

**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
1.1 Аналіз СУБД.....	5
1.2 Аналіз бібліотек та фреймворків.....	5
2 Структура бази даних.....	7
3 Опис програмного забезпечення	8
3.1 Загальна структура програмного забезпечення.....	8
3.2 Опис модулів програмного забезпечення.....	8
3.3 Опис основних алгоритмів роботи.....	9
4 Аналіз функціонування засобів реплікації	10
5 Аналіз функціонування засобів резервування/відновлення	10
6 Аналіз результатів підвищення швидкодії запитів	11
7 Опис результатів аналізу предметної галузі	12
ДОДАТКИ	16
ДОДАТКИ А. Консольний інтерфейс.....	24
ДОДАТКИ Б. Створені діаграми.....	28
ДОДАТКИ В. Фрагменти коду.....	31

Вступ

Починаючи з середини XX 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 взаємодіяє 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: C++, C#, 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 містить 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.
- playgrounds — 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 playgrounds)
- 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 Tournament та Team в таблицях tournaments 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.
- Вид представлений модулем View, що надає можливості вводу/виводу даних з консолі
- Контролери:
 - ActionController — 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 benchmark, 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.

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”:

<pre>dump_tournaments.sh timestamp=\$(date +%s) pg_dump -U postgres tournaments > /var/my_dir/tournaments_dump_\$timestamp</pre>

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 відновляться 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 РК часто зв’язуються за допомогою LEFT JOIN. Спробуємо створити індекси, що прискорювали б вид “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)

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 кожній команді 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 видів у СУБД за допомогою використання адапетра для роботи з PostgreSQL — `psycopg2`.

Назва функції/виду	Призначення
<code>generate_tournament</code>	Генерування запису турніру
<code>generate_playground_in_tournament(tournamnets_id)</code>	Генерування майданчика для турніру
<code>generate_team_in_tournament(tournamnets_id)</code>	Генерування команди для турніру
<code>generate_match_schedule_record_in_tournament(tournamnets_id)</code>	Генерування запису запланованого матчу для турніру
<code>generate_match_results_record_in_tournament(tournamnets_id)</code>	Генерування запису зіграного матчу
<code>tournaments_statistics</code>	Повертає статистику всіх турнірів
<code>teams_statistics_in_tournament(tournamnets_id)</code>	Повертає статистику команди в турнірі

tournaments_schedule(tournamnets_id)	Повертає розклад матчів для турніру
--	-------------------------------------

Код яких наведено у Додатку В2.

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 13).

Код тригерної функції наведено у Додатку В3.

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
 - 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. psycorg2: Project description [Електронний ресурс] – Режим доступу до ресурсу: <https://pypi.org/project/psycorg2/>.
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/>.

ДОДАТКИ

ДОДАТОК 1 Таблиці морфологічного аналізу вибору РСУБД

Таблиця 1 – Морфологічний аналіз вибору РСУБД

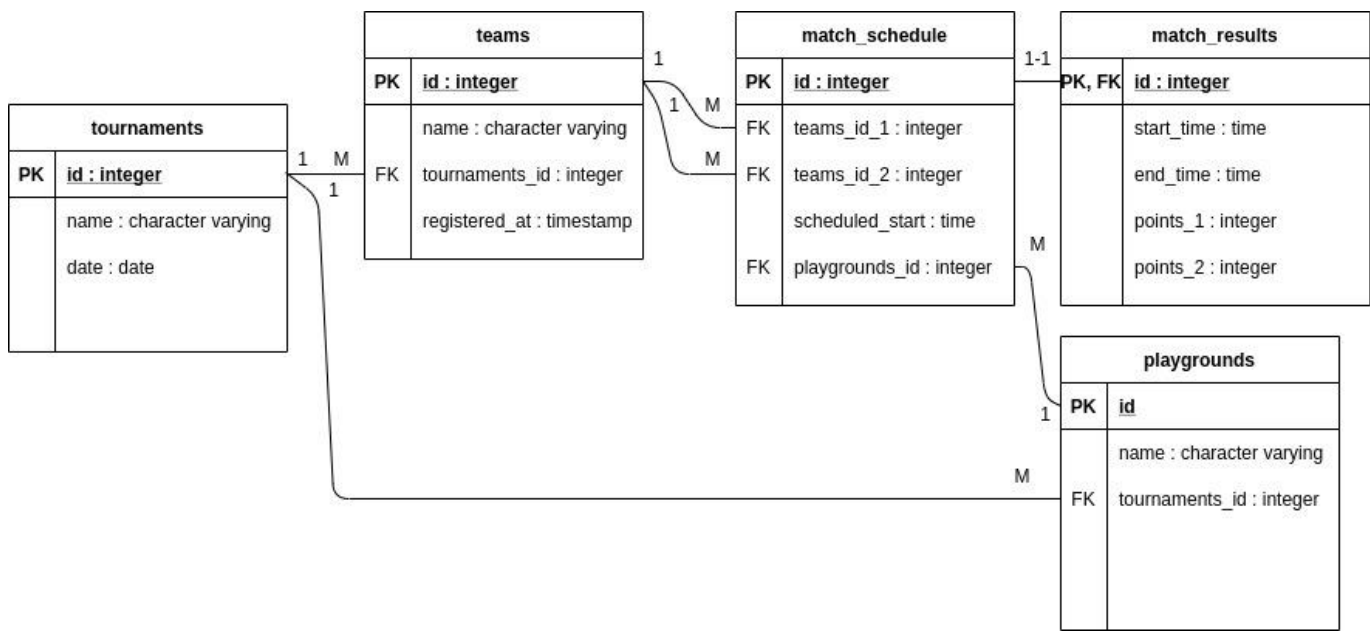
Критерій	Вага критерію	MySQL	PostgreSQL
Підтримка стандартів SQL	1	2	1
Складність освоєння	1	1	2
Набутий досвід використання	3	2	1
Швидкодія	2	2	1
Типи даних	1	2	1
Складність реплікації	2	2	1
Якість документації	2	2	1
Сумарний пріоритет	-	23	13

