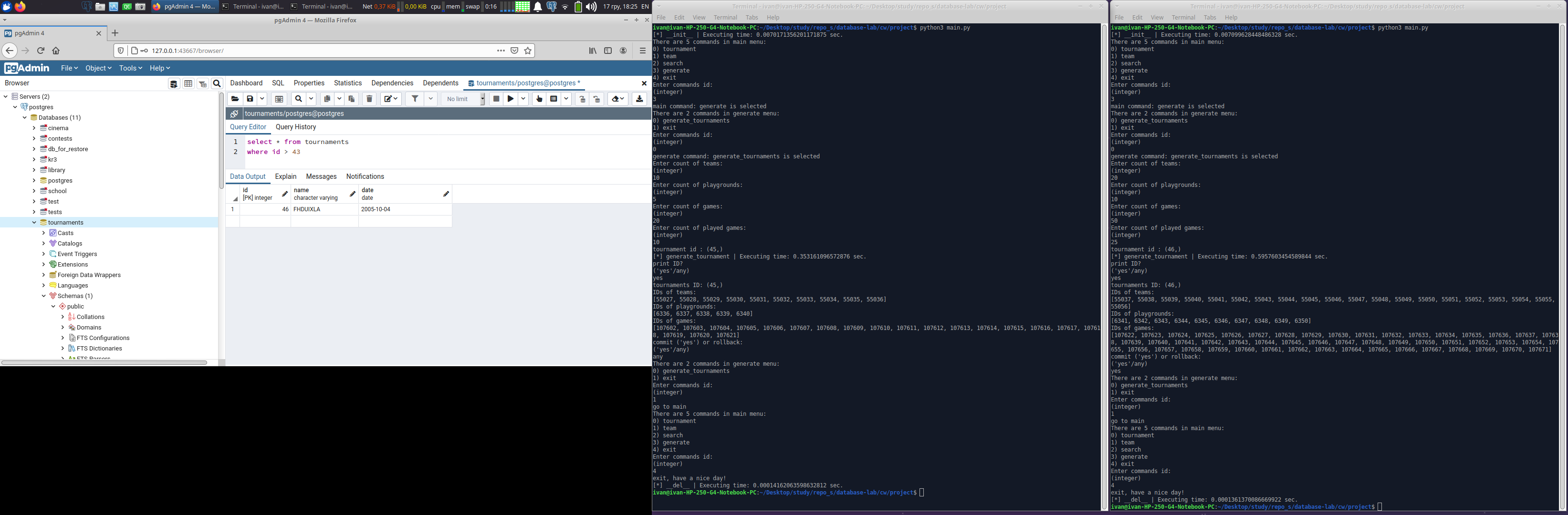
ДОДАТОК 14 Демонстрація роботи commit та rollback

ДОДАТОК 14.2 commit генерації турніра з id 46



ДОДАТОК 14 Демонстрація роботи commit та rollback

ДОДАТОК 14.3 Підтвердження коректної роботи commit та rollback



ДОДАТОК 15 Налаштування файлу postgresql.conf

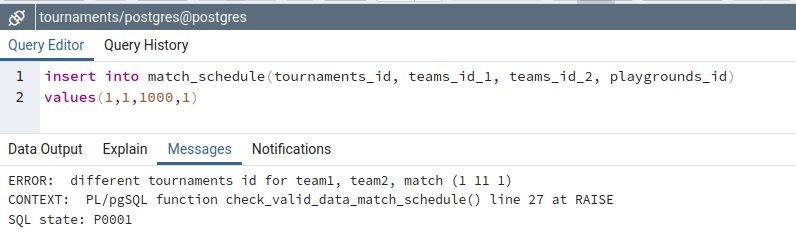
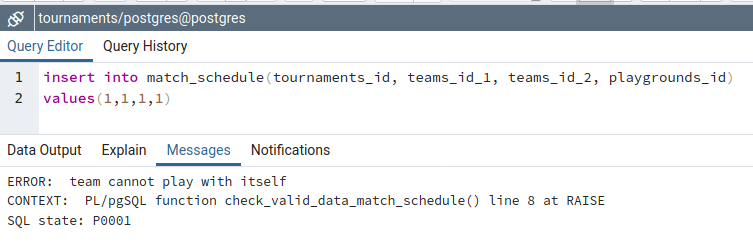
для автоматичного VACUUM





