

Санкт-Петербургский политехнический университет Петра Великого
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторным работам №1-4

Дисциплина: Базы данных

Выполнил студент гр. 3530901/70203 _____ В.И.Костарев
(подпись)

Преподаватель _____ А.В.Мяснов
(подпись)

Санкт-Петербург
2021

Содержание

1. Лабораторная работа №1	3
1.1 Цель работы.....	3
1.2. Программа работы	3
1.3. Ход работы	3
1.4 Выводы	7
2. Лабораторная работа №2	7
2.1 Цель работы.....	7
2.2. Программа работы	8
2.3. Ход работы	8
2.4 Выводы	9
3. Лабораторная работа №3	9
3.1 Цель работы.....	9
3.2. Программа работы	9
3.3. Ход работы	9
4. Лабораторная работа №4	14
5. Листинги.....	49

1. Лабораторная работа №1

1.1 Цель работы

- Познакомиться с основами проектирования схемы БД, способами организации данных в SQL-БД.
- Познакомиться с языком описания сущностей и ограничений БД SQL-DDL.

1.2. Программа работы

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

1.3. Ход работы

Предметная область: Онлайн-игра

Описание таблиц:

- User_data – таблица зарегистрированных пользователей
 - Id – идентификатор пользователя
 - Email – электронная почта пользователя (до 40 символов)

Password – пароль пользователя (до 16 символов)

Nickname – уникальный ник (до 20 символов)

- Person – таблица созданных персонажей пользователей

Id – идентификатор персонажей

Class_of_person_id – идентификатор типа класса

Health – текущее здоровье персонажа

Experience – текущий опыт персонажа

User_id – идентификатор пользователя, владеющего персонажем
(может быть Null, если персонаж – бот)

Update_date – дата последнего выполненного персонажем действия

Is_enemy – является ли персонаж ботом/прс/ИИ... (Bool) (при этом user_id = Null)

- Class_of_person – таблица существующих в игре классов персонажей

Id – идентификатор класса персонажа

Name – Название класса (до 30 символов)

Description – Описание класса

- Skill – таблица существующих в игре навыков, принадлежащих определённому классу

Id – идентификатор навыка

Name – название навыка (до 30 символов)

Description – описание навыка

Cost – стоимость навыка в очках опыта

Class_of_person_id – идентификатор класса персонажа к которому принадлежит навык

- Person_skill – таблица навыков персонажей

Person_id – идентификатор персонажа

Skill_id – идентификатор навыка

Is_equiped – куплен ли навык у персонажа (Bool) (я знаю что equipped)

- Meetup – таблица сражений персонажей

Person_id – идентификатор персонажа, инициировавшего сражение (нападающего)

Result – результат сражения (enum – win, draw, loose)

Meetup_date – дата и время сражения

Enemy_id – идентификатор персонажа на которого нападают (жертвы)

- Inventory_person – таблица инвентарей персонажей (их может быть несколько)

Id – идентификатор инвентаря

Person_id – идентификатор персонажа

Inventory_size – сколько вещей вмещает в себя инвентарь

- Inventory_person_items – таблица вещей в инвентарях персонажей

Inventory_person_id – идентификатор инвентаря персонажа

Item_id – идентификатор вещи

Add_date – дата и время добавления вещи в инвентарь персонажа

Is_deleted – удалена ли вещь из инвентаря (отобрали/потратили...)

Amount – количество вещей такого же типа (item_id) в инвентаре персонажа

Update_date – дата обновления вещи (добавления вещи такого же типа/удаления вещи такого же типа)

- Item – таблица существующих в игре типов вещей

Id – идентификатор вещи

Name – имя вещи (до 30 символов)

Description – описание вещи

По заданию преподавателя реализованы следующие изменения схемы:

1. Объединить таблицы person и enemy (существовала ранее), inventory_person и inventory_enemy (существовала ранее), inventory_person_items, inventory_enemy_items (существовала ранее).

2. В таблицу person добавить поле is_enemy и сделать user_id возможным Null. При этом необходимо сохранить имеющиеся данные в БД.

Все изменения были оформлены одним отдельным скриптом, указанном в листинге 5.1.

Схема полученной БД до изменения представлена на рис. 1.3.1. После изменения – на рис. 1.3.2.