ДОДАТОК 2 Таблиці морфологічного аналізу вибору мови програмування

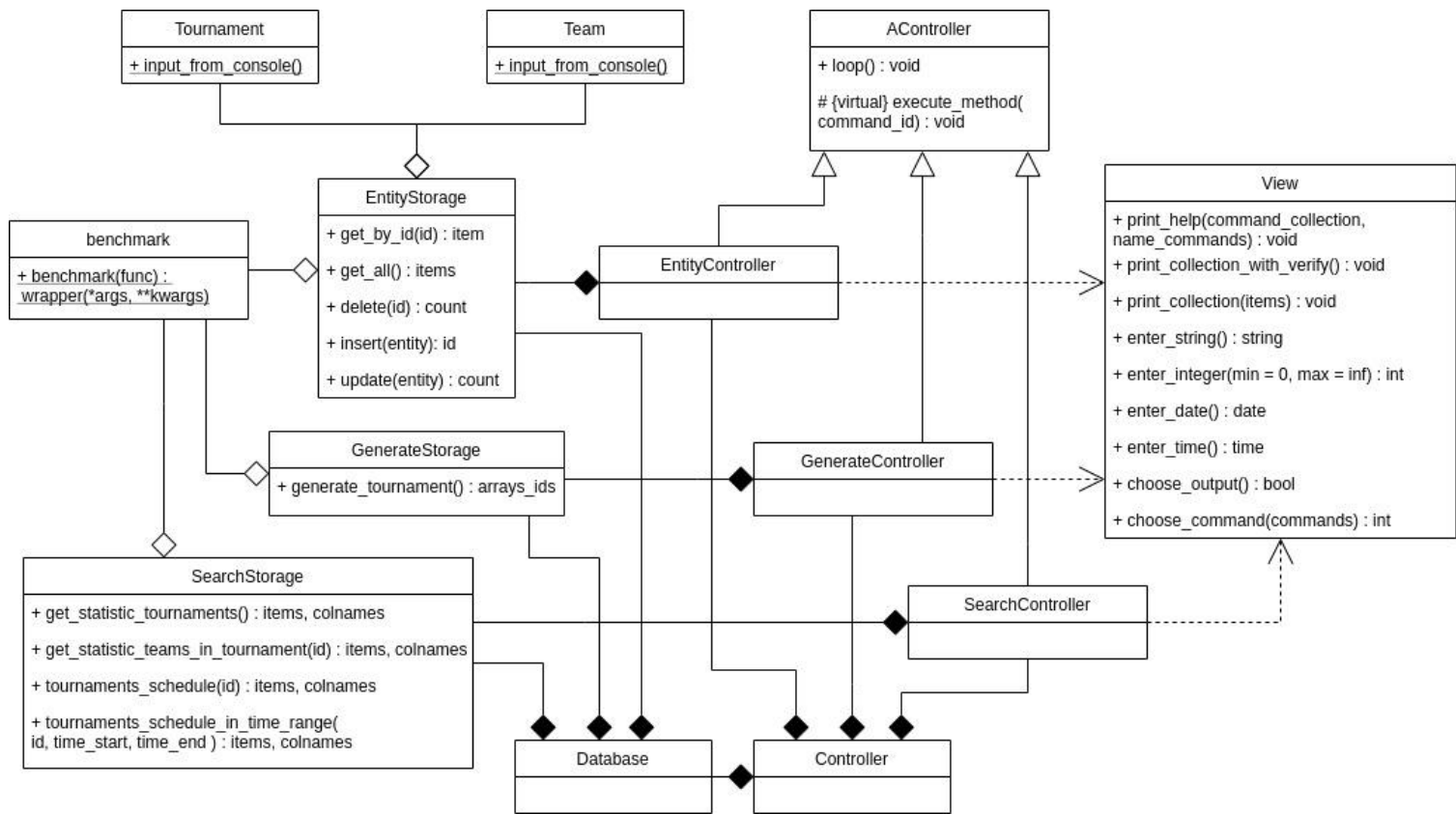
Таблиця 2 – Морфологічний аналіз вибору мови програмування