ДОДАТОК 16 Генерація помилки при вставці

некоректних даних у таблицю «match\_schedule»

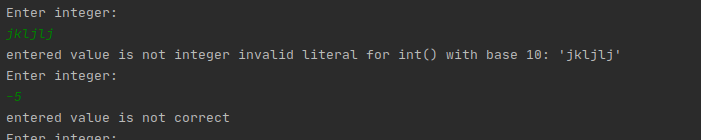
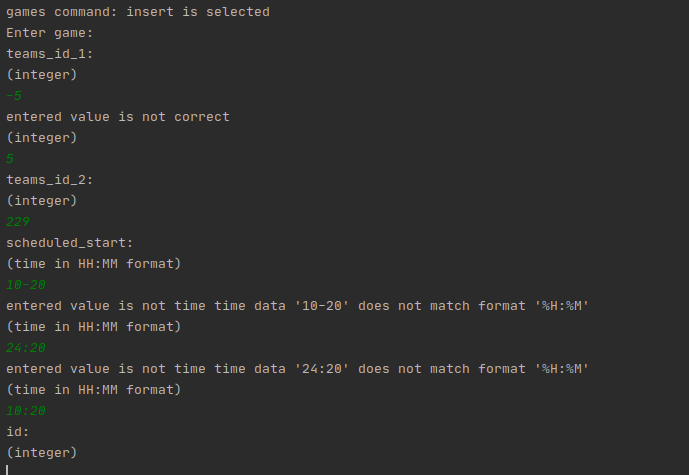


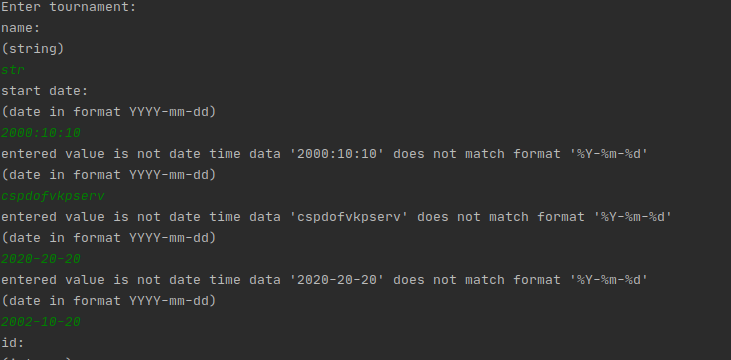
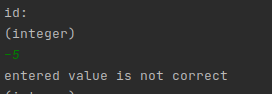
# ДОДАТКИ А. Консольний інтерфейс

ДОДАТОК А1 Інтерфейси меню

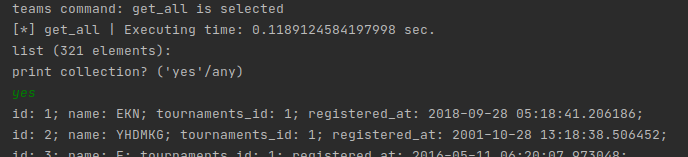
|  |  |
| --- | --- |
|  |  |

ДОДАТОК А2 Приклади обробки некоректних даних користувача

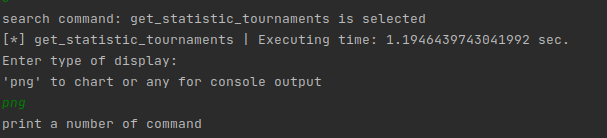
* відповідність типу введеного значення
  + Цілочисельний тип:
  + Час:
  + Дата
* відповідність множині значень (якщо існують обмеження)