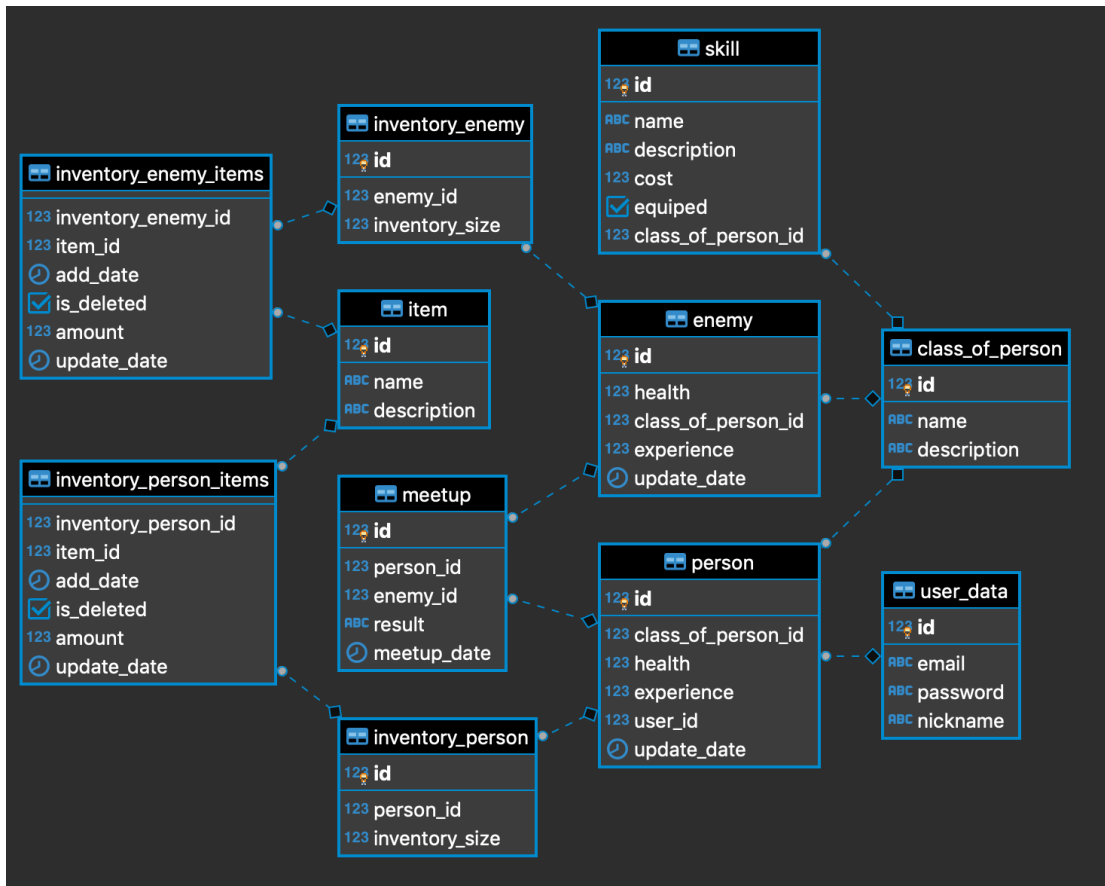


Рис. 1.3.1. Структура базы данных для игры до изменения

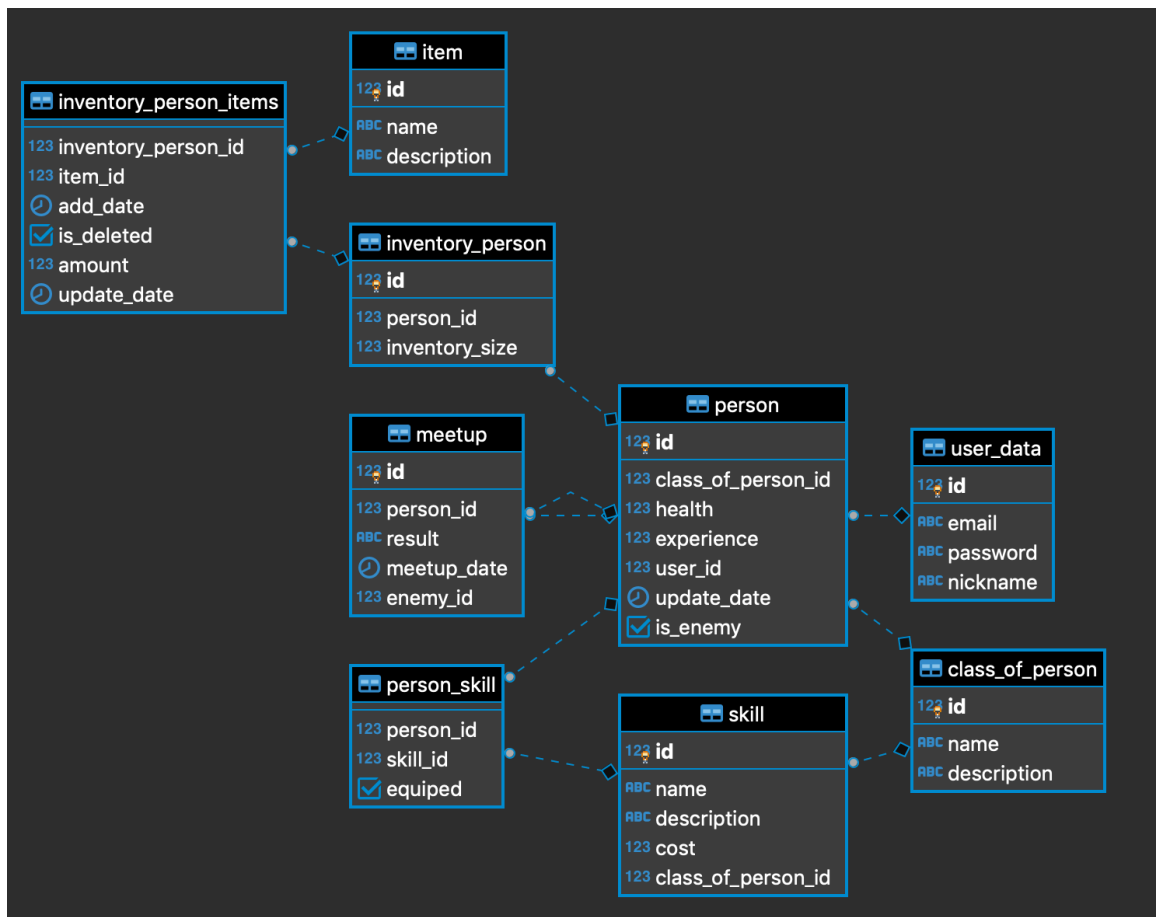


Рис. 1.3.2. Структура базы данных для игры после изменения

1.4 Выводы

В ходе выполнения данной лабораторной работы проведено ознакомление с основами проектирование схемы базы данных. Были изучены основы создания скриптов на языке SQL. С помощью SQL-DDL описаны структуры разрабатываемой схемы БД. Было проведено знакомство с первичными и внешними ключами, на значения были наложены ограничения в виде диапазонов.

2. Лабораторная работа №2

2.1 Цель работы

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

2.2. Программа работы

- Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.
- 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

2.3. Ход работы

Программа была реализована на ЯП Python. Для взаимодействий с базой данных использовалась библиотека `psycopg2`.

Количество генерируемых данных определяется пользователем через аргументы командной строки. Можно указать количество для таблиц пользователей, персонажей, инвентарей, вещей в инвентарях и сражений.

Все email генерируются как случайный набор букв + доменное имя из файла, пароль генерируется, как случайный набор символов. Никнеймы, названия классов, вещей, скиллов выбираются случайно из подготовленных текстовых файлов. Все остальные значения так же генерируются случайным образом, учитывая ограничения, накладываемые базой данных.

