Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторным работам
по дисциплине
«Базы данных»

выполнил:
студент группы 3530901/70202
Павлов Д.В.
преподаватель:
Мяснов А.В.

Санкт-Петербург
2020

Содержание

Лабораторная работа номер 1 3
Лабораторная работа номер 2 8
Лабораторная работа номер 3 11
Лабораторная работа номер 4 14
Приложение 1..... 24
Приложение 2..... 28
Приложение 3..... 40

Лабораторная работа номер 1

1. Цель работы

Познакомиться с основами проектирования схемы БД, способами организации данных в SQL-БД.

2. Программа работы

1. Создание проекта для работы в GitLab.
2. Выбор задания (предметной области), описание набора данных и требований к хранимым данным в свободном формате в wiki своего проекта в GitLab.
3. Формирование в свободном формате (предпочтительно в виде графической схемы) схемы БД, соответствующей заданию. Должно получиться не менее 7 таблиц.
4. Согласование с преподавателем схемы БД. Обоснование принятых решений и соответствия требованиям выбранного задания.
5. Выкладывание схемы БД в свой проект в GitLab.
6. Демонстрация результатов преподавателю.
7. Самостоятельное изучение SQL-DDL.
8. Создание скрипта БД в соответствии с согласованной схемой. Должны присутствовать первичные и внешние ключи, ограничения на диапазоны значений. Демонстрация скрипта преподавателю.
9. Создание скрипта, заполняющего все таблицы БД данными.
10. Выполнение SQL-запросов, изменяющих схему созданной БД по заданию преподавателя. Демонстрация их работы преподавателю.

3. Выполнение работы

3.1. Выбор задания

В качестве задания была выбрана тема футбольных лиг. База данных будет хранить информацию о лигах, сезонах, командах, игроках и матчах.

3.2. Описание таблиц

- **league** – таблица всех лиг.
  1) **id** – идентификатор
  2) **league_** – название лиги

- **team –** таблица всех команд
  1) **id** – идентификатор
  2) **league_id –** идентификатор лиги
  3) **team_** – название команды
- **season –** таблица всех сезонов
  1) **id** – идентификатор
  2) **season_** – название сезона

- **personal_data –** таблица всех футболистов
  1) **id** – идентификатор
  2) **team_id** – идентификатор команды
  3) **name** – имя и фамилия
  4) **birthday** – день рождения
  5) **nationality** – национальность
  6) **weight** – вес
  7) **height** – рост

- **contract –** таблица всех контрактов
  1) **id** – идентификатор
  2) **personal_data_id** – идентификатор футболиста
  3) **fee –** зарплата
  4) **date_from –** начало контракта
  5) **date_to –** окончание контракта

- **match –** таблица всех матчей
  1) **id** – идентификатор
  2) **team1_id** – идентификатор команды хозяина
  3) **team2_id** – идентификатор команды гостя
  4) **result** – результат матча
  5) **season_id** – идентификатор сезона
  6) **date** – дата матча

- **team_position –** таблица позиций команд
  1) **id** – идентификатор
  2) **league_id** – идентификатор лиги
  3) **season_id** – идентификатор сезона
  4) **team_id** – идентификатор команды
  5) **week –** игровая неделя
  6) **w_position –** позиция на этой неделе
  7) **points –** очки этой недели

- **player –** таблица футболистов в определенном сезоне
  1) **id** – идентификатор
  2) **personal_data_id** – идентификатор футболиста
  3) **position –** позиция на поле
  4) **season_id** – идентификатор сезона

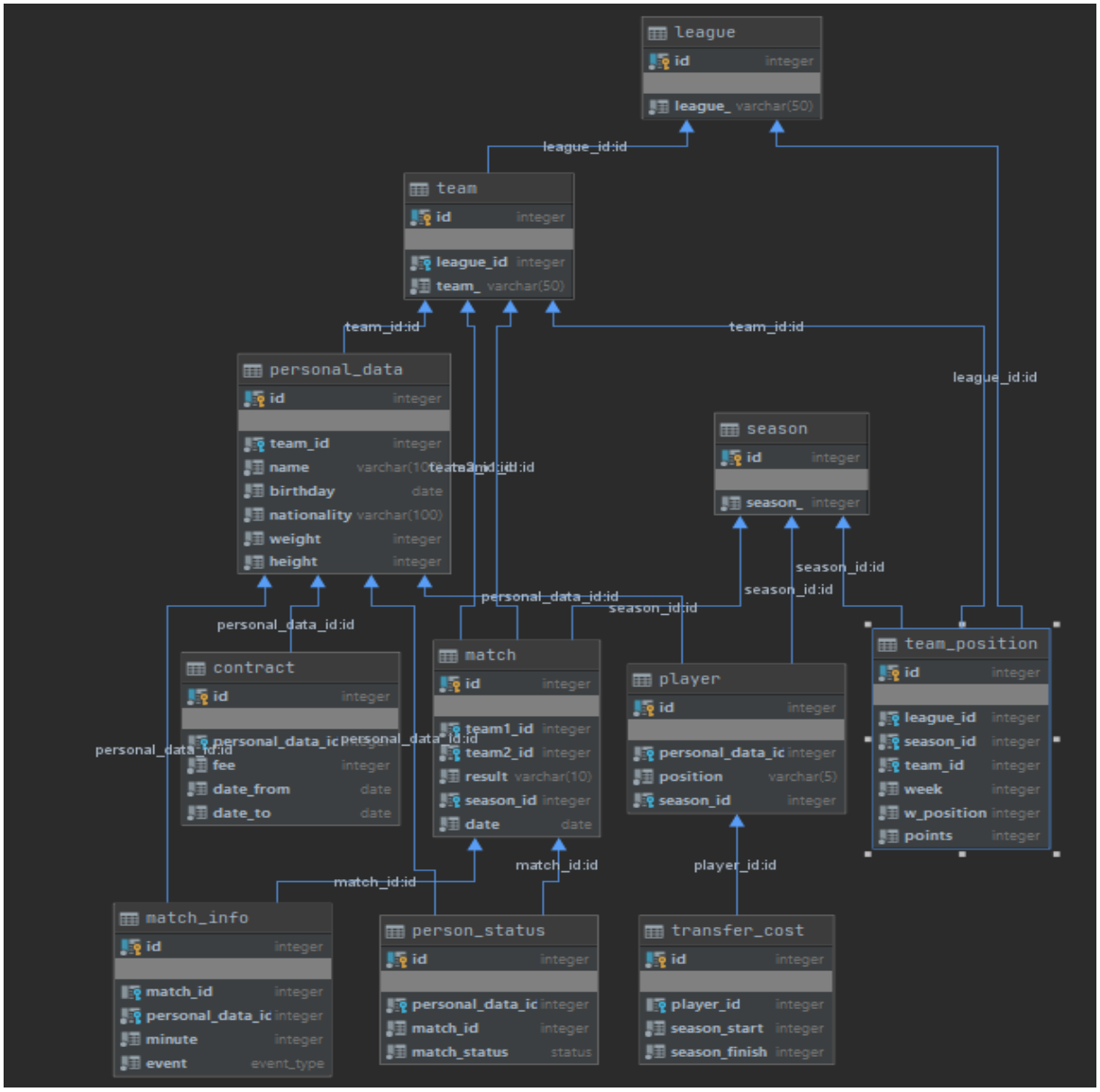
- **match_info** – таблица событий матчей
  1) **id** – идентификатор
  2) **match_id** – идентификатор матча
  3) **personal_data_id** – идентификатор футболиста
  4) **minute –** минута события
  5) **event –** событие (гол, ассист, замена)

- **person_status –** таблица статуса игроков в матчах
  1) **id** – идентификатор
  2) **personal_data_id** – идентификатор футболиста
  3) **match_id** – идентификатор матча
  4) **match_status –** статус игрока в матче (замена или основа)

- **transfer_cost –** таблица стоимости игроков
  1) **id** – идентификатор
  2) **player_id** – идентификатор футболиста в определенном сезоне
  3) **season_start –** стоимость в начале сезона
  4) **season_finish –** стоимость в конце сезона

3.3. Структура базы данных

Полученная схема представлена на рисунке:



По заданию преподавателя схема БД была изменена для удовлетворения следующим требованиям:

1) Сформировать таблицу для хранения текущего расположения команд в турнирной таблице.
2) Типизировать события, которые могут быть выполнены игроком на поле.

3) Реализовать учет участников конкретных матчей: заявка, основа, замены.

Скрипт создания базы данных вместе с изменениями представлен в Приложении 1.

4. Выводы

В ходе выполнения данной лабораторной работы я познакомился с основами проектирование схемы базы данных. Было выделено несколько основных сущностей (информация об игроках, матчи) и множество вспомогательных. На практике были применены такие виды связи как один ко одному и один ко многим.

Также были изучены основы создания скриптов на языке SQL. С помощью SQL-DDL были описаны структуры разрабатываемой схемы БД. Было проведено знакомство с первичными и внешними ключами, на значения были наложены ограничения в виде диапазонов.

Лабораторная работа номер 2

1. Цель работы

Сформировать набор данных, позволяющий производить операции на реальных объемах данных.

2. Программа работы

1. Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.
2. Частные требования к генератору, набору данных и результирующему набору данных:
   - количество записей в справочных таблицах должно соответствовать ограничениям предметной области
   - количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации
   - значения для внешних ключей необходимо брать из связанных таблиц
   - сохранение уже имеющихся данных в базе данных.

3. Выполнение работы

Программа параметризуемого генератора была реализована на ЯП Python. Для взаимодействий с базой данных использовалась библиотека psycopg2 для PostgreSQL.