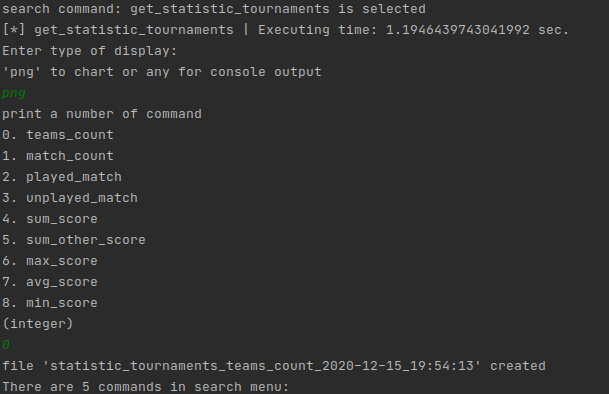
ДОДАТОК А3 Підтвердження виводу сутностей на екран

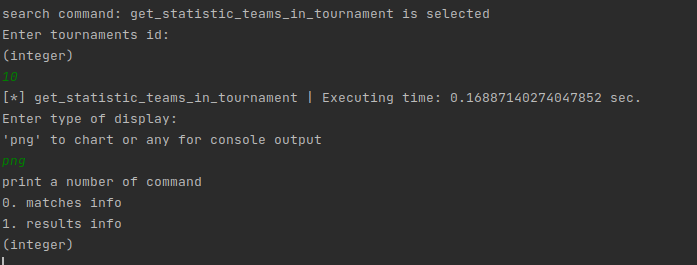


ДОДАТОК А4 Обрання типу виведення статистик

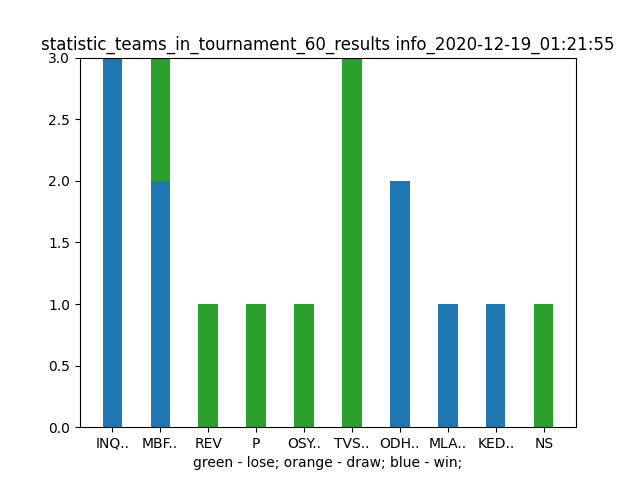
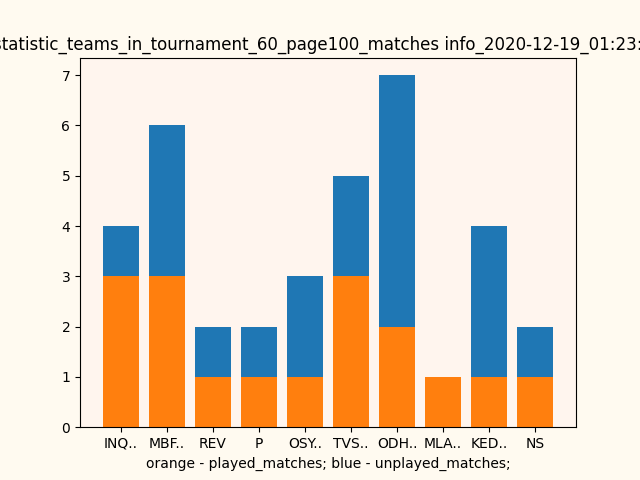