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

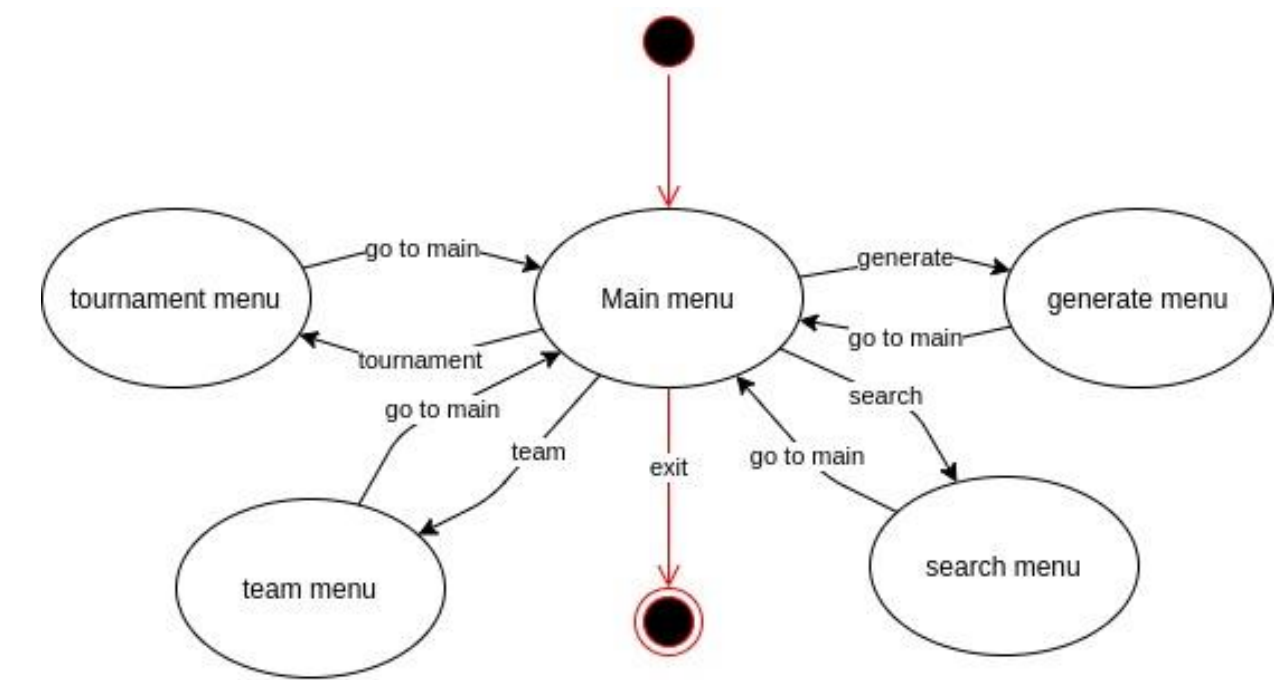
ДОДАТОК 3 Схеми баз даних



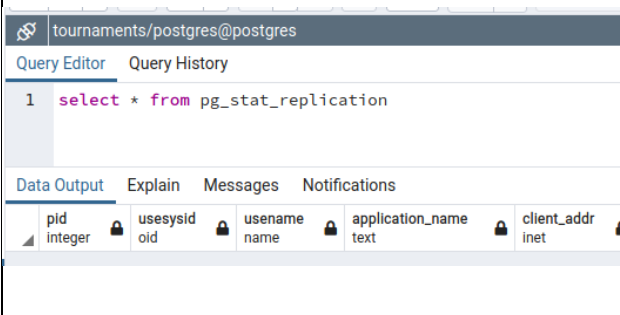
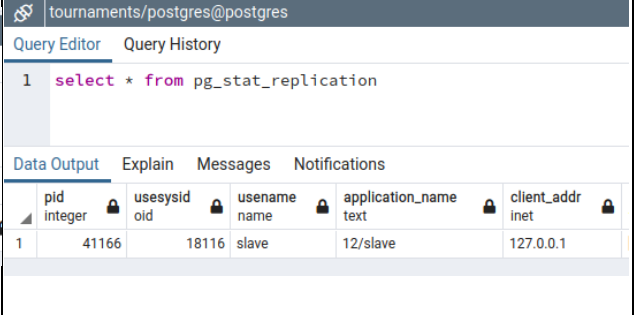
ДОДАТОК 4 Діаграма класів