Во время работы генератор последовательно запрашивает разрешение на заполнение таблиц либо их очистку, все это происходит путем ввода данных в терминал с использованием функции input(), 1 – true \ 0 - false. Запуск происходит путем вызова функции main. Разберем по порядку:

1) Clear the table?[1/0] – если соглашаемся, происходит полная очистка таблицы.
2) Generate leagues?[1/0] – если соглашаемся, мы вбиваем желаемое количество лиг для генерации и заполняем их из заготовленного файла с

лигами. Если мы не соглашаемся, либо вводим 0, то заполнение лиг\команд\персональных данных\контрактов происходить не будет.

3) Generate teams?[1/0] – если соглашаемся, заполняем лиги командами из заготовленного файла с командами в количестве 20 штук на лигу. Если мы не соглашаемся, то команды, персональные данные и контракты заполнены не будут.
4) Generate personal datas?[1/0] – если соглашаемся, мы вбиваем желаемое количество игроков для каждой команды и заполняем их из заготовленного файла с игроками. Если мы не соглашаемся, то персональные данные и команды заполнены не будут.
5) Generate contracts?[1/0] – если соглашаемся, заполняем.
6) Generate seasons?[1/0] – если соглашаемся, мы вбиваем желаемое количество сезонов, далее по очереди пишем их названия (обычно это год старта сезона). Если мы не соглашаемся, либо вводим 0, то заполнение сезонов\игроков сезона\трансферной стоимости происходить не будет.
7) Generate players?[1/0] – если соглашаемся, для каждого игрока генерируется его профиль для сезона. Если мы не соглашаемся, заполнение игроков сезона и их трансферной стоимости не будет.
8) Generate transfer cost?[1/0] – если соглашаемся заполняем.
9) Generate matches?[1/0] – если соглашаемся, для каждой лиги в каждом сезоне генерируются матчи и результаты команд на каждой неделе. Если не соглашаемся, матчи и их информация сгенерированы не будут.
10) Generate match infos?[1/0] – если соглашаемся, для каждого матча будем сгенерирована информация о нем и статус игроков на матч.

Коннект к БД происходит с помощью psycopg2.connect(имя базы, имя пользователя, пароль, хост). Каждая запись вставляется в таблицу с помощью написанной функции def insert_in_table.

Код генератора вместе с изменениями, внесенными после завершения индивидуального задания по 1 лабораторной работе, представлен в Приложении 2.

4. Выводы

В ходе выполнения данной лабораторной работы был разработан генетарор тестовых данных на ЯП Python с использованием psycopg2 (чтобы не писать SQL-запросы вручную).

Были получены практические навыки взаимодействия с БД через собственную программу, что помогло лучше разобраться с ключами и связями между таблицами.

Лабораторная работа номер 3

1. Цель работы

Познакомить студентов с языком создания запросов управления данными SQL-DML.

2. Программа работы

1. Изучение SQL-DML.
2. Выполнение всех запросов из списка стандартных запросов. Демонстрация результатов преподавателю.
3. Получение у преподавателя и реализация SQL-запросов в соответствии с индивидуальным заданием. Демонстрация результатов преподавателю.
4. Сохранение в БД выполненных запросов SELECT в виде представлений, запросов INSERT, UPDATE или DELETE -- в виде ХП. Выкладывание скрипта в GitLab.

3. Выполнение работы

3.1. Выполнение стандартных запросов.

Для изучения SQL-DML были проделаны все запросы, представленные на странице лабораторной работы. Ниже отдельно кратко расскажу о назначении SELECT, INSERT, UPDATE, DELETE.

1) Для извлечения записей из таблиц в SQL определен оператор SELECT. Этот оператор возвращает ни одного, одно или множество строк, удовлетворяющих указанному условию и упорядоченных по заданному критерию.
2) Для добавления записи в таблицу в SQL определен оператор INSERT.
3) Для модификации записи в таблице в SQL определен оператор UPDATE.
4) Для удаления записей из таблицы в SQL определен оператор DELETE. Этот оператор удаляет ни одного, одно или множество строк, удовлетворяющих указанному условию.

3.2. Индивидуальные задания

Задание 1. Вывести 10 футболистов, которые больше всего голов забивают во втором тайме.

Запрос:

```
select personal_data.id, name, count(event) as cnt
from personal_data
        JOIN match_info ON (personal_data.id = personal_data_id and minute > 45 and event = 'goal')
group by personal_data.id, name
order by count(event) DESC LIMIT 10;
```

Его результат:

	id	name	cnt
1	2106	VICTOR KRUG	31
2	2131	CAREY AVENDANO	31
3	1949	MOHAMMAD COURTNAGE	29
4	883	SANDY GROCH	28
5	931	SAMUEL REGINALDO	28
6	1467	DARELL SONNENBERG	26
7	1008	STEVIE TAUBERT	26
8	779	ERWIN POREE	26
9	538	MODESTO WERN	26
10	1354	FREDDY SHUKERT	26

Задание 2. Вывести 5 футболистов, которых реже всего заменяют.

Запрос:

```
select personal_data.id, name, count(event) as cnt
from personal_data
        JOIN match_info ON (personal_data.id = personal_data_id and event = 'replaced')
group by personal_data.id
order by count(event) LIMIT 5
```

Его результат:

	id	name	cnt
1	1460	LYMAN NANIA	6
2	432	NORBERT AKONI	8
3	1415	WILBER SCHWARZLOSE	9
4	842	HUGH TOYAMA	9
5	1445	ART FECK	9

4. Выводы

В ходе выполнения данной лабораторной работы было проведено ознакомление с языком создания запросов и управления данными SQL-DML. В предыдущих лабораторных работах тоже приходилось с ним сталкиваться, но запросы были примитивными.

Была проведена работа с выборкой данных, их вставкой, удалением, модификацей. Проделаны все стандартные запросы и выполнены индивидуальные задания. Отдельно хочется упомянуть оператор JOIN и его разновидности:

1) INNER – внутреннее соединение 2 таблиц по определенному условию.
2) LEFT OUTER – выборка будет содержать все строки из левой таблицы (первой). Сначала выполняется внутреннее соединение, а потом добавляются все строки из левой таблицы, которым не соответствуют никакие строки из правой, а вместо значений правой таблицы вставляются NULL.
3) RIGHT OUTER – выборка будет содержать все строки из правой таблицы (второй). Сначала выполняется внутреннее соединение, а потом добавляются все строки из правой таблицы, которым не соответствуют никакие строки из левой, а вместо значений левой таблицы вставляются NULL.
4) FULL OUTER – выборка будет содержать все строки из обеих таблиц.
5) CROSS – каждая строка из одной таблицы соединяется с каждой строкой из второй таблицы.

Лабораторная работа номер 4

1. Цель работы

Знакомство студентов с проблемами, возникающими при высокой нагрузке на базу данных, и методами их решения, путем оптимизации запросов.

2. Программа работы

1. Написание параметризированных типовых запросов пользователей. Необходимо проанализировать БД и сформировать не менее 5 запросов, результаты которых требовались бы потенциальным пользователям данных. Запросы должны содержать соединения таблиц.
2. Моделирование нагрузки базы данных. Параметризируемое (количество запросов, количество потоков) приложение, где можно было моделировать нагрузку на БД и получать результаты выполнения в виде графиков и таблиц.
3. Снятие показателей работы сервиса и построение соответствующих графиков. Необходимо оценить зависимость времени ответа от количества потоков и от количества запросов в единицу времени. Также полезно иметь возможность оценить нарастание нагрузки с разной скоростью и зависимость показателей от скорости. При оценке показателей необходимо пытаться добиться ситуации, когда нагрузка "кладет" БД.
4. Применение возможных оптимизаций запросов и повторное снятие показателей. Анализ планов запросов, формирование предложений по снижению времени выполнения запросов. Последовательное применение шагов по оптимизации и снятие показателей после каждого шага.
5. Сравнительный анализ результатов.
6. Демонстрация результатов преподавателю.

3. Выполнение работы

3.1. Реализация оптимизации

Как и во 2 лабораторной работе, был выбран ЯП Python с psycopg2. Оптимизацию решено было делать с помощью PREPARE и CREATE INDEX, а анализировать ситуацию – с помощью EXPLAIN ANALYSE. Далее о каждом немного подробнее:

1) PREPARE – при выполнении данной команды указанный оператор анализируется и переписывается (становится подготовленным). Далее при выполнении EXECUTE подготовленный оператор планируется и исполняется. Это придумано для того, чтобы запрос каждый раз повторно не разбирался, а также позволяет выбрать лучший план выполнения. Также, в такой оператор можно подставлять свои значения во время выполнения.
   Данный способ, конечно, ускоряет время ответа БД, но, так-как это всего лишь заготовка запроса, ожидать *сильного* ускорения не стоит. А вот о том, от кого стоит, пойдет речь ниже.

2) CREATE INDEX – создает индексы по указанному столбцу. Для них можно добавлять условие WHERE, если оно требуется. В то же время излишняя индексация может привести к противоположному эффекту, когда неоптимизированный запрос выполняется быстрее такого *оптимизированного*. Данный способ может очень сильно ускорить работу, так как в будущем БД будет знать, что на данный столбец создан индекс и начнет поиск сначала по индексу, начиная с корня и спускаясь по узлам до тех пор, пока не найдет искомое значение(ия). В итоге вместо того, чтобы проверять каждое значение, которое могло бы занять очень много времени, мы находим результат быстро.