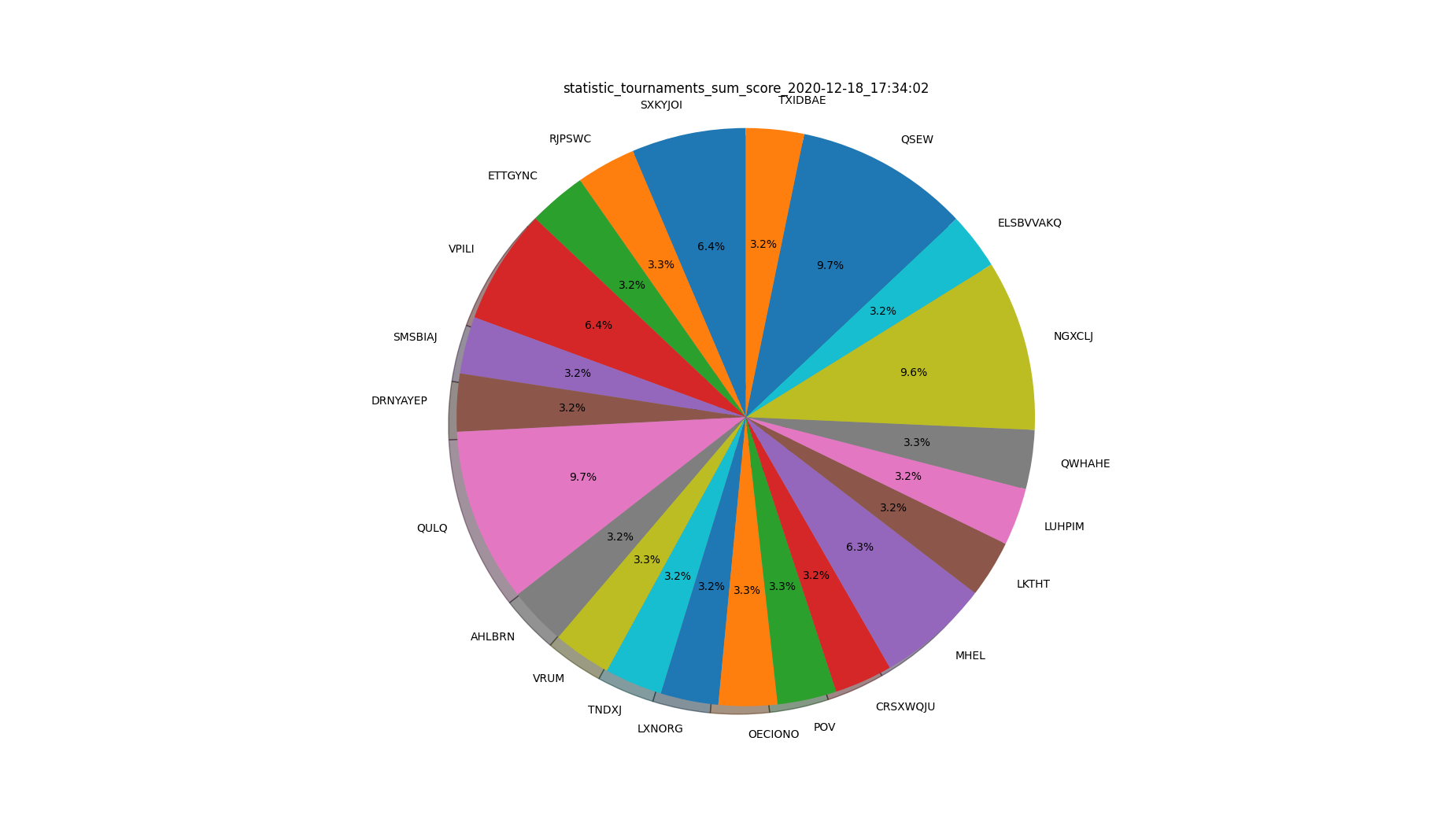
ДОДАТОК А5 Меню обрання типу діаграм





# ДОДАТКИ Б. Створені діаграми





# ДОДАТКИ В. Фрагменти коду

ДОДАТОК В1 Приклад використання SQLAlchemy

|  |
| --- |
| models.entities.tournament.py |
| # //-----------------------  class Tournament(Base):  \_\_tablename\_\_ = 'tournaments'  id = Column(Integer, Sequence('tournaments\_id\_seq'), primary\_key=True)  name = Column(String)  date = Column(Date)  # //----------------------- |
| models.entities.team.py |
| # //-----------------------  class Team(Base):  \_\_tablename\_\_ = 'teams'  id = Column(Integer, Sequence('teams\_id\_seq'), primary\_key=True)  name = Column(String)  tournaments\_id = Column(Integer, ForeignKey('tournaments.id', ondelete='CASCADE'))  registered\_at = Column(DateTime, default=datetime.datetime.utcnow)  # //----------------------- |
| models.storages.entity.py |
| from benchmark import benchmark  from db import session  def get\_entity\_mapped\_keys(item):  mapped\_values = {}  for entity in item.\_\_dict\_\_.items():  key = entity[0]  value = entity[1]  if key != '\_sa\_instance\_state' and key != 'id':  mapped\_values[key] = value  return mapped\_values  class StorageEntity:  def \_\_init\_\_(self, instance):  self.instance = instance  @benchmark  def get\_by\_id(self, id: int):  return session. \  query(self.instance). \  get(id)  @benchmark  def get\_all(self):  return session. \  query(self.instance). \  all()  @benchmark  def delete(self, id: int):  result = session. \  query(self.instance). \  filter(self.instance.id == id). \  delete()  session.commit()  return result  @benchmark  def insert(self, entity):  entity.id = None  session.add(entity)  session.commit()  return entity.id  @benchmark  def update(self, entity):  result = session. \  query(self.instance). \  filter(self.instance.id == entity.id). \  update(get\_entity\_mapped\_keys(entity))  session.commit()  return result |

ДОДАТОК В2 Код представлень та функцій серверу postgres

|  |
| --- |
| функції генерування даних |