Данные генерируются последовательно, то есть сначала генерируются пользователи, классы, скиллы и вещи, так как на них буду ссылаться в первую очередь. Затем генерируются персонажи и их инвентари (по умолчанию 1, если не было задано больше). Далее инвентари заполняются, после чего генерируются встречи персонажей. Вещи добавляются с учётом уже существующих в инвентаре (если у персонажа уже есть вещь такого типа – её поле `amount` увеличивается на 1 с помощью `UPDATE`).

Данные для заполнения каждой таблицы генерируются в циклах, где формируется массив значений, который позже вставляется в `INSERT INTO`.

После завершения выполнения цикла запрос выполняется, то есть на заполнение одной таблицы осуществляется одна транзакция.

Код программы генератора представлен в Листинге 5.2.

2.4 Выводы

В ходе выполнения данной лабораторной работы был реализован в виде программы генератор, который позволяет сформировать набор связанных данных в каждой таблице. Были получены практические навыки взаимодействия с базой данных через собственную программу.

3. Лабораторная работа №3

3.1 Цель работы

- Познакомиться с языком создания запросов управления данными SQL-DML.

3.2. Программа работы

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

3.3. Ход работы

Были выполнены все запросы из списка стандартных запросов. Они приведены в Листинге 5.3. Отчёт по стандартным запросам находится в папке lab3_sql_dml в репозитории проекта.