ДОДАТОК 5 Діаграма станів



ДОДАТОК 6 Демонстрація наявності потокової реплікації на первинній БД

При відключеному slave	При запущеному slave
<pre>~\$ sudo systemctl stop postgresql@12-slave</pre>	<pre>~\$ sudo systemctl start postgresql@12-slave</pre>
	

ДОДАТОК 7 Демонстрація коректної роботи реплікації

 tournaments/postgres@postgres

Query Editor

Query History




```
1 insert into tournaments(name)
2 values('test replication tournaments');
3 select * from tournaments;
```


Data Output

Explain

Messages

Notifications

	 id [PK] integer	 name character varying	 date date	
4		7 UETCJ	2009-09...	
5		8 HNKUCMN	2020-06...	
6		9 3	2020-10...	
7		10 test replication tourna...	[null]	



tournaments/postgres@slave

Query Editor

Query History

1

select * from tournaments;

Data Output

Explain

Messages

Notifications

	id [PK] integer	name character varying	date date	
5		STRAW	2014-07...	
4		UETCJ	2009-09...	
5		HNKUCMN	2020-06...	
6		9 3	2020-10...	
7		10 test replication tourna...	[null]	

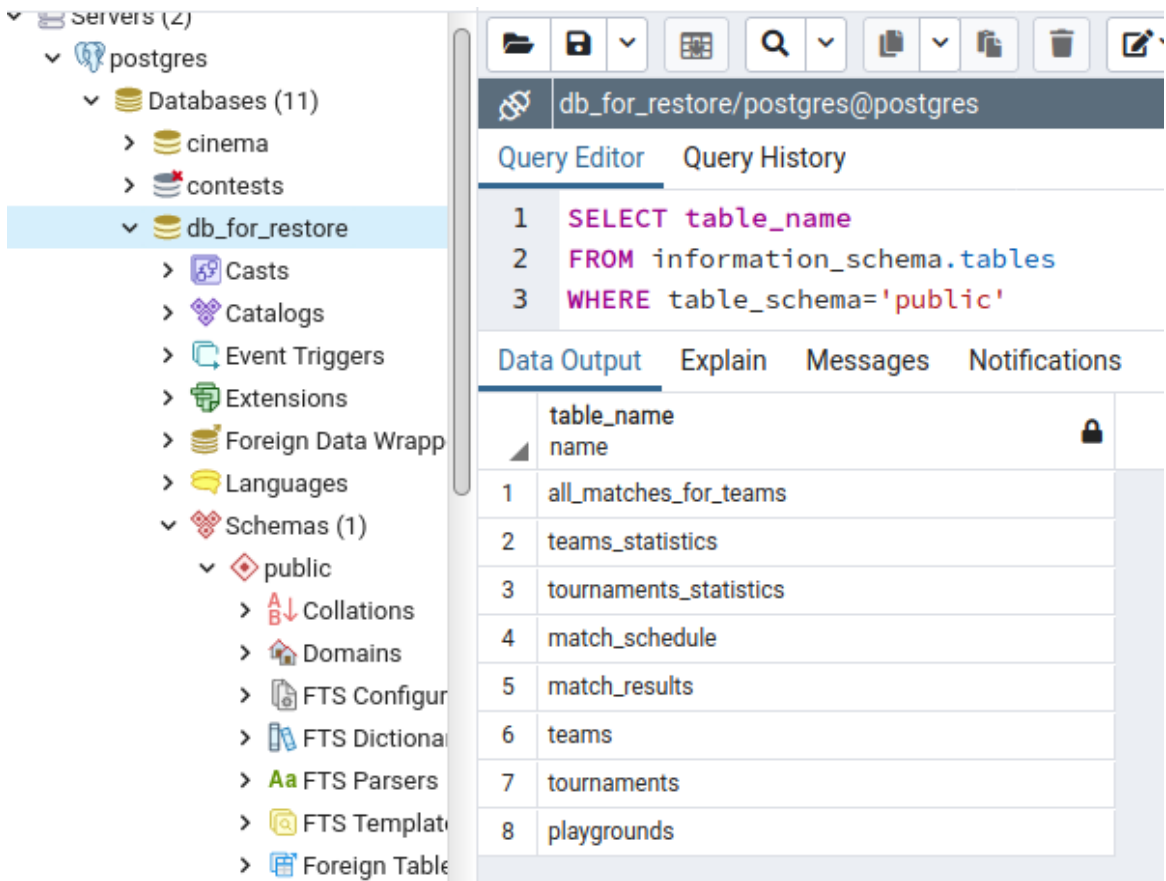
В первинній базі даних (ліворуч, @postgres) було створено новий запис. За допомогою запиту в базі даних реплікації (@slave) перевіряємо наявність відповідного запиту