3) ANALYSE – инструмент анализа запросов. Показывает оценку стоимости выполнения данного узла, которую сделал для него планировщик. Это значение он старается минимизировать. В выводе мы имеем такие показатели, как стоимость до вывода данных и общая стоимость (нужна если мы доходим до конца, иначе не актуально), число строк (до конца) и размер строк в байтах.
   Точность оценок планировщика можно проверить, используя команду EXPLAIN ANALYSE. С этим параметром EXPLAIN на самом деле

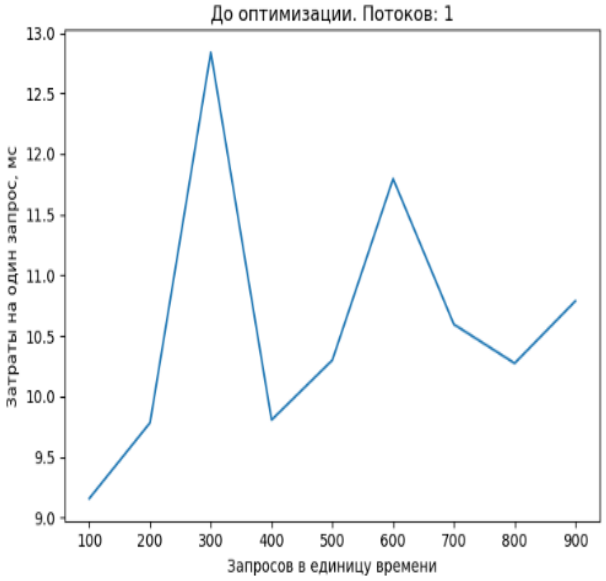
выполнит запрос и нам становится доступна дополнительная статистика. Можно увидеть подробный разбор каждого узла выполнения запроса, ключ сортировки, метод сортировки (некоторые методы можно выключать с помощью команды SET, либо наоборот включать. Делается для того, если мы уверены, что данный метод будет медленно работать в запросе, и чтобы планировщик его не выбирал, как и для метода сканирования), метод сканирования и другое.
Выбирать строки по отдельности дороже, чем читать последовательно. Но если читать нужно не все страницы таблицы, то выбирать дешевле. Добавление условия уменьшает оценку числа результирующих строк, но не стоимость запроса, так как просматриваться будет тот же набор строк, что и раньше. Стоимость даже может увеличиться. Если таблица слишком маленькая для сканирования по id, происходит последовательное сканирование.

Перейдем непосредственно к описанию проводимой мною оптимизации (код представлен в Приложении 3).
Была импортирована библиотека matplotlib.pyplot для визуализации нагрузки на нашу БД. Написаны функции для неоптимизированных и оптимизированных запросов, а также функции оптимизации (используется PREPARE) и индексации (CREATE INDEX).
Сначала запускались неоптимизированные запросы. Максимальное количество запросов в единицу времени и количество одновременно работающих потоков было параметризировано и в процессе тестирования менялось. Если потоки задавались изначально и все, то количество запросов в единицу времени постепенно увеличивалось до введенного нами максимума.
Далее по отдельности и потом вместе происходила оптимизация и индексация.

3.3 Результаты оптимизации

1) До 900 запросов в секунду:

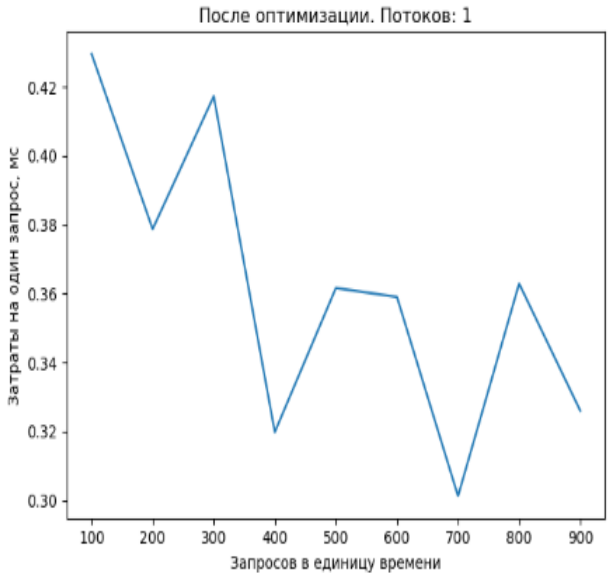
1.1) 1 поток



Not optimized

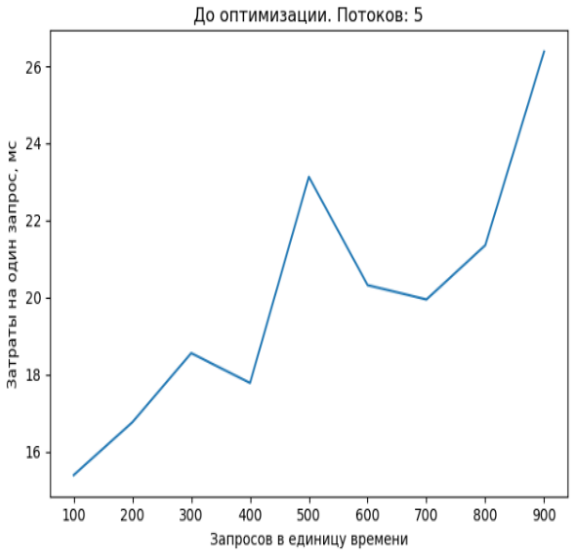
Prepare

Indexes

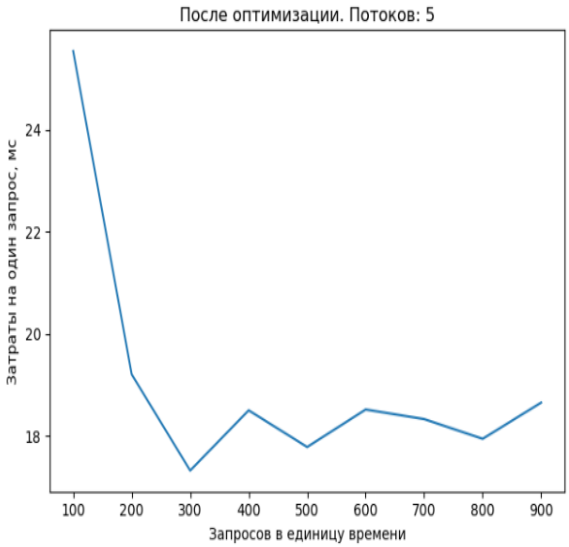


Prepare & Indexes

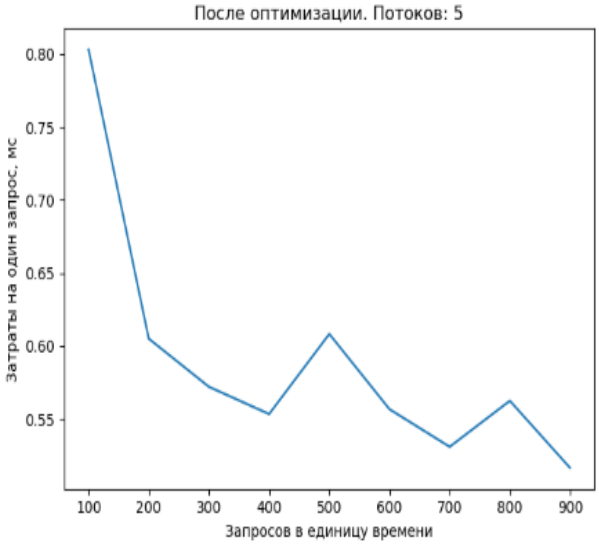
1.2) 5 потоков



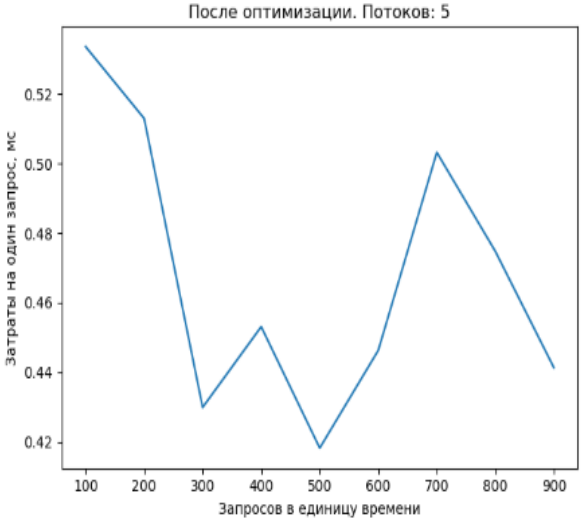
Not optimized



Prepare

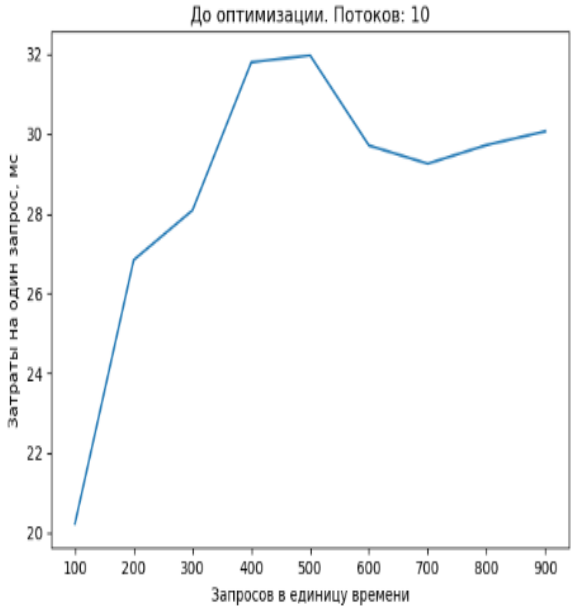


Indexes



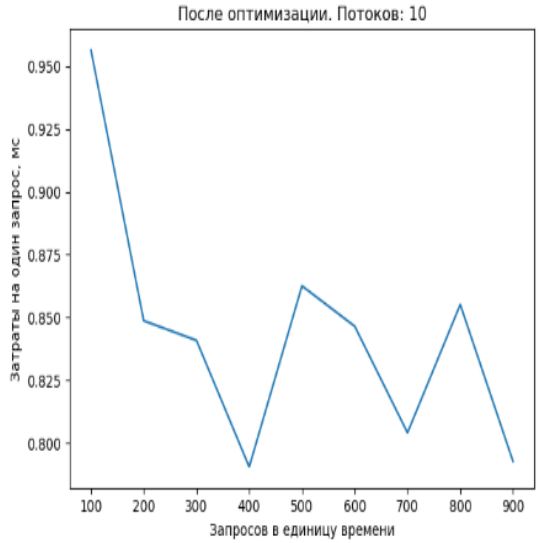
Prepare & Indexes

1.3) 10 потоков

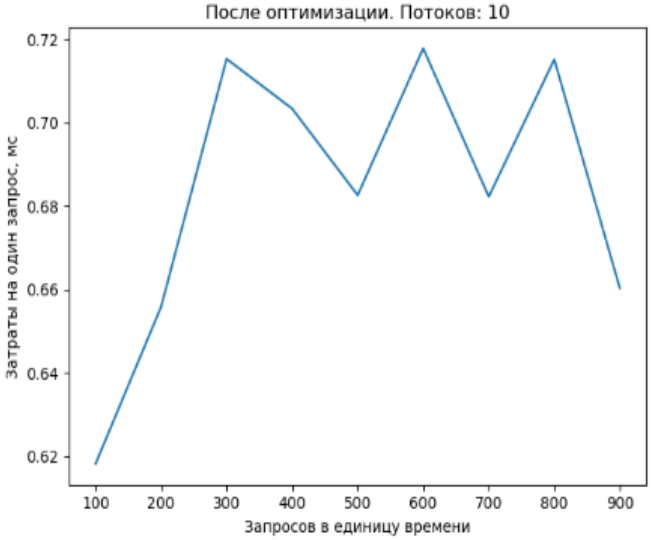


Not optimized

Prepare



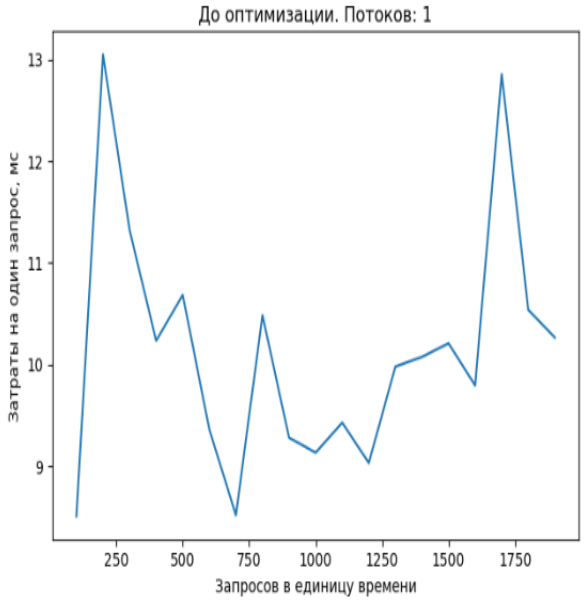
Indexes



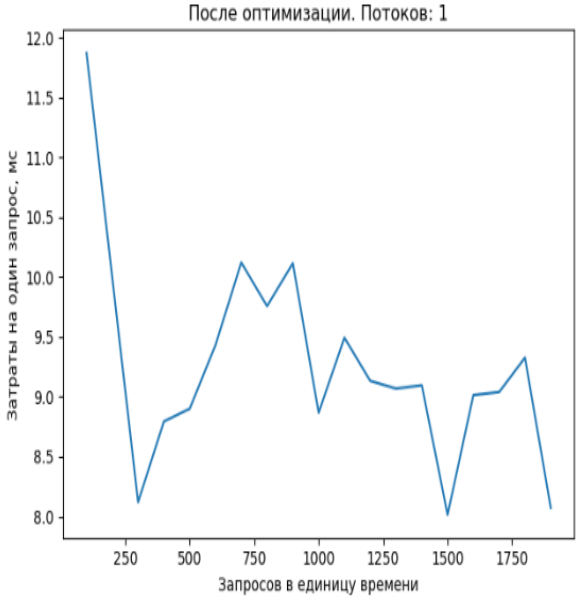
Prepare & Indexes

2) До 1900 запросов в секунду

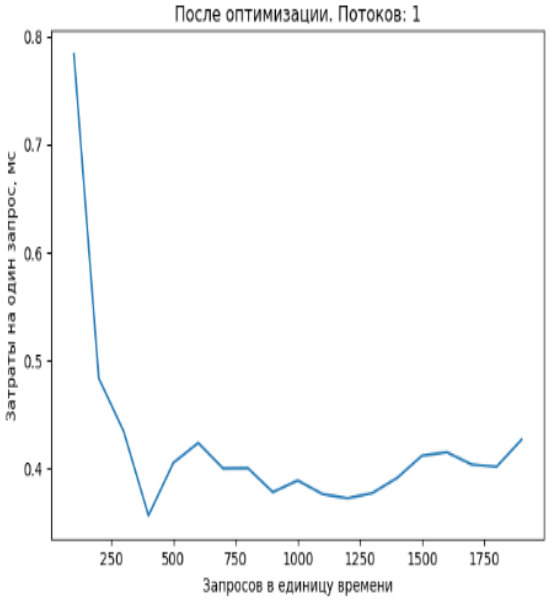
2.1) 1 поток



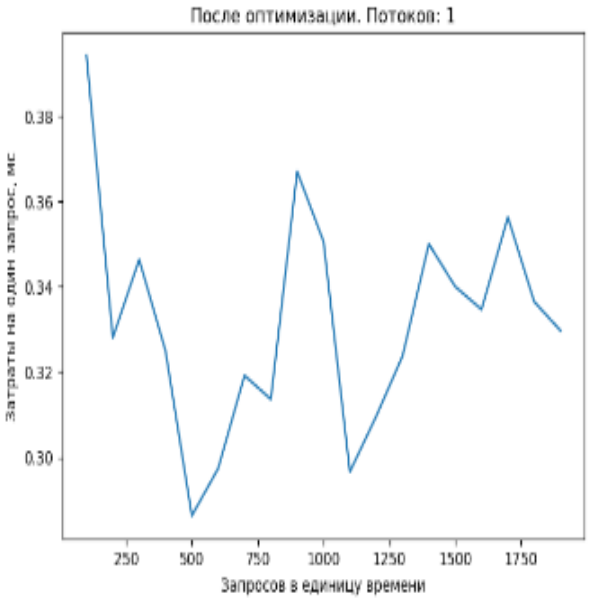
Not optimized



Prepare

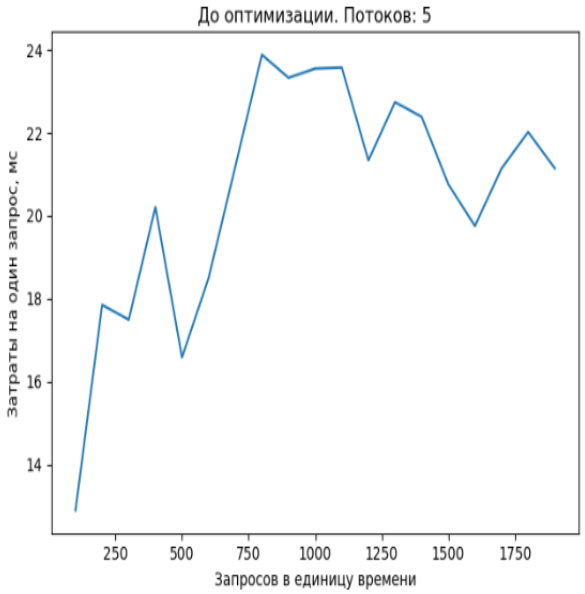


Indexes



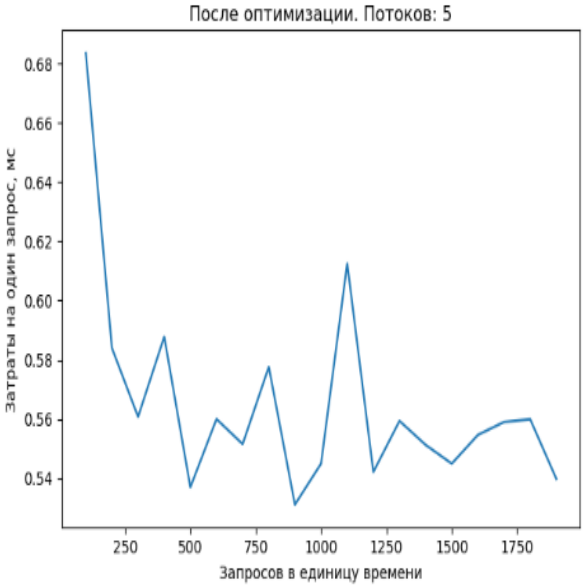
Prepare & Indexes

2.2) 5 потоков

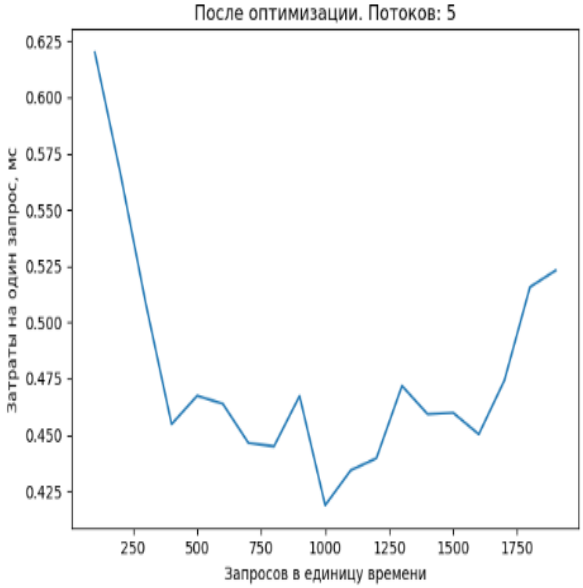


Not optimized

Prepare



Indexes

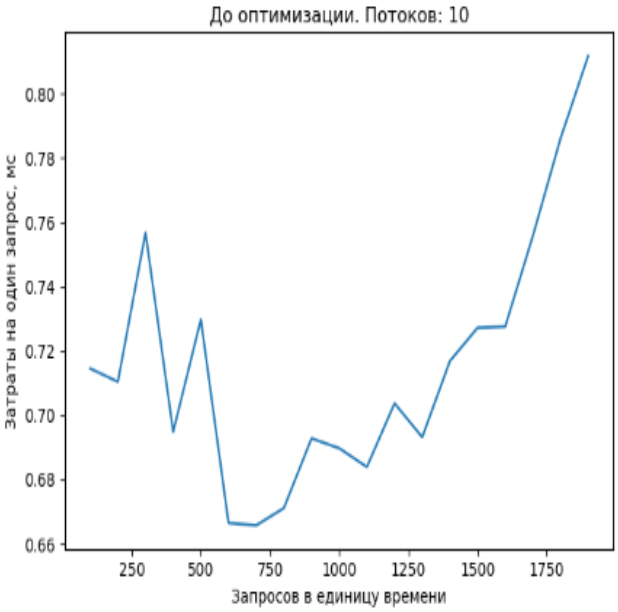


Prepare & Indexes

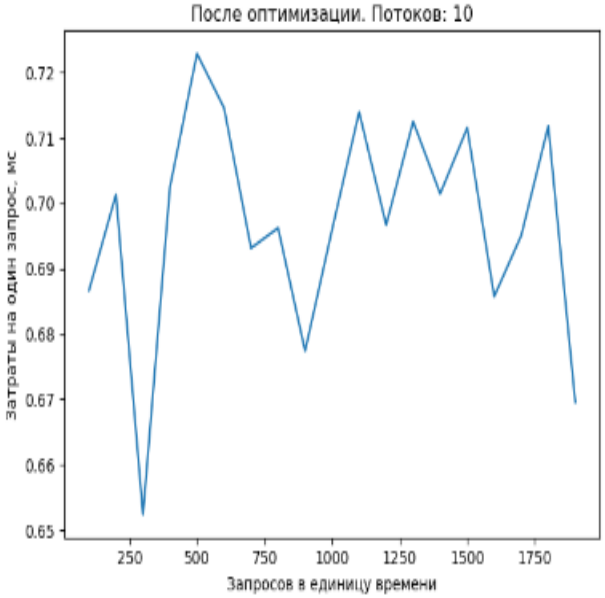
2.3) 10 потоков

Not optimized

Prepare



Indexes



Prepare & Indexes

4. Выводы