В ходе написания запросов использовались следующие операторы:

- LIKE – используется в предложении WHERE для поиска заданного шаблона в столбце

- BETWEEN – выбирает значения в заданном диапазоне. Значения могут быть числами, текстом или датами
 - IN – позволяет указать несколько значений в предложении WHERE
 - ORDER BY – позволяет сортировать записи по определенному полю при выборе из базы данных
 - JOIN – используется для объединения строк из двух или более таблиц на основе соответствующего столбца между ними
 - GROUP BY – часто используется с агрегатными функциями для группировки результирующего набора одним или несколькими столбцами
 - SELECT – используется для выбора данных из базы данных. Возвращаемые данные сохраняются в таблице результатов, называемой результирующим набором
 - HAVING – используется в сочетании с оператором GROUP BY, чтобы ограничить группы возвращаемых строк только теми, чье условие TRUE
 - INSERT – используется для добавления записи в таблицу.
 - UPDATE – используется для изменения существующих записей в таблице
 - DELETE – используется для удаления существующих записей в таблице
- И следующие агрегатные функции:
- MIN() – возвращает наименьшее значение выбранного столбца
 - MAX() – возвращает наибольшее значение выбранного столбца
 - COUNT() – возвращает количество входных строк

По заданию преподавателя были реализованы следующие запросы к БД:

1. Вывести игроков, которые за последние полгода каждый месяц увеличивали количество уникальных полученных артефактов

Запрос:

```
WITH inventory_to_person_half_year_table AS (
    SELECT ip.person_id, item_id, MIN(add_date) AS add_date FROM
    inventory_person_items ipi
    INNER JOIN inventory_person ip ON ipi.inventory_person_id = ip.id
    WHERE add_date > now()::date - '6 month'::interval
```

```

GROUP BY ip.person_id, item_id
ORDER BY ip.person_id, item_id
), gain_rare_item_every_month AS (
  SELECT person_id FROM inventory_to_person_half_year_table
  WHERE (add_date >= (now()::date - '1 month'::interval)
        AND add_date < now()::date)
  AND person_id IN (
    SELECT person_id FROM inventory_to_person_half_year_table
    WHERE (add_date >= (now()::date - '2 month'::interval)
          AND add_date < (now()::date - '1 month'::interval))
    AND person_id IN (
      SELECT person_id FROM
inventory_to_person_half_year_table
      WHERE (add_date >= (now()::date - '3 month'::interval)
            AND add_date < (now()::date - '2
month'::interval))
      AND person_id IN (
        SELECT person_id FROM
inventory_to_person_half_year_table
        WHERE (add_date >= (now()::date - '4
month'::interval)
              AND add_date < (now()::date - '3
month'::interval))
        AND person_id IN (
          SELECT person_id FROM
inventory_to_person_half_year_table
          WHERE (add_date >= (now()::date - '5
month'::interval)
                AND add_date < (now()::date -
'4 month'::interval))
          AND person_id IN (
            SELECT person_id FROM
inventory_to_person_half_year_table
            WHERE add_date >=
(now()::date - '6 month'::interval)
              AND add_date < (now()::date
- '5 month'::interval)
          )
        )
      )
    )
  )
)
SELECT * FROM person WHERE id IN (SELECT * FROM
gain_rare_item_every_month)

```

Результат (рис. 3.3.1):

	id	class_of_person_id	health	experience	user_id	update_date	is_enemy
3	120325	10	150	80	60008	2020-09-17 19:07:12.115950	false
4	120527	1	190	95	<null>	2020-10-16 19:07:12.115950	true
5	120591	10	120	70	62727	2020-09-21 19:07:12.115950	false
6	120614	11	140	65	67488	2020-09-08 19:07:12.115950	false
7	120646	1	110	30	65897	2020-09-18 19:07:12.115950	false
8	120700	12	140	75	68006	2020-06-24 19:07:12.115950	false
9	120729	10	170	30	61197	2020-09-09 19:07:12.115950	false
10	120738	2	250	50	66289	2020-11-16 19:07:12.115950	false
11	120778	1	170	35	65986	2020-11-13 19:07:12.115950	false
12	121009	2	180	75	64901	2019-11-03 19:07:12.115950	false
13	121061	11	220	75	67918	2020-11-12 19:07:12.115950	false
14	121100	1	290	65	60077	2020-11-07 19:07:12.115950	false
15	121248	3	240	65	65963	2020-11-21 19:07:12.115950	false
16	121280	3	120	25	65991	2020-08-26 19:07:12.115950	false
17	121370	10	230	55	63036	2020-08-27 19:07:12.115950	false