ДОДАТОК 8 Демонстрація роботи резервування та відновлення

```
ivan@ivan-HP-250-G4-Notebook-PC:~$ time sudo ./dump_tournaments.sh
real    0m0,189s
user    0m0,062s
sys     0m0,016s
ivan@ivan-HP-250-G4-Notebook-PC:~$ ls -l /var/my_dir
total 1872
-rw-r--r-- 1 root root 49352 rpy 14 16:39 tournaments_dump_1607956779
-rw-r--r-- 1 root root 1861912 rpy 14 17:10 tournaments_dump_1607958633
ivan@ivan-HP-250-G4-Notebook-PC:~$ time sudo psql -U postgres -d db_for_restore < /var/my_dir/tournaments_dump_1607958633
SET
SET
SET
SET
SET
set_config
-----
```

```
real    0m3,100s
user    0m0,086s
sys     0m0,011s
ivan@ivan-HP-250-G4-Notebook-PC:~$
```

ДОДАТОК 9 Результат роботи відновлення в pgAdmin4



ДОДАТОК 10 Індексування поля “tournaments_id” таблиці “teams

Без індексів:	З BTree-індексом:
<pre>explain analyse SELECT * FROM teams WHERE tournaments_id=17;</pre> <div><div>Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div> <div>QUERY PLAN</div> <div>text</div> <div>Seq Scan on teams (cost=0.00..1038.01 rows=4022 width=21) (ac...</div> <div>Filter: (tournaments_id = 17)</div> <div>Rows Removed by Filter: 50321</div> <div>Planning Time: 0.135 ms</div> <div>Execution Time: 5.705 ms</div>	<pre>create index btree_teams_tournaments_ explain analyse SELECT * FROM teams WHERE tournaments_id=17;</pre> <div><div>Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div> <div>QUERY PLAN</div> <div>text</div> <div>Bitmap Heap Scan on teams (cost=79.46..488.74 rows=4022</div> <div>Recheck Cond: (tournaments_id = 17)</div> <div>Heap Blocks: exact=28</div> <div>-> Bitmap Index Scan on btree_teams_tournaments_id (cos</div> <div>Index Cond: (tournaments_id = 17)</div> <div>Planning Time: 0.295 ms</div> <div>Execution Time: 2.178 ms</div>


ДОДАТОК 11 Оптимізація з'єднання таблиць

Hash Join — 375ms	Merge Join — 343 ms
<pre>set enable_hashjoin to on; explain analyse select * from all_matches_for_</pre>	<pre>set enable_hashjoin to off; explain analyse select * from all_matches_for_teams</pre>
<p>Output Explain Messages Notifications</p> <p>QUERY PLAN text</p> <p>Unique (cost=33923.26..37664.44 rows=213782 width=24) (actual time=2</p> <p>-> Sort (cost=33923.26..34457.71 rows=213782 width=24) (actual time=</p> <p>Sort Key: s.id, s.tournaments_id, s.teams_id_1, s.teams_id_2, r.points_</p> <p>Sort Method: external merge Disk: 6344kB</p> <p>-> Append (cost=1567.06..10611.87 rows=213782 width=24) (actual</p> <p>-> Hash Left Join (cost=1567.06..3702.57 rows=106891 width=24</p> <p>Hash Cond: (s.id = r.match_schedule_id)</p> <p>-> Seq Scan on match_schedule s (cost=0.00..1854.91 rows=1</p> <p>-> Hash (cost=865.36..865.36 rows=56136 width=12) (actual t</p> <p>Buckets: 65536 Batches: 1 Memory Usage: 2925kB</p> <p>-> Seq Scan on match_results r (cost=0.00..865.36 rows=56136</p> <p>-> Hash Left Join (cost=1567.06..3702.57 rows=106891 width=24</p> <p>Hash Cond: (s_1.id = r_1.match_schedule_id)</p> <p>-> Seq Scan on match_schedule s_1 (cost=0.00..1854.91 rows=</p> <p>-> Hash (cost=865.36..865.36 rows=56136 width=12) (actual t</p> <p>Buckets: 65536 Batches: 1 Memory Usage: 2925kB</p> <p>-> Seq Scan on match_results r_1 (cost=0.00..865.36 rows=</p> <p>Planning Time: 0.658 ms</p> <p>Execution Time: 374.261 ms</p>	<p>Output Explain Messages Notifications</p> <p>QUERY PLAN text</p> <p>Unique (cost=39414.13..43155.31 rows=213782 width=24) (</p> <p>-> Sort (cost=39414.13..39948.58 rows=213782 width=24)</p> <p>Sort Key: s.id, s.tournaments_id, s.teams_id_1, s.teams_i</p> <p>Sort Method: external merge Disk: 6344kB</p> <p>-> Append (cost=0.62..16102.74 rows=213782 width=2</p> <p>-> Merge Left Join (cost=0.62..6448.00 rows=10689</p> <p>Merge Cond: (s.id = r.match_schedule_id)</p> <p>-> Index Scan using match_schedule_pkey on ma</p> <p>-> Index Scan using btree_match_results_id on m</p> <p>-> Merge Left Join (cost=0.62..6448.00 rows=10689</p> <p>Merge Cond: (s_1.id = r_1.match_schedule_id)</p> <p>-> Index Scan using match_schedule_pkey on ma</p> <p>-> Index Scan using btree_match_results_id on m</p> <p>Planning Time: 0.642 ms</p> <p>Execution Time: 343.663 ms</p>