В ходе выполнения данной лабораторной работы было проведено ознакомление со структурой и инструментами, помогающими при оптимизации запросов пользователей к БД.

Исходя из выше приведенных графиков можно сделать несколько выводов:

1) Prepare ненамного, но все-таки ускоряет работу бд;
2) Indexes действуют очень ощутимо, разница с неоптимизированными запросами составила минимум 200%;
3) Использование prepare при созданных indexes ненамного, но все-таки ускоряет работу бд;
4) При увеличении максимального количества запросов в единицу времени до 10000, неоптимизированный тест “клал” бд, результата от оптимизированного теста пришлось ждать около 5-10 минут;
5) С увеличением количества потоков наблюдается явное увеличение затрат времени на запросы;
6) При увеличении количества запросов в единицу времени бд лишь дольше их обрабатывала, затраты времени на запрос оставались на одном уровне (возвращаяясь к пункту (4) – наблюдался эффект зависания бд, при увеличении количества неоптимизированных запросов до 10000);
7) Нельзя исключать фактор разных ситуаций в самой БД и фактор относительно небольшой мощности сервера (коим является ноутбук), из-за чего могли происходить “скачки” времени и долгая обработка;

Ниже приведен пример оптимизированных запросов с максимальным количеством 10000 в единицу времени (выполнялось до 10 минут):