Рис. 3.3.1. Результат выполнения первого индивидуального запроса

- Ввести 5 лучших игроков по отношению количества побед в схватках к количеству полученных артефактов за последний год

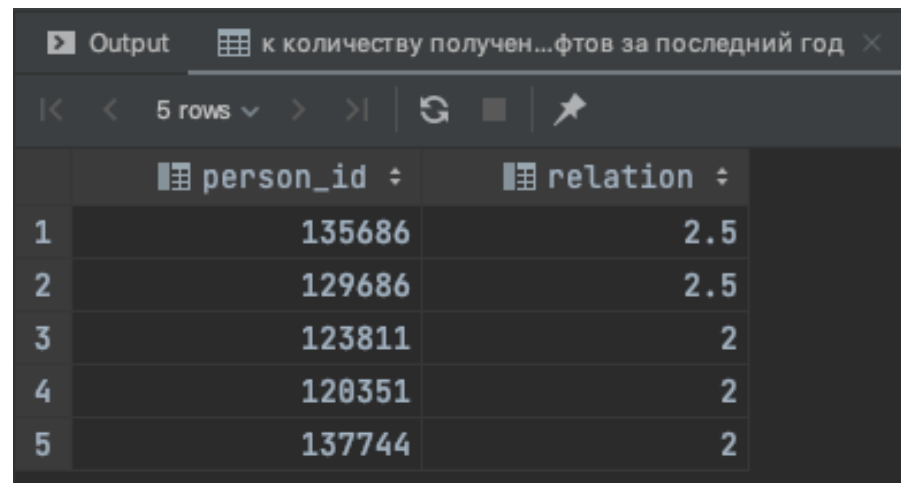
Запрос:

```
WITH person_win_meetup_count AS
(
    SELECT person_id, count(*) AS wins_count FROM
    (
        SELECT person_id FROM meetup
        WHERE meetup_date > now()::date - '1 year'::interval
        AND result = 'win'
        UNION ALL
        SELECT enemy_id FROM meetup
        WHERE meetup_date > now()::date - '1 year'::interval
        AND result = 'lose' AND enemy_id IS NOT NULL
    ) AS winners
    GROUP BY person_id ORDER BY wins_count DESC
), inventory_to_person_half_year_table AS
(
    SELECT ip.person_id, SUM(amount) AS collected_items FROM
    inventory_person_items ipi
    INNER JOIN inventory_person ip ON ipi.inventory_person_id = ip.id
    WHERE add_date > now()::date - '1 year'::interval
    GROUP BY ip.person_id
    ORDER BY collected_items DESC
)

SELECT person_id, (pwmc.wins_count::real/iphyt.collected_items::real)
AS relation FROM person_win_meetup_count pwmc
```

```
INNER JOIN inventory_to_person_half_year_table iphyt USING (person_id)
ORDER BY relation DESC LIMIT 5
```

Результат (рис. 3.3.2):



The screenshot shows a database query result window with a dark theme. The title bar includes 'Output' and a tab labeled 'к количеству получен...фтов за последний год'. Below the title bar is a toolbar with navigation icons and a '5 rows' indicator. The main area displays a table with two columns: 'person_id' and 'relation'. The table contains five rows of data, numbered 1 to 5 in the first column.

	person_id	relation
1	135686	2.5
2	129686	2.5
3	123811	2
4	120351	2
5	137744	2

Рис. 3.3.2. Результат выполнения второго индивидуального запроса

Используемые операторы:

- INNER JOIN – соединение двух таблиц, при котором для каждой строки R1 из T1 в результирующей таблице содержится строка для каждой строки в T2, удовлетворяющей условию соединения с R1.
- WITH – предоставляет способ записывать дополнительные операторы для применения в больших запросах. Эти операторы, которые также называют общими табличными выражениями (Common Table Expressions, CTE), можно представить как определения временных таблиц, существующих только для одного запроса. Дополнительным оператором в предложении WITH может быть SELECT, INSERT, UPDATE или DELETE, а само предложение WITH присоединяется к основному оператору, которым также может быть SELECT, INSERT, UPDATE или DELETE.
- LIMIT – позволяет извлечь определенное количество строк

3.4 Выводы

В ходе выполнения данной лабораторной работы было проведено ознакомление с языком создания запросов и управления данными SQL-DML. Была проведена работа с выборкой данных, их вставкой, удалением и

модификацией. Прделаны все стандартные запросы и выполнены индивидуальные задания

4. Лабораторная работа №4

4.1 Цель работы

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

4.2. Программа работы

- Написание параметризованных типовых запросов пользователей.
- Моделирование нагрузки базы данных.
- Снятие показателей работы сервиса и построение соответствующих графиков.
- Применение возможных оптимизаций запросов и повторное снятие показателей.
- Сравнительный анализ результатов.
- Демонстрация результатов преподавателю.