ДОДАТОК 12 Оптимізація запиту

Не оптимізований	Оптимізований
<p>tournaments/postgres@postgres</p> <p>Query Editor Query History</p> <pre>1 explain analyse 2 select * 3 from teams_statistics_in_tournament_opt(17)</pre> <p>Data Output Explain Messages Notifications</p> <p>QUERY PLAN text</p> <p>1 Function Scan on teams_statistics_in_to...</p> <p>2 Planning Time: 0.030 ms</p> <p>3 Execution Time: 92.767 ms</p>	<p>tournaments/postgres@postgres</p> <p>Query Editor Query History</p> <pre>1 explain analyse 2 select * 3 from teams_statistics_in_tournament(17)</pre> <p>Data Output Explain Messages Notifications</p> <p>QUERY PLAN text</p> <p>1 Function Scan on teams_statistics_in_tournament (cost=0.25..10.2</p> <p>2 Planning Time: 0.028 ms</p> <p>3 Execution Time: 538.811 ms</p>

ДОДАТОК 13 Генерація помилки при вставці некоректних даних у таблицю «match_schedule»


 tournaments/postgres@postgres

[Query Editor](#) [Query History](#)

```
1 insert into match_schedule(tournaments_id, teams_id_1, teams_id_2, playgrounds_id)
2 values(1,1,1,1)
```

[Data Output](#) [Explain](#) [Messages](#) [Notifications](#)

ERROR: team cannot play with itself
CONTEXT: PL/pgSQL function check_valid_data_match_schedule() line 8 at RAISE
SQL state: P0001

 tournaments/postgres@postgres

[Query Editor](#) [Query History](#)

```
1 insert into match_schedage(tournaments_id, teams_id_1, teams_id_2, playgrounds_id)
2 values(1,1,1000,1)
```

[Data Output](#) [Explain](#) [Messages](#) [Notifications](#)

ERROR: different tournaments id for team1, team2, match (1 11 1)
CONTEXT: PL/pgSQL function check_valid_data_match_schedule() line 27 at RAISE
SQL state: P0001

ДОДАТКИ А. Консольний інтерфейс

ДОДАТОК А1 Інтерфейси меню

```
There are 5 commands in main menu:
0) tournament
1) team
2) search
3) generate
4) exit
Enter commands id:
(integer)
0
main command: tournament is selected
There are 6 commands in tournaments menu:
0) get_all
1) get_by_id
2) insert
3) update
4) delete
5) exit
Enter commands id:
(integer)
5
go to main
There are 5 commands in main menu:
```

```
main command: search is selected
There are 5 commands in search menu:
0) get_statistic_tournaments
1) get_statistic_teams_in_tournament
2) tournaments_schedule
3) tournaments_schedule_in_time_range
4) exit
Enter commands id:
(integer)
4
go to main
```

```
main command: generate is selected
There are 2 commands in generate menu:
0) generate_tournaments
1) exit
Enter commands id:
(integer)
1
go to main
```

ДОДАТОК А2 Приклади обробки некоректних даних користувача

- відповідність типу введеного значення

- Цілочисельний тип:

```
Enter integer:
jkljlj
entered value is not integer invalid literal for int() with base 10: 'jkljlj'
```

- Час:

```
scheduled_start:
(time in HH:MM format)
10-20
entered value is not time time data '10-20' does not match format '%H:%M'
(time in HH:MM format)
24:20
entered value is not time time data '24:20' does not match format '%H:%M'
```

- Дата

```
Enter tournament:
name:
(string)
jkl
start date:
(date in format YYYY-mm-dd)
2000:10:10
entered value is not date time data '2000:10:10' does not match format '%Y-%m-%d'
(date in format YYYY-mm-dd)
cspdofvkpserv
entered value is not date time data 'cspdofvkpserv' does not match format '%Y-%m-%d'
(date in format YYYY-mm-dd)
2020-20-20
entered value is not date time data '2020-20-20' does not match format '%Y-%m-%d'
```

- відповідність множині значень (якщо існують обмеження)

```
id:
(integer)
-5
entered value is not correct
```