Приложение 1

```
DROP TABLE IF EXISTS
league,team,personal_data,transfer_cost,contract,player,match,match_info,person_status,team_position;
DROP TYPE IF EXISTS status, event_type;

CREATE TABLE IF NOT EXISTS public.league
(
    id     SERIAL     NOT NULL,
    league_ VARCHAR(50) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.team
(
    id       SERIAL     NOT NULL,
    league_id int       NOT NULL,
    team_    VARCHAR(50) NOT NULL,
    PRIMARY KEY (id),

    CONSTRAINT league_id_foreign
        FOREIGN KEY (league_id) REFERENCES public.league (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION
);

CREATE TABLE IF NOT EXISTS public.personal_data
(
    id        SERIAL      NOT NULL,
    team_id    int        NOT NULL,
    name       VARCHAR(100) NOT NULL,
    birthday   date       NOT NULL,
    nationality VARCHAR(100) NOT NULL,
    weight     int        NOT NULL,
    height     int        NOT NULL,
    PRIMARY KEY (id),

    CONSTRAINT team_id_foreign
        FOREIGN KEY (team_id) REFERENCES public.team (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION
);


CREATE TABLE IF NOT EXISTS public.contract
(
    id            SERIAL NOT NULL,
    personal_data_id int   NOT NULL,
    fee           int    NOT NULL,
    date_from      date   NOT NULL,
    date_to        date   NOT NULL,
    PRIMARY KEY (id),

    CONSTRAINT personal_id_foreign
        FOREIGN KEY (personal_data_id) REFERENCES public.personal_data (id)
```

```
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);

CREATE TABLE IF NOT EXISTS public.season
(
   id     SERIAL NOT NULL,
   season_ int    NOT NULL,
   PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.event_types
(
   id   SERIAL  NOT NULL,
   type varchar(20) NOT NULL,
   PRIMARY KEY (id)
);


CREATE TABLE IF NOT EXISTS public.player
(
   id             SERIAL     NOT NULL,
   personal_data_id int       NOT NULL,
   position        VARCHAR(5) NOT NULL,
   season_id       int       NOT NULL,
   PRIMARY KEY (id),

   CONSTRAINT personal_id_foreign
      FOREIGN KEY (personal_data_id) REFERENCES public.personal_data (id)
         ON DELETE NO ACTION
         ON UPDATE NO ACTION,
   CONSTRAINT season_id_foreign
      FOREIGN KEY (season_id) REFERENCES public.season (id)
         ON DELETE NO ACTION
         ON UPDATE NO ACTION
);

CREATE TABLE IF NOT EXISTS public.transfer_cost
(
   id          SERIAL NOT NULL,
   player_id     int    NULL,
   season_start  int    NOT NULL,
   season_finish int    NOT NULL,
   PRIMARY KEY (id),

   CONSTRAINT player_id_foreign
      FOREIGN KEY (player_id) REFERENCES public.player (id)
         ON DELETE NO ACTION
         ON UPDATE NO ACTION
);

CREATE TABLE IF NOT EXISTS public.match
(
   id      SERIAL     NOT NULL,
   team1_id  int       NOT NULL,
```

```
    team2_id  int         NOT NULL,
    result    VARCHAR(10) NOT NULL,
    season_id int         NOT NULL,
    date      date        NOT NULL,
    PRIMARY KEY (id),

    CONSTRAINT team1_id_foreign
        FOREIGN KEY (team1_id) REFERENCES public.team (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION,
    CONSTRAINT team2_id_foreign
        FOREIGN KEY (team2_id) REFERENCES public.team (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION,
    CONSTRAINT season_id_foreign
        FOREIGN KEY (season_id) REFERENCES public.season (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION
);

CREATE TYPE status AS ENUM ('main', 'sub');

CREATE TABLE IF NOT EXISTS public.person_status
(
    id              SERIAL NOT NULL,
    personal_data_id int    NOT NULL,
    match_id        int    NOT NULL,
    match_status    status NOT NULL,
    PRIMARY KEY (id),

    CONSTRAINT personal_id_foreign
        FOREIGN KEY (personal_data_id) REFERENCES public.personal_data (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION
);

CREATE TYPE event_type AS ENUM ('goal', 'assist', 'replaced');

CREATE TABLE IF NOT EXISTS public.match_info
(
    id              SERIAL     NOT NULL,
    match_id        int        NULL,
    personal_data_id int       NOT NULL,
    minute          int        NOT NULL,
    event           event_type NOT NULL,
    PRIMARY KEY (id),

    CONSTRAINT match_id_foreign
        FOREIGN KEY (match_id) REFERENCES public.match (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION,

    CONSTRAINT person_status_foreign
        FOREIGN KEY (personal_data_id) REFERENCES public.personal_data (id)
            ON DELETE NO ACTION
```

```
        ON UPDATE NO ACTION
);

CREATE TABLE IF NOT EXISTS public.team_position
(
    id         SERIAL NOT NULL,
    league_id  int    NOT NULL,
    season_id  int    NOT NULL,
    team_id    int    NOT NULL,
    week       int    NOT NULL,
    w_position int    NOT NULL,
    points     int    NOT NULL,
    PRIMARY KEY (id),

    CONSTRAINT league_id_foreign
        FOREIGN KEY (league_id) REFERENCES public.league (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION,

    CONSTRAINT season_id_foreign
        FOREIGN KEY (season_id) REFERENCES public.season (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION,

    CONSTRAINT team_id_foreign
        FOREIGN KEY (team_id) REFERENCES public.team (id)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION
);
```

```
import numpy
import psycopg2.extras as pgext
import random
import psycopg2

connection = psycopg2.connect(dbname='football_league', user='postgres', password='ligaliga',
host='127.0.0.1')
cursor = connection.cursor()

countries = open("countries.txt", "r").readlines()
leagues = open("countries.txt", "r").readlines()
teams = open("teams.txt", "r").readlines()
first_names = open("first_name.txt", "r").readlines()
last_names = open("last_name.txt", "r").readlines()
tables = open("tables.txt", "r").readlines()
position_type = open("positions.txt", "r").readlines()

rint = random.randint


def clear_database():
    with connection.cursor() as cursor:
        for table in tables:
            cursor.execute("DELETE FROM {0} ".format(table))
            cursor.execute('ALTER SEQUENCE league_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE team_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE personal_data_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE contract_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE transfer_cost_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE player_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE match_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE season_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE match_info_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE team_position_id_seq RESTART')
            cursor.execute('ALTER SEQUENCE person_status_id_seq RESTART')
    connection.commit()


def insert_in_table(table_name, values):
    if not values:
        return
    fields = ', '.join(values[0].keys())
    template = '(' + ', '.join('%({})s'.format(field) for field in values[0].keys()) + ')'
    pgsql = "INSERT INTO {} ({}) VALUES %s".format(table_name, fields)
    with connection.cursor() as cursor:
        pgext.execute_values(cursor, pgsql, values, template)
    connection.commit()


def generate_leagues(size):
    output = []
    i = 0
    while i != size:
```

```
        cur_league = random.choice(leagues).replace("\n", " ").replace(' ', '')
        query = "SELECT COUNT(league_) FROM league WHERE league_ LIKE " + "'" + cur_league + "'"
        cursor.execute(query)
        duplicate = cursor.fetchall()[0]
        if duplicate[0] > 0:
            print("Old {}".format(cur_league))
        else:
            output.append(
                {'league_': cur_league,
                 }
            )
            insert_in_table('league', output)
            output.clear()
            i = i + 1
    return 0


def generate_teams(size):
    query = "SELECT id FROM league"
    cursor.execute(query)
    id_league = cursor.fetchall()
    output = []
    arr_team = []
    for j in range(len(id_league)):
        query = "SELECT COUNT(team) FROM team WHERE league_id = " + str(id_league[j][0])
        cursor.execute(query)
        iftrue = cursor.fetchall()[0]
        arr_team.clear()
        i = 0
        if iftrue[0] == 0:
            while i != size:
                cur_team = random.choice(teams).replace("\n", " ").replace(' ', '')
                if cur_team not in arr_team:
                    output.append(
                        {
                            'team_': cur_team,
                            'league_id': id_league[j][0],
                        }
                    )
                    arr_team.append(cur_team)
                    i = i + 1
    return output


def generate_personal_datas(size):
    output = []
    query = "SELECT id FROM team"
    cursor.execute(query)
    id_team = cursor.fetchall()
    for j in range(len(id_team)):
        query = "SELECT COUNT(id) FROM personal_data WHERE team_id = " + str(id_team[j][0])
        cursor.execute(query)
        iftrue = cursor.fetchall()[0]
        if iftrue[0] == 0:
            for i in range(size):
```

```
            new_name = random.choice(first_names).replace("\n", " ").replace(' ', '') + " " \
                    + random.choice(last_names).replace("\n", " ").replace(' ', '')
            output.append(
                {
                    'name': new_name,
                    'team_id': id_team[j][0],
                    'birthday': str(1984 + rint(0, 11)) + '-' + str(rint(1, 12)) + '-' + str(rint(1, 27)),
                    'nationality': random.choice(countries).replace("\n", " ").replace(' ', ''),
                    'weight': rint(50, 90),
                    'height': rint(161, 198),
                }
            )
    return output


def generate_contracts(people_size, teams_size, leagues_size):
    output = []
    query = "SELECT id FROM personal_data"
    cursor.execute(query)
    id_personal = cursor.fetchall()
    for i in range(people_size * teams_size * leagues_size):
        query = "SELECT COUNT(id) FROM contract WHERE personal_data_id = " + str(id_personal[i][0])
        cursor.execute(query)
        iftrue = cursor.fetchall()[0]
        if iftrue[0] == 0:
            output.append(
                {
                    'personal_data_id': id_personal[i][0],
                    'fee': rint(10000, 450000),
                    'date_from': str(rint(2016, 2020)) + '-' + str(rint(1, 12)) + '-' + str(rint(1, 27)),
                    'date_to': str(rint(2021, 2026)) + '-' + str(rint(1, 12)) + '-' + str(rint(1, 27)),
                }
            )
    return output


def generate_seasons(seasons_size):
    output = []
    i = 0
    while i != seasons_size:
        cur_season = int(input("Type season: "))
        query = "SELECT COUNT(season_) FROM season WHERE season_ = " + str(cur_season)
        cursor.execute(query)
        duplicate = cursor.fetchall()[0]
        if duplicate[0] > 0:
            print("Old {}".format(cur_season))
        else:
            output.append(
                {'season_': cur_season,
                 }
            )
            insert_in_table('season', output)
            output.clear()
            i = i + 1
    return output
```

```
new_players = 0


def generate_players():
    global new_players
    query = "SELECT id FROM season"
    cursor.execute(query)
    id_season = cursor.fetchall()
    query = "SELECT id FROM personal_data"
    cursor.execute(query)
    id_personal = cursor.fetchall()
    output = []
    for j in range(len(id_season)):
        for i in range(len(id_personal)):
            query = "SELECT COUNT(position) FROM player WHERE personal_data_id = " + str(
                i + 1) + " and season_id = " + str(j + 1)
            cursor.execute(query)
            iftrue = cursor.fetchall()[0]
            if iftrue[0] == 0:
                new_players = new_players + 1
                output.append(
                    {
                        'personal_data_id': id_personal[i][0],
                        'position': random.choice(position_type),
                        'season_id': id_season[j][0],
                    }
                )
    return output


def generate_transfer_costs():
    output = []
    query = "SELECT id FROM player"
    cursor.execute(query)
    id_player = cursor.fetchall()
    for i in range(len(id_player)):