4.3. Ход работы

Программа была реализована на ЯП Python. Для взаимодействий с базой данных использовалась библиотека `psycopg2`.

Оптимизация проводилась с помощью `PREPARE` и `CREATE INDEX`, а анализ – с помощью `EXPLAIN ANALYSE`.

- `PREPARE` – при выполнении данной команды указанный оператор анализируется и переписывается. Далее при выполнении `EXECUTE` подготовленный оператор планируется и исполняется. Таким образом запрос повторно не разбирается, а также это позволяет выбрать лучший план выполнения.
- `CREATE INDEX` – создает индексы по указанному столбцу. 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. В итоге результат может быть найден быстро.

- 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, 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, 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.

Используемые запросы:

```
-- Запрос 1
-- Вывести информацию обо всех персонажах юзера не активных на
-- протяжении года
-- на которо нападали больше чем напал сам персонаж (параметр –
-- никнейм)
EXPLAIN (ANALYZE, COSTS OFF)
  SELECT DISTINCT p.id, cap.name AS class FROM person p
    INNER JOIN class_of_person cap ON p.class_of_person_id =
cap.id
    INNER JOIN user_data ud ON p.user_id = ud.id
    INNER JOIN meetup m on p.id = m.person_id OR p.id =
m.enemy_id
  WHERE ud.nickname = 'Lamb5Wool'
    AND p.update_date <= now() - '1 year'::interval
  GROUP BY p.id, cap.name, m.person_id, m.enemy_id
  HAVING count(m.enemy_id = p.id OR NULL) != 0
    AND count(m.person_id = p.id OR NULL)/count(m.enemy_id =
```

```
p.id OR NULL) < 1;
```

```
-- Запрос 2
```

```
-- Вывести ники юзеров-лидеров и кол-во требуемых результатов  
(победа/поражение/ничья) в инициированных
```

```
-- сражениях за последний год (параметр – тип результата сражения)
```

```
EXPLAIN (ANALYZE, COSTS OFF) SELECT ud.nickname, count(m.*) AS results  
FROM user_data ud
```

```
    INNER JOIN person p ON ud.id = p.user_id
```

```
    INNER JOIN meetup m ON p.id = m.person_id
```

```
    AND m.result = 'loose'
```

```
    AND m.meetup_date >= now() - '1 year'::interval
```

```
    AND m.meetup_date < now()
```

```
    GROUP BY ud.nickname
```

```
    ORDER BY results DESC;
```

```
-- Запрос 3
```

```
-- Вывести и стакнуть артефакты из всех инвентарей персонажа (параметр  
-- id персонажа)
```

```
EXPLAIN (ANALYZE, COSTS OFF) SELECT i.name, SUM(amount) FROM  
inventory_person_items ipi
```

```
    INNER JOIN item i on i.id = ipi.item_id
```

```
    INNER JOIN inventory_person ip on ipi.inventory_person_id =
```

```
ip.id
```

```
    INNER JOIN person p on ip.person_id = p.id
```

```
    WHERE p.id = 120007
```

```
    GROUP BY i.name;
```

```
-- Запрос 4
```

```
-- Вывести персонажей определённого класса которые прошли всю игру
```

```
-- (вкачали оба скилла, макс жизней, получили редкий айтем) (параметр  
-- имя класса)
```

```
EXPLAIN (ANALYZE, COSTS OFF) SELECT ps.person_id FROM person_skill ps
```

```
    INNER JOIN skill s ON ps.skill_id = s.id
```

```
    INNER JOIN class_of_person cop on s.class_of_person_id = cop.id
```

```
    INNER JOIN person p ON ps.person_id = p.id
```

```
    INNER JOIN inventory_person ip ON p.id = ip.person_id
```

```
    INNER JOIN inventory_person_items ipi on ip.id =
```

```
ipi.inventory_person_id
```

```
    WHERE cop.name = 'Archer' AND health = 300 AND ipi.item_id = 11
```

```
    GROUP BY ps.person_id HAVING count(equiped=true OR NULL)=2;
```

```
-- Запрос 5
```

```
-- Вывести рейтинг игроков (ники) по полученным за определённый  
промежуток времени редким артефактам
```

```
-- для всех персонажей (параметр – даты)
```

```
EXPLAIN (ANALYZE, COSTS OFF) SELECT ud.nickname, sum(ipi.amount) FROM  
inventory_person_items ipi
```

```
    INNER JOIN inventory_person ip on ip.id = ipi.inventory_person_id
```

```
    INNER JOIN person p on p.id = ip.person_id
```

```
    INNER JOIN user_data ud on p.user_id = ud.id
```

```
    WHERE (ipi.update_date < now() - '0 months'::interval
```

```
        AND ipi.update_date >= now() - '12 months'::interval)
```

```
        AND ipi.add_date < now() - '0 months'::interval
```

```
        AND ipi.add_date >= now() - '12 months'::interval
```

```
        AND ipi.item_id = 11
```



```
GROUP BY ud.nickname  
ORDER BY sum(ipi.amount) DESC;
```