ДОДАТОК А3 Підтвердження виводу сутностей на екран

```
teams command: get_all is selected
[*] get_all | Executing time: 0.1189124584197998 sec.
list (321 elements):
print collection? ('yes'/any)
yes
id: 1; name: EKN; tournaments_id: 1; registered_at: 2018-09-28 05:18:41.206186;
id: 2; name: YHDMKG; tournaments_id: 1; registered_at: 2001-10-28 13:18:38.506452;
```

ДОДАТОК А4 Обрання типу виведення статистик

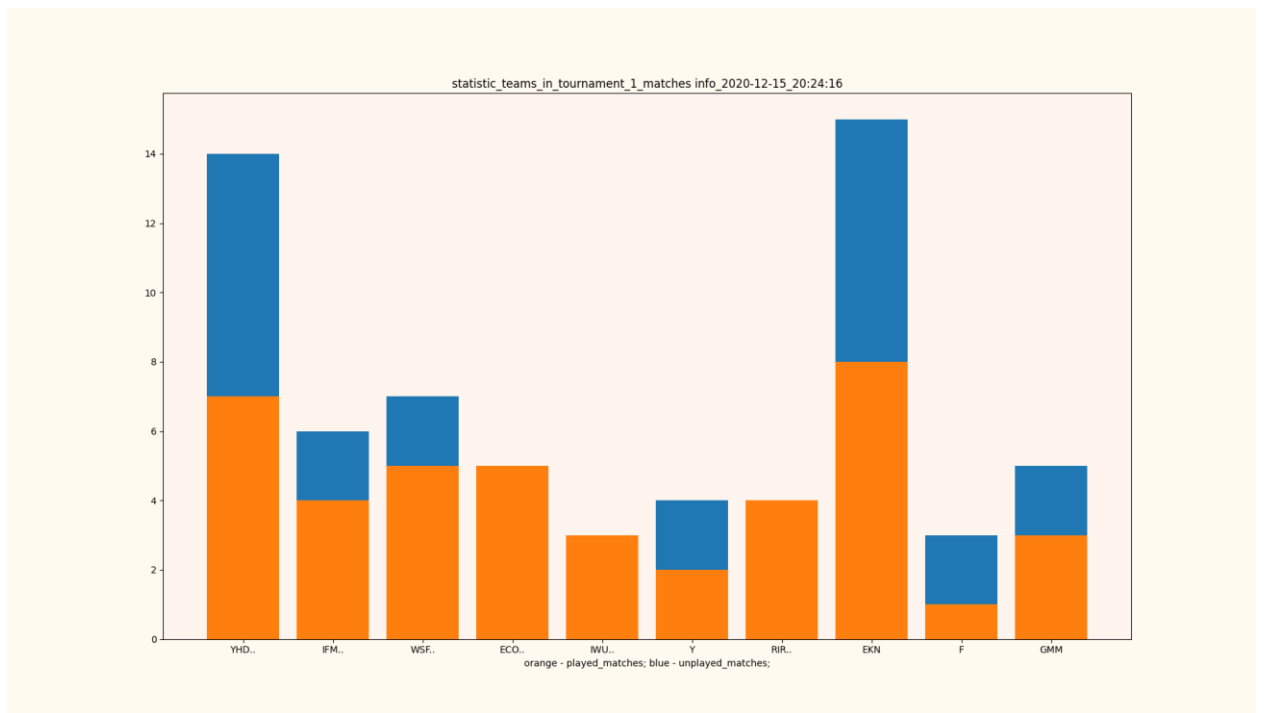
```
search command: get_statistic_tournaments is selected
[*] get_statistic_tournaments | Executing time: 1.1946439743041992 sec.
Enter type of display:
'png' to chart or any for console output
png
print a number of command
```