        query = "SELECT COUNT(id) FROM transfer_cost WHERE player_id = " + str(id_player[i][0])
        cursor.execute(query)
        iftrue = cursor.fetchall()[0]
        if iftrue[0] == 0:
            output.append(
                {
                    'player_id': id_player[i][0],
                    'season_start': int(round(rint(1000000, 150000000))),
                    'season_finish': int(round(rint(1000000, 150000000))),
                }
            )
    return output


match_results = []
match_teams = []
```

```
def rotate(l, n):
    return l[-n:] + l[:-n]


def generate_matches(teams_size):
    global old_matches
    output = []
    output2 = []
    cursor.execute("SELECT id FROM team")
    teams_id = cursor.fetchall()
    cur_id_count = 0
    id_count = 0
    query = "SELECT COUNT(id) FROM match"
    cursor.execute(query)
    old_matches = cursor.fetchall()[0]
    query = "SELECT COUNT(league_) FROM league"
    cursor.execute(query)
    cnt_league = cursor.fetchall()[0]
    query = "SELECT id FROM season"
    cursor.execute(query)
    id_season = cursor.fetchall()
    for m in range(len(id_season)):
        query = "SELECT season_ FROM season WHERE id = {}".format(id_season[m][0])
        cursor.execute(query)
        year = cursor.fetchall()[0]
        for j in range(cnt_league[0]):
            query = "SELECT COUNT(id) FROM match WHERE team1_id = " + str(
                teams_size * j + 1) + "and season_id = " + str(id_season[m][0])
            cursor.execute(query)
            iftrue = cursor.fetchall()[0]
            week = 0

            if iftrue[0] == 0:
                month = 8
                day = 1
                cur_id_count = cur_id_count + 1
                cur_id = teams_id[teams_size * j:(j + 1) * teams_size]
                points = [0] * 20
                for rs in range(teams_size):
                    points[rs] = 0
                while id_count != (cur_id_count) * (teams_size - 1) * int(round((teams_size / 2))):
                    step = 0
                    week += 1
                    flag = 0
                    day = day + 7
                    if day > 28:
                        day = 1
                        month = month + 1
                    while flag == 0:

                        id_count = id_count + 1
                        if id_count % int(round(teams_size / 2)) == 0:
                            flag = 1
```

```
            first = rint(0, 4)
            second = rint(0, 4)
            if first > second:
                points[step] = points[step] + 3
            elif first < second:
                points[teams_size - 1 - step] = points[teams_size - 1 - step] + 3
            else:
                points[step] = points[step] + 1
                points[teams_size - 1 - step] = points[teams_size - 1 - step] + 1

            cur_result = str(first) + '-' + str(second)
            match_results.append(cur_result)
            match_teams.append(str(cur_id[step]).replace("(", "").replace(')', '').replace(',', '') + "-" +
                               str(cur_id[teams_size - 1 - step]).replace("(", "").replace(')', '').replace(
                                   ',',
                                   ''))
            output.append(
                {
                    'team1_id': cur_id[step],
                    'team2_id': cur_id[teams_size - 1 - step],
                    'result': cur_result,
                    'season_id': id_season[m][0],
                    'date': str(year[0]) + '-' + str(month) + '-' + str(day),
                }
            )
            step = step + 1

        o = 0
        to_sort = numpy.zeros((20, 3))
        for f in range(20):
            to_sort[o][0] = o + 1
            to_sort[o][1] = int(str(cur_id[f]).replace("(", "").replace(')', '').replace(',', ''))
            to_sort[o][2] = points[f]
            o += 1

        for ff in range(19):
            kek = ff
            for aa in range(ff + 1, 20):
                if (to_sort[aa][2] < to_sort[kek][2]):
                    kek = aa
            to_sort[ff][1], to_sort[kek][1], to_sort[ff][2], to_sort[kek][2] = to_sort[kek][1], to_sort[ff][
                1], to_sort[kek][2], to_sort[ff][2]

        for qq in range(20):
            output2.append(
                {
                    'league_id': j + 1,
                    'season_id': id_season[m][0],
                    'team_id': to_sort[qq][1],
                    'week': week,
                    'w_position': 21 - to_sort[qq][0],
                    'points': to_sort[qq][2],
                }
            )
        insert_in_table('team_position', output2)
```

```
            output2.clear()

            cur_id = rotate(cur_id[1:teams_size], 1)
            cur_id.insert(0, teams_id[teams_size * j])
        month = 1
        day = 1
        cur_id_count = cur_id_count + 1
        while id_count != (cur_id_count) * (teams_size - 1) * int(round((teams_size / 2))):
            week += 1
            step = 0
            flag = 0
            day = day + 7
            if day > 28:
                day = 1
                month = month + 1
            while flag == 0:
                id_count = id_count + 1
                if id_count % int(round(teams_size / 2)) == 0:
                    flag = 1

                first = rint(0, 4)
                second = rint(0, 4)
                if first > second:
                    points[teams_size - 1 - step] = points[teams_size - 1 - step] + 3
                elif first < second:
                    points[step] = points[step] + 3
                else:
                    points[step] = points[step] + 1
                    points[teams_size - 1 - step] = points[teams_size - 1 - step] + 1

                cur_result = str(first) + '-' + str(second)
                match_results.append(cur_result)
                match_teams.append(
                    str(cur_id[teams_size - 1 - step]).replace("(", "").replace(')', '').replace(',', '') +
                    "-" + str(cur_id[step]).replace("(", "").replace(')', '').replace(',', ''))
                output.append(
                    {
                        'team1_id': cur_id[teams_size - 1 - step],
                        'team2_id': cur_id[step],
                        'result': cur_result,
                        'season_id': id_season[m][0],
                        'date': str(year[0] + 1) + '-' + str(month) + '-' + str(day),
                    }
                )
                step = step + 1

            o = 0
            to_sort = numpy.zeros((20, 3))
            for f in range(20):
                to_sort[o][0] = o + 1
                to_sort[o][1] = int(str(cur_id[f]).replace("(", "").replace(')', '').replace(',', ''))
                to_sort[o][2] = points[f]
                o += 1

            for ff in range(19):
```

```
                kek = ff
                for aa in range(ff + 1, 20):
                    if (to_sort[aa][2] < to_sort[kek][2]):
                        kek = aa
                to_sort[ff][1], to_sort[kek][1], to_sort[ff][2], to_sort[kek][2] = to_sort[kek][1], to_sort[ff][
                    1], to_sort[kek][2], to_sort[ff][2]

            for qq in range(20):
                output2.append(
                    {
                        'league_id': j + 1,
                        'season_id': id_season[m][0],
                        'team_id': to_sort[qq][1],
                        'week': week,
                        'w_position': 21 - to_sort[qq][0],
                        'points': to_sort[qq][2],
                    }
                )
            insert_in_table('team_position', output2)
            output2.clear()

            cur_id = rotate(cur_id[1:teams_size], 1)
            cur_id.insert(0, teams_id[teams_size * j])
    return output


def generate_match_infos():
    output = []
    output3 = []
    i = old_matches[0]
    query = "SELECT id FROM match"
    cursor.execute(query)
    id_match = cursor.fetchall()
    res_count = 0
    while i != len(id_match):
        t1_g, t2_g = match_results[res_count].split("-")
        t1_, t2_ = match_teams[res_count].split("-")
        i = i + 1

        query = "SELECT id FROM personal_data WHERE team_id = {}".format(t1_)
        cursor.execute(query)
        team_persons = cursor.fetchall()
        main1 = 0
        mains = []
        subs = []
        sub1 = 0
        while main1 != 11:
            curr = random.choice(team_persons)
            if curr not in mains:
                mains.append(curr)
                main1 += 1
                output3.append(
                    {
                        'personal_data_id': curr,
                        'match_id': id_match[i - 1][0],
```

```
                'match_status': 'main',
            }
        )
        insert_in_table('person_status', output3)
        output3.clear()
    while sub1 != 6:
        curr = random.choice(team_persons)
        if curr not in mains and curr not in subs:
            subs.append(curr)
            sub1 += 1
            output3.append(
                {
                    'personal_data_id': curr,
                    'match_id': id_match[i - 1][0],
                    'match_status': 'sub',
                }
            )
            insert_in_table('person_status', output3)
            output3.clear()

    query = "SELECT id FROM personal_data WHERE team_id = {}".format(t2_)
    cursor.execute(query)
    team_persons = cursor.fetchall()
    main1 = 0
    mains = []
    subs = []
    sub1 = 0
    while main1 != 11:
        curr = random.choice(team_persons)
        if curr not in mains:
            mains.append(curr)
            main1 += 1
            output3.append(
                {
                    'personal_data_id': curr,
                    'match_id': id_match[i - 1][0],
                    'match_status': 'main',
                }
            )
            insert_in_table('person_status', output3)
            output3.clear()
    while sub1 != 6:
        curr = random.choice(team_persons)
        if curr not in mains and curr not in subs:
            subs.append(curr)
            sub1 += 1
            output3.append(
                {
                    'personal_data_id': curr,
                    'match_id': id_match[i - 1][0],
                    'match_status': 'sub',
                }
            )
            insert_in_table('person_status', output3)
            output3.clear()
```

```
    cursor.execute("SELECT id FROM personal_data WHERE team_id ={0}".format(t1_))
    players1 = cursor.fetchall()
    cursor.execute("SELECT id FROM personal_data WHERE team_id ={0}".format(t2_))
    players2 = cursor.fetchall()
    cursor.execute("SELECT personal_data_id FROM person_status WHERE match_status = " + "'" + "main" +
"'")
    main_a = cursor.fetchall()

    main_p = []
    for nm in range(20):
        if players1[nm] in main_a:
            main_p.append(players1[nm])

    sub_times = rint(0, 3)
    z = 0
    subbed = []
    while z != sub_times:
        minute = str(rint(1, 96))
        flag = True
        while (flag):
            man = random.choice(main_p)
            if man not in subbed:
                subbed.append(man)
                output.append(
                    {
                        'match_id': id_match[i - 1][0],
                        'personal_data_id': man,
                        'minute': minute,
                        'event': 'replaced',
                    }
                )
                flag = False
                z = z + 1

    for j in range(int(t1_g)):
        minute = str(rint(1, 96))
        man = random.choice(players1)
        output.append(
            {
                'match_id': id_match[i - 1][0],
                'personal_data_id': man,
                'minute': minute,
                'event': 'goal',
            }
        )
        flag = True
        while (flag):
            temp = random.choice(players1)
            if (man != temp):
                flag = False
                man = temp
        output.append(
            {
                'match_id': id_match[i - 1][0],
```

```
            'personal_data_id': man,
            'minute': minute,
            'event': 'assist',
        }
    )

main_p = []
for sm in range(20):
    if players2[sm] in main_a:
        main_p.append(players2[sm])

sub_times = rint(0, 3)
z = 0
subbed = []
while z != sub_times:
    minute = str(rint(1, 96))
    flag = True
    while (flag):
        man = random.choice(main_p)
        if man not in subbed:
            subbed.append(man)
            output.append(
                {
                    'match_id': id_match[i - 1][0],
                    'personal_data_id': man,
                    'minute': minute,
                    'event': 'replaced',
                }
            )
            flag = False
            z = z + 1

for j in range(int(t2_g)):
    minute = str(rint(1, 96))
    man = random.choice(players2)
    output.append(
        {
            'match_id': id_match[i - 1][0],
            'personal_data_id': man,
            'minute': minute,
            'event': 'goal',
        }
    )
    flag = True
    while (flag):
        temp = random.choice(players2)
        if (man != temp):
            man = temp
            flag = False
    output.append(
        {
            'match_id': id_match[i - 1][0],
            'personal_data_id': man,
            'minute': minute,
            'event': 'assist',
```

```
            }
        )
    res_count = res_count + 1
  return output


if __name__ == '__main__':
  clear = int(input("clear the table?:"))
  if clear:
    clear_database()

  seasons_size = 0
  leagues_size = 0

  if int(input("generate leagues?[1/0]")):
    leagues_size = int(input("How many?:"))
    league = generate_leagues(leagues_size)
    if int(input("generate teams?[1/0]")):
      teams_size = 20
      team = generate_teams(teams_size)
      insert_in_table('team', team)
      if int(input("generate personal datas?[1/0]")):
        people_size = int(input("How many (in each team)?:"))
        personal_data = generate_personal_datas(people_size)
        insert_in_table('personal_data', personal_data)
        if int(input("generate contracts?[1/0]")):
          contracts = generate_contracts(people_size, teams_size, leagues_size)
          insert_in_table('contract', contracts)

  if int(input("generate seasons?[1/0]")):
    seasons_size = int(input("How many?:"))
    seasons = generate_seasons(seasons_size)
    insert_in_table('season', seasons)
    if int(input("generate players?[1/0]")):
      players = generate_players()
      insert_in_table('player', players)
      if int(input("generate transfer costs?[1/0]")):
        transfer_cost = generate_transfer_costs()
        insert_in_table('transfer_cost', transfer_cost)

  if int(input("generate matches?[1/0]")):
    matches = generate_matches(20)
    insert_in_table('match', matches)
    if int(input("generate match infos?[1/0]")):
      match_infos = generate_match_infos()
      insert_in_table('match_info', match_infos)
```

```
import threading
import time
import matplotlib.pyplot as plotpy
import random
import psycopg2

connection = psycopg2.connect(dbname='football_league', user='postgres', password='ligaliga',
host='127.0.0.1')
cursor = connection.cursor()
connection.autocommit = True

first_names = open("first_name.txt", "r").readlines()
last_names = open("last_name.txt", "r").readlines()
position_type = open("positions.txt", "r").readlines()
countries = open("countries.txt", "r").readlines()
match_results = open("results.txt", "r").readlines()
queries = open("queries.txt", "r").readlines()
rint = random.randint


def before_optimization(thread_cursor, query):
    if query == queries[0]:
        thread_cursor.execute("EXPLAIN ANALYZE " + queries[0], {"team_id": rint(1, 140), "season_id": rint(1,
7)})
    elif query == queries[1]:
        thread_cursor.execute("EXPLAIN ANALYZE " + queries[1], {"season_id": rint(1, 7), "result":
random.choice(match_results)})
    elif query == queries[2]:
        thread_cursor.execute("EXPLAIN ANALYZE " + queries[2], {"team_id": rint(1, 140)})
    elif query == queries[3]:
        thread_cursor.execute("EXPLAIN ANALYZE " + queries[3], {"person": rint(1, 2800)})
    elif query == queries[4]:
        thread_cursor.execute("EXPLAIN ANALYZE " + queries[4], {"team_id": rint(1, 140)})
    elif query == queries[5]:
        thread_cursor.execute("EXPLAIN ANALYZE " + queries[5], {"cost": rint(1000000, 150000000)})
    result = thread_cursor.fetchall()
    # print(result[-1][0])
    return float(result[-1][0].split(" ")[2])

def optima1(thread_cursor):
    thread_cursor.execute("explain analyze select personal_data.id, name, count(event) as cnt from
personal_data JOIN "
                          "match_info ON (personal_data.id = personal_data_id and minute > 45 and event = 'goal') group
by personal_data.id, name "
                          "order by count(event) DESC LIMIT 1")
    result = thread_cursor.fetchall()
    # print(result[-1][0])
    return float(result[-1][0].split(" ")[2])


def optima2(thread_cursor):
    thread_cursor.execute("EXPLAIN ANALYZE select id, name, top10.second_half_goals from personal_data
JOIN (select max_sum.max_goals as second_half_goals, max_sum.person as name_id "
```

```
                    "from (select goals_sum.personal_data_id as person, sum(goals_sum.sh_goals) as max_goals "
                    "from (select personal_data_id, count(event) as sh_goals from match_info where minute > 45
and event = 'goal' "
                    "group by personal_data_id, minute, event) goals_sum group by goals_sum.personal_data_id)
max_sum "
                    "order by max_sum.max_goals DESC LIMIT 10) as top10 ON id = top10.name_id order by
top10.second_half_goals DESC")
    result = thread_cursor.fetchall()
    # print(result[-1][0])
    return float(result[-1][0].split(" ")[2])


query1 = queries[0]
query2 = queries[1]
query3 = queries[2]
query4 = queries[3]
query5 = queries[4]
query6 = queries[5]
optimized_queries = (query1, query2, query3, query4, query5, query6)


def drop_indexes():
    cursor.execute("DROP INDEX IF EXISTS team1_idx")
    cursor.execute("DROP INDEX IF EXISTS team2_idx")
    cursor.execute("DROP INDEX IF EXISTS season_idx")
    cursor.execute("DROP INDEX IF EXISTS result_idx")
    cursor.execute("DROP INDEX IF EXISTS person_team_idx")
    cursor.execute("DROP INDEX IF EXISTS person_idx")
    cursor.execute("DROP INDEX IF EXISTS team_pos_idx")
    cursor.execute("DROP INDEX IF EXISTS cost_finish_idx")
    cursor.execute("DROP INDEX IF EXISTS cost_start_idx")

    cursor.execute("DROP INDEX IF EXISTS pirs")
    cursor.execute("DROP INDEX IF EXISTS iven")
    cursor.execute("DROP INDEX IF EXISTS minu")


def create_indexes():
    cursor.execute("CREATE INDEX team1_idx ON match(team1_id)")
    cursor.execute("CREATE INDEX team2_idx ON match(team2_id)")
    cursor.execute("CREATE INDEX season_idx ON match(season_id)")
    cursor.execute("CREATE INDEX result_idx ON match(result)")
    cursor.execute("CREATE INDEX person_team_idx ON personal_data(team_id)")
    cursor.execute("CREATE INDEX person_idx ON match_info(personal_data_id)")
    cursor.execute("CREATE INDEX team_pos_idx ON team_position(team_id)")
    cursor.execute("CREATE INDEX cost_finish_idx ON transfer_cost(season_finish)")
    cursor.execute("CREATE INDEX cost_start_idx ON transfer_cost(season_finish)")

    cursor.execute("CREATE INDEX pirs ON personal_data(name)")
    cursor.execute("CREATE INDEX iven ON match_info(event)")
    cursor.execute("CREATE INDEX minu ON match_info(minute)")
```

```
def queries_optimization(thread_cursor):
    thread_cursor.execute(
        "PREPARE query1 (int, int) AS SELECT team.team_ as team, match.result, season.season_ as season, date "
        "FROM match JOIN team on team.id = $1 and (team2_id = team.id or team1_id = team.id ) "
        "JOIN season on season.id = $2 and match.season_id = season.id")
    thread_cursor.execute("PREPARE query2  (int, varchar) AS SELECT count(result) amount, season.season_
FROM match "
                          "JOIN season on season.id = $1 "
                          "WHERE match.result LIKE $2 and season_id = season.id group by season.season_")
    thread_cursor.execute("PREPARE query3  (int) AS SELECT name, birthday, nationality, team.team_ as team
"
                          "FROM personal_data JOIN team ON team_id = $1 and team.id = team_id")
    thread_cursor.execute("PREPARE query4 (int) AS SELECT personal_data.name, match_id, minute, event "
                          "FROM match_info JOIN personal_data "
                          "on personal_data.id = $1 and match_info.personal_data_id = personal_data.id")
    thread_cursor.execute("PREPARE query5  (int) AS SELECT season.season_, week, team.team_ as team,
w_position, points"
                          " FROM team_position JOIN team ON team_position.team_id = $1 and "
                          "team.id = team_position.team_id and week = 38 JOIN season ON season_id = season.id")
    thread_cursor.execute("PREPARE query6 (int) AS SELECT player_id, season_finish, season_start FROM
transfer_cost "
                          "WHERE season_finish > $1 and season_start > season_finish")


def after_optimization(thread_cursor, query):
    if query == query1:
        thread_cursor.execute("EXPLAIN ANALYZE EXECUTE query1 (%s,%s)", [rint(1, 140), rint(1, 7)])
    elif query == query2:
        thread_cursor.execute("EXPLAIN ANALYZE EXECUTE query2 (%s,%s)",
                              [rint(1, 7), random.choice(match_results).replace("\n", " ").replace('  ', '')])
    elif query == query3:
        thread_cursor.execute("EXPLAIN ANALYZE EXECUTE query3 (%s)", [rint(1, 140)])
    if query == query4:
        thread_cursor.execute("EXPLAIN ANALYZE EXECUTE query4 (%s)", [rint(1, 2800)])
    elif query == query5:
        thread_cursor.execute("EXPLAIN ANALYZE EXECUTE query5 (%s)", [rint(1, 140)])
    elif query == query6:
        thread_cursor.execute("EXPLAIN ANALYZE EXECUTE query6 (%s)", [rint(1000000, 150000000)])
    result = thread_cursor.fetchall()
    return float(result[-1][0].split(" ")[2]) + float(result[-2][0].split(" ")[2])


avg_respond_per_thread = []


def threads_stress(t_number, q_number):
    avg_respond_per_thread.clear()
    for i in range(t_number):
        current_thread = ThreadsWithQueries(i)
        current_thread.start()
    while threading.activeCount() > 1: time.sleep(1)
    plot_x = [g for g in range(100, q_number, 100)]
    plot_y = []
    for j in range(len(avg_respond_per_thread[0])):
        respond_time = 0
```

```
        for f in range(t_number): respond_time = avg_respond_per_thread[f][j] + respond_time
        plot_y.append(respond_time / t_number)
    plotpy.plot(plot_x, plot_y)
    plotpy.xlabel('Запросов в единицу времени')
    plotpy.ylabel('Затраты на один запрос, мс')
    if optimized:
        plotpy.title("После оптимизации. Потоков: {}".format(t_number))
    else:
        plotpy.title("До оптимизации. Потоков: {}".format(t_number))
    plotpy.show()


class ThreadsWithQueries(threading.Thread):
    def __init__(self, current_thread):
        self.current_thread = current_thread
        threading.Thread.__init__(self)
        self.conn = psycopg2.connect(dbname='football_league', user='postgres', password='ligaliga',
                                     host='127.0.0.1')
        self.cur = self.conn.cursor()
        if optimized: queries_optimization(self.cur)

    def run(self):
        if len(avg_respond_per_thread) <= self.current_thread:
            avg_respond_per_thread.append([])
        for i in range(100, query_number, 100):
            respond_time = []
            for j in range(0, i):
                if not optimized:
                    # random_query = random.choice(queries)
                    # respond_time.append(before_optimization(self.cur, random_query))
                    respond_time.append(optima2(self.cur))
                else:
                    # random_query = random.choice(optimized_queries)
                    # respond_time.append(after_optimization(self.cur, random_query))
                    respond_time.append(optima2(self.cur))
            avg_respond_per_thread[self.current_thread].append(sum(respond_time) / i)
        self.conn.commit()


if __name__ == "__main__":
    thread_number = 1
    query_number = 500
    drop_indexes()
    optimized = False
    threads_stress(thread_number, query_number)
    create_indexes()
    optimized = True
    threads_stress(thread_number, query_number)
```