Эксперименты были проведены на двух таблицах.

Количество записей в таблицах обычной базы данных:

User_data – 1 000

Person – 5 000

Person_skill – 10 000

Class_of_person – 6

Skill – 12

Meetup – 10 000

Inventory_person – 12 000

Inventory_person_items – 10 000

Item – 15

Количество записей в таблицах большой базы данных:

User_data – 10 000

Person – 20 000

Person_skill – 40 000

Class_of_person – 6

Skill – 12

Meetup – 100 000

Inventory_person – 22 000

Inventory_person_items – 150 000

Item – 15

Количество потоков (threads_count), запускаемых одновременно для нагрузки БД и время исполнения запросов (unit_of_time) в секундах задаются в программе в качестве значений переменных. В начале работы программа загружает базу данных, запуская заданное в threads_count количество потоков, которые должны успеть выполнить все запросы за единицу времени, заданную в unit_of_time. Такая операция проводится сначала для неоптимизированных запросов, затем с использованием индексов и в конце используя индексы и

Prepare одновременно. При успешном выполнении потоком заданного количества запросов – рассчитывается среднее время выполнения этого количества запросов. При неудаче поток останавливается на выполненном количестве запросов. При построении графика выбирается поток, выполнивший наибольшее количество запросов.

Также был построен график зависимости времени ответа на запрос от количества потоков при фиксированном количестве запросов.

В качестве индексов были выбраны Foreign Keys и некоторые поля типа varchar и timestamp.

Код программы представлен в листинге 5.4.

В результате проведения экспериментов с обычной базой данных были построены графики для 1, 50 и 97 (макс подключений для бд) потоков. Единица времени равна 5 секундам. Шаг изменения количества запросов равен 10. Графики представлены на рисунках 4.3.1–4.3.5.