| CREATE FUNCTION public.generate\_match\_schedule\_record\_in\_tournament(value\_tournaments\_id integer) RETURNS integer  LANGUAGE sql  AS $$  WITH  source\_teams AS  (SELECT id FROM public.teams  WHERE tournaments\_id = value\_tournaments\_id  ORDER BY random() LIMIT 2),  source\_playground\_id AS  (SELECT id FROM public.playgrounds  WHERE tournaments\_id = value\_tournaments\_id  ORDER BY random() LIMIT 1),  source\_to\_insert\_game AS  (SELECT  (SELECT id FROM source\_teams  LIMIT 1 OFFSET 0) AS teams\_id\_1,  (SELECT id FROM source\_teams  LIMIT 1 OFFSET 1) AS teams\_id\_2,  (time '06:00:00' + random() \* (time '16:00:00')) AS scheduled\_start,  (SELECT id from source\_playground\_id) as playgrounds\_id,  value\_tournaments\_id)  INSERT INTO public.match\_schedule(teams\_id\_1, teams\_id\_2, scheduled\_start, tournaments\_id, playgrounds\_id)  (SELECT teams\_id\_1, teams\_id\_2, scheduled\_start, value\_tournaments\_id, playgrounds\_id FROM source\_to\_insert\_game)  RETURNING id;  $$; |
| CREATE FUNCTION public.generate\_name(lenght integer DEFAULT 5) RETURNS name  LANGUAGE plpgsql  AS $$  DECLARE  answer varchar(50) default '';  rec\_char record;  cur\_char cursor(p\_lenght integer)  for select chr(trunc(65 + random() \* 25)::int) as chr  from generate\_series(1, p\_lenght);  BEGIN  open cur\_char(lenght);    loop    fetch cur\_char into rec\_char;    exit when not found;    answer := answer || rec\_char.chr;    end loop;    close cur\_char;    return answer;  END;  $$; |
| CREATE FUNCTION public.generate\_playground\_in\_tournament(id\_tournament integer) RETURNS integer  LANGUAGE sql  AS $$  INSERT INTO public.playgrounds(name, tournaments\_id)  VALUES(  generate\_name((random()\*10000)::int % 9 + 1),  id\_tournament)  RETURNING id  $$; |
| CREATE FUNCTION public.generate\_team\_in\_tournament(id\_tournament integer) RETURNS integer  LANGUAGE sql  AS $$  INSERT INTO public.teams(name, registered\_at, tournaments\_id)  VALUES(  generate\_name((random()\*10000)::int % 9 + 1),  (timestamp '2000-01-01' + random() \* (current\_timestamp - timestamp '2000-01-01')),  id\_tournament)  RETURNING id  $$; |
| CREATE FUNCTION public.generate\_tournament() RETURNS integer  LANGUAGE sql  AS $$  INSERT INTO public.tournaments(name, date)  VALUES(  generate\_name((random()\*10000)::int % 9 + 1),  date(timestamp  '2000-01-01' + random() \* (timestamp '2031-01-01' - timestamp '2000-01-01'))  )  RETURNING id;  $$; |