ДОДАТОК А5 Меню обрання типу діаграм

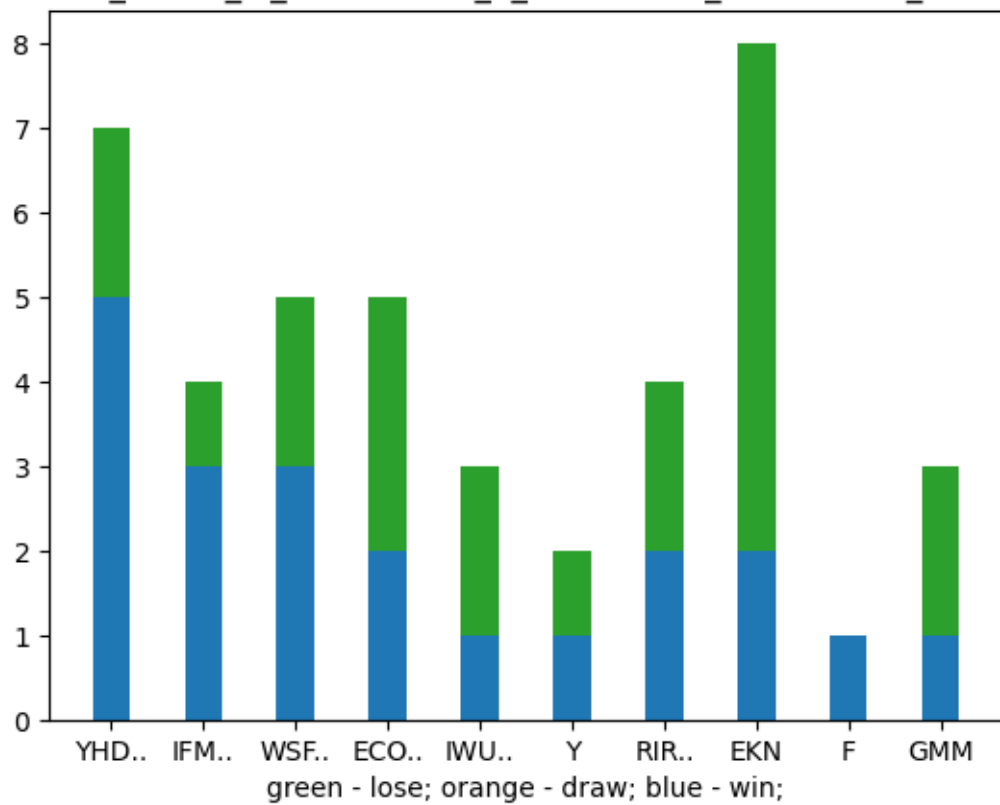
```
search command: get_statistic_tournaments is selected
[*] get_statistic_tournaments | Executing time: 1.1946439743041992 sec.
Enter type of display:
'png' to chart or any for console output
png
print a number of command
0. teams_count
1. match_count
2. played_match
3. unplayed_match
4. sum_score
5. sum_other_score
6. max_score
7. avg_score
8. min_score
(integer)
0
file 'statistic_tournaments_teams_count_2020-12-15_19:54:13' created
There are 5 commands in search menu:
```

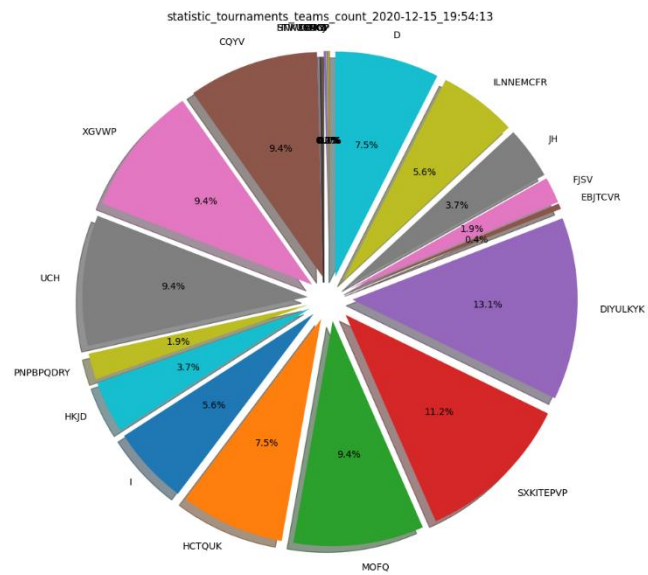
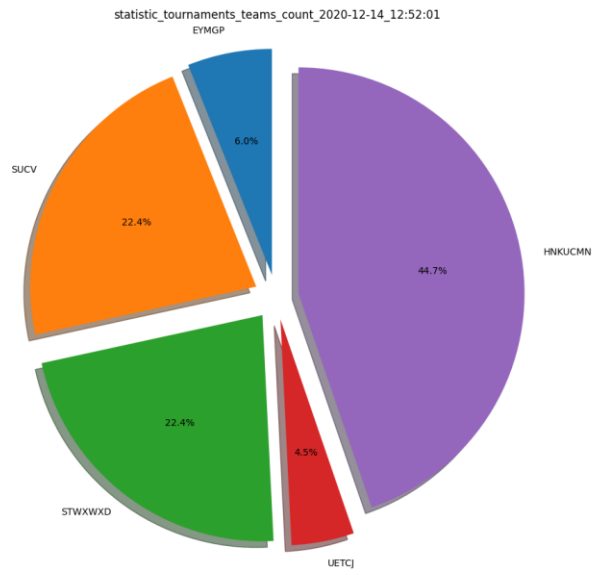
```
search command: get_statistic_teams_in_tournament is selected
Enter tournaments id:
(integer)
10
[*] get_statistic_teams_in_tournament | Executing time: 0.16887140274047852 sec.
Enter type of display:
'png' to chart or any for console output
png
print a number of command
0. matches info
1. results info
(integer)
1
```

ДОДАТКИ Б. Створені діаграми



statistic_teams_in_tournament_1_results info_2020-12-15_20:23:43





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 видів 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 види 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)

```