|  |
| --- |
| функції та представлення запитів статистик |
| CREATE FUNCTION public.teams\_statistics\_in\_tournament(value\_tournament\_id integer) RETURNS TABLE(teams\_id integer, match\_count bigint, played\_match bigint, unplayed\_match bigint, win\_count bigint, draw\_count bigint, lose\_count bigint, sum\_score bigint, max\_score integer, avg\_score numeric, min\_score integer, match\_count\_freq numeric, sum\_score\_freq numeric)  LANGUAGE sql  AS $$  with t2 as  (select \* from tournaments\_statistics  where tournaments\_id=value\_tournament\_id)  select  t1.\*  ,t1.match\_count / (SELECT SUM(t2.match\_count) FROM t2) / 2 as match\_count\_freq  ,t1.sum\_score / (SELECT SUM(t2.sum\_score) FROM t2) / 2 as sum\_score\_freq  from teams\_statistics t1  where teams\_id in (select id from teams where tournaments\_id=value\_tournament\_id)  $$; |
| CREATE FUNCTION public.tournaments\_schedule(value\_tournaments\_id integer) RETURNS TABLE(id integer, teams\_name\_1 character varying, teams\_name\_2 character varying, scheduled\_start time without time zone, playground character varying)  LANGUAGE sql  AS $$  SELECT  id,  (select name from teams where id=match\_schedule.teams\_id\_1) AS teams\_name\_1,  (select name from teams where id=match\_schedule.teams\_id\_2) AS teams\_name\_2,  scheduled\_start,  (select name from playgrounds where id=match\_schedule.playgrounds\_id) as playground  FROM match\_schedule  WHERE  tournaments\_id=value\_tournaments\_id;  $$; |
| CREATE VIEW public.all\_matches\_for\_teams AS  WITH match\_where\_team1 AS (  SELECT s.id AS match\_id,  s.tournaments\_id,  s.teams\_id\_1 AS teams\_id,  s.teams\_id\_2 AS other\_teams\_id,  r.points\_1 AS score,  r.points\_2 AS other\_score  FROM (public.match\_schedule s  LEFT JOIN public.match\_results r ON ((s.id = r.match\_schedule\_id)))  ), match\_where\_team2 AS (  SELECT s.id AS match\_id,  s.tournaments\_id,  s.teams\_id\_2 AS teams\_id,  s.teams\_id\_1 AS other\_teams\_id,  r.points\_2 AS score,  r.points\_1 AS other\_score  FROM (public.match\_schedule s  LEFT JOIN public.match\_results r ON ((s.id = r.match\_schedule\_id)))  )  SELECT match\_where\_team1.match\_id,  match\_where\_team1.tournaments\_id,  match\_where\_team1.teams\_id,  match\_where\_team1.other\_teams\_id,  match\_where\_team1.score,  match\_where\_team1.other\_score  FROM match\_where\_team1  UNION  SELECT match\_where\_team2.match\_id,  match\_where\_team2.tournaments\_id,  match\_where\_team2.teams\_id,  match\_where\_team2.other\_teams\_id,  match\_where\_team2.score,  match\_where\_team2.other\_score  FROM match\_where\_team2; |
| CREATE VIEW public.teams\_statistics AS  SELECT all\_matches\_for\_teams.teams\_id,  count(\*) AS match\_count,  count(  CASE  WHEN (all\_matches\_for\_teams.score IS NULL) THEN NULL::integer  ELSE 1  END) AS played\_match,  count(  CASE  WHEN (all\_matches\_for\_teams.score IS NULL) THEN 1  ELSE NULL::integer  END) AS unplayed\_match,  count(  CASE  WHEN (all\_matches\_for\_teams.score > all\_matches\_for\_teams.other\_score) THEN 1  ELSE NULL::integer  END) AS win\_count,  count(  CASE  WHEN (all\_matches\_for\_teams.score = all\_matches\_for\_teams.other\_score) THEN 1  ELSE NULL::integer  END) AS draw\_count,  count(  CASE  WHEN (all\_matches\_for\_teams.score < all\_matches\_for\_teams.other\_score) THEN 1  ELSE NULL::integer  END) AS lose\_count,  sum(all\_matches\_for\_teams.score) AS sum\_score,  max(all\_matches\_for\_teams.score) AS max\_score,  avg(all\_matches\_for\_teams.score) AS avg\_score,  min(all\_matches\_for\_teams.score) AS min\_score  FROM public.all\_matches\_for\_teams  GROUP BY all\_matches\_for\_teams.teams\_id; |
| CREATE VIEW public.tournaments\_statistics AS  SELECT all\_matches\_for\_teams.tournaments\_id,  count(all\_matches\_for\_teams.teams\_id) AS teams\_count,  (count(\*) / 2) AS match\_count,  (count(  CASE  WHEN (all\_matches\_for\_teams.score IS NULL) THEN NULL::integer  ELSE 1  END) / 2) AS played\_match,  (count(  CASE  WHEN (all\_matches\_for\_teams.score IS NULL) THEN 1  ELSE NULL::integer  END) / 2) AS unplayed\_match,  (sum(all\_matches\_for\_teams.score) / 2) AS sum\_score,  (sum(all\_matches\_for\_teams.other\_score) / 2) AS sum\_other\_score,  max(all\_matches\_for\_teams.score) AS max\_score,  avg(all\_matches\_for\_teams.score) AS avg\_score,  min(all\_matches\_for\_teams.score) AS min\_score  FROM public.all\_matches\_for\_teams  GROUP BY all\_matches\_for\_teams.tournaments\_id; |

ДОДАТОК В3 Код тригерної функції

|  |
| --- |
| CREATE FUNCTION public.check\_valid\_data\_match\_schedule() RETURNS trigger  LANGUAGE plpgsql  AS $$  DECLARE  t1 int;  t2 int;  t3 int;  BEGIN  IF NEW.teams\_id\_1=NEW.teams\_id\_2 THEN  RAISE EXCEPTION 'team cannot play with itself';  END IF;    SELECT tournaments\_id INTO STRICT t1 from teams  WHERE teams.id = NEW.teams\_id\_1;  SELECT tournaments\_id INTO STRICT t2 from teams  WHERE teams.id = NEW.teams\_id\_2;    IF NEW.playgrounds\_id IS NOT NULL THEN  SELECT tournaments\_id INTO STRICT t3 from playgrounds  WHERE playgrounds.id = NEW.playgrounds\_id;  IF t1!=t3 THEN  RAISE EXCEPTION 'different tournaments id for playground';  END IF;  END IF;    IF t1=t2 AND t1=NEW.tournaments\_id THEN  RETURN NEW;  ELSE  RAISE EXCEPTION 'different tournaments id for team1, team2, match (% % %)', t1, t2, NEW.tournaments\_id;  END IF;    END;  $$; |
| CREATE TRIGGER check\_valid\_data\_match\_schedule BEFORE INSERT OR UPDATE ON public.match\_schedule FOR EACH ROW EXECUTE FUNCTION public.check\_valid\_data\_match\_schedule(); |

ДОДАТОК В4 Приклад використання бібліотек numpy та matplotlib

|  |
| --- |
| controllers.search.py |
| import view as View  from controllers.abstract import AController  from models.storages.search import SearchStorage  import matplotlib.pyplot as plt  from datetime import datetime  import numpy as np  class SearchController(AController):  def \_\_init\_\_(self, search\_storage: SearchStorage):  super().\_\_init\_\_('search', 'go to main')  self.commands = [  'get\_statistic\_tournaments',  'get\_statistic\_teams\_in\_tournament',  'tournaments\_schedule',  'tournaments\_schedule\_in\_time\_range'  ] + self.commands  self.methods = [  self.\_\_get\_statistic\_tournaments,  self.\_\_get\_statistic\_teams\_in\_tournament,  self.\_\_tournaments\_schedule,  self.\_\_tournaments\_schedule\_in\_time\_range,  ]  self.search\_storage = search\_storage  def execute\_method(self, command\_id: int):  self.methods[command\_id]()  def \_\_get\_statistic\_tournaments(self):  items, colnames = self.search\_storage.get\_statistic\_tournaments()  if View.choose\_output():  index\_of\_name = colnames.index('name')  count\_of\_info\_field = colnames.index('teams\_count')  colnames = colnames[count\_of\_info\_field:]  field = View.choose\_command(colnames)  sizes = [row[field + count\_of\_info\_field] for row in items]  labels = [row[index\_of\_name] for row in items]  index\_empty\_fields = []  for i in range(0, len(sizes)):  if sizes[i] is None:  index\_empty\_fields.append(i)  count = 0  for i in index\_empty\_fields:  sizes.pop(i - count)  labels.pop(i - count)  count += 1  explode = [0.1 for row in labels]  fig, ax = plt.subplots()  fig.set\_size\_inches(18.5, 10.5, forward=True)  ax.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',  shadow=True, startangle=90)  ax.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.  temp\_str = f'statistic\_tournaments\_{colnames[field]}\_{datetime.now().strftime("%Y-%m-%d\_%H:%M:%S")}'  plt.title(temp\_str)  plt.savefig(temp\_str)  print(f'file \'{temp\_str}\' created')  else:  print(colnames)  View.print\_collection\_with\_verify(items)  def \_\_get\_statistic\_teams\_in\_tournament(self):  print('Enter tournaments id:')  tournaments\_id = View.enter\_integer()  items, colnames = self.search\_storage.get\_statistic\_teams\_in\_tournament(tournaments\_id)  if View.choose\_output():  index\_of\_name = colnames.index('name')  labels = [row[index\_of\_name] for row in items]  for i in range(0, len(labels)):  if len(labels[i]) > 3:  labels[i] = labels[i][:3] + '..'  commands = ['matches info', 'results info']  commands\_id = View.choose\_command(commands)  if commands\_id == 0:  played\_matches = [row[colnames.index('played\_match')] for row in items]  matches\_count = [row[colnames.index('match\_count')] for row in items]  for i in range(0, len(matches\_count)): # len(matches\_count) == len(played\_matches)  if played\_matches[i] is None:  played\_matches[i] = 0  if matches\_count[i] is None:  matches\_count[i] = 0  fig, ax = plt.subplots()  fig.set\_size\_inches(18.5, 10.5, forward=True)  ax.bar(labels, matches\_count)  ax.bar(labels, played\_matches)  ax.set\_facecolor('seashell')  fig.set\_facecolor('floralwhite')  plt.xlabel('orange - played\_matches; blue - unplayed\_matches;')  temp\_str = f'statistic\_teams\_in\_tournament\_{tournaments\_id}\_{commands[commands\_id]}\_{datetime.now().strftime("%Y-%m-%d\_%H:%M:%S")}'  plt.title(temp\_str)  plt.savefig(temp\_str)  if commands\_id == 1:  wins = [row[colnames.index('win\_count')] for row in items]  self.\_\_\_\_null\_to\_0(wins)  draws = [row[colnames.index('draw\_count')] for row in items]  self.\_\_\_\_null\_to\_0(draws)  loses = [row[colnames.index('lose\_count')] for row in items]  self.\_\_\_\_null\_to\_0(loses)  fig, ax = plt.subplots()  ax.bar(labels, wins, width=0.4)  ax.bar(labels, draws, width=0.4, bottom=wins)  ax.bar(labels, loses, width=0.4, bottom=np.add(wins, draws))  plt.xlabel('green - lose; orange - draw; blue - win;')  temp\_str = f'statistic\_teams\_in\_tournament\_{tournaments\_id}\_{commands[commands\_id]}\_{datetime.now().strftime("%Y-%m-%d\_%H:%M:%S")}'  plt.title(temp\_str)  plt.savefig(temp\_str)  print(f'file \'{temp\_str}\' created')  else:  print(colnames)  View.print\_collection\_with\_verify(items) |