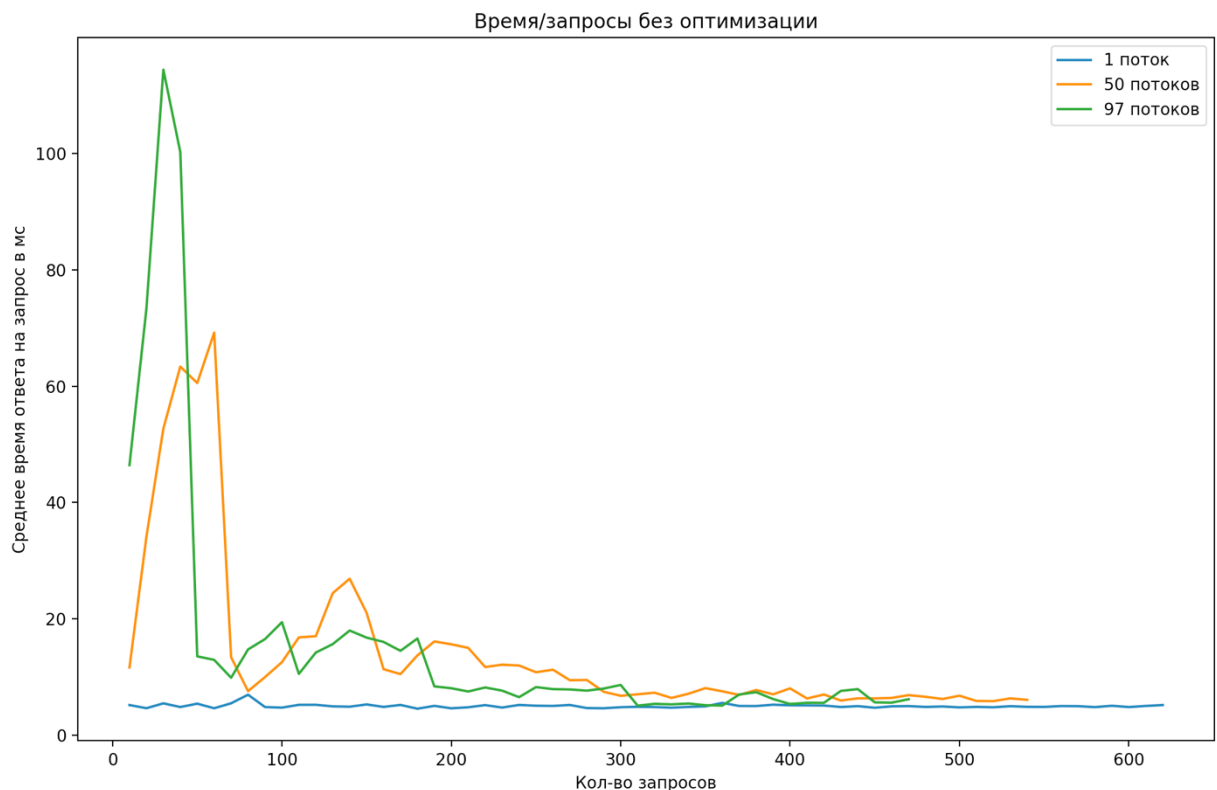


Рис. 4.3.1. График выполнения запросов без оптимизации для 1, 50 и 97 потоков на обычной бд

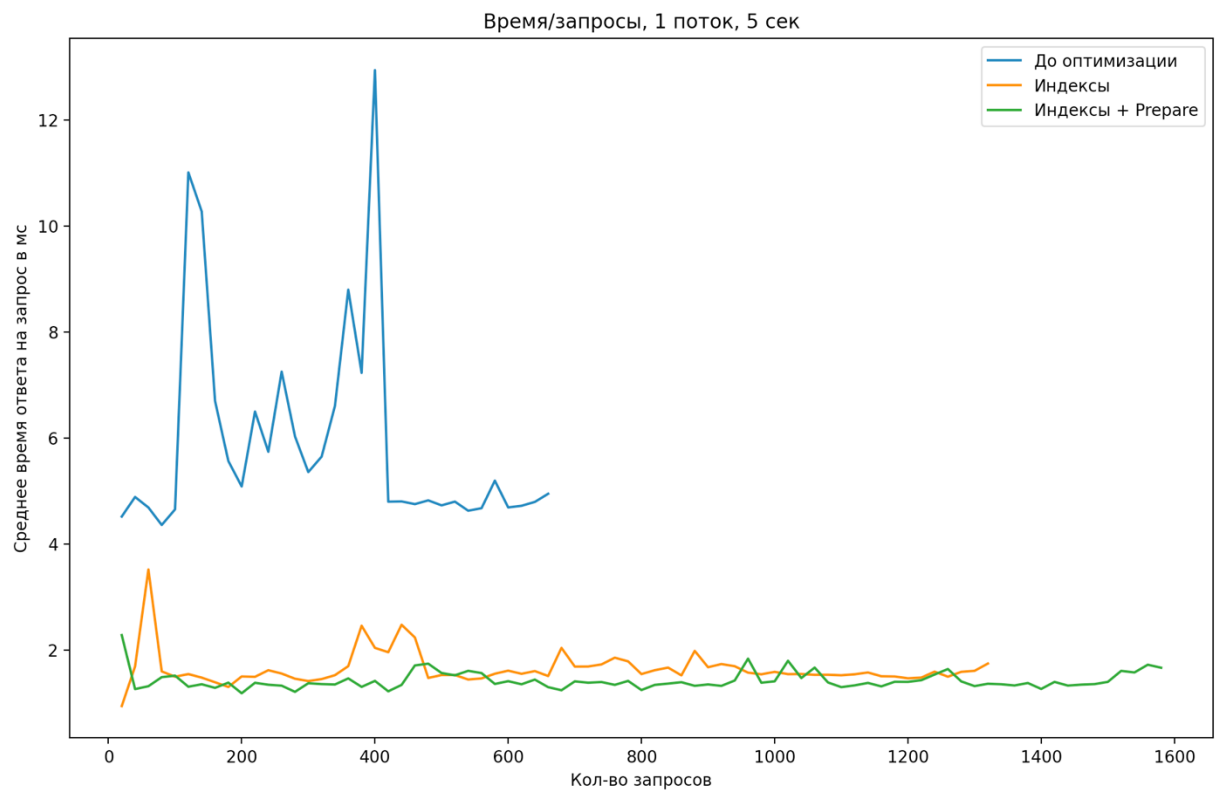


Рис. 4.3.2. График выполнения оптимизированных запросов для 1 потока на обычной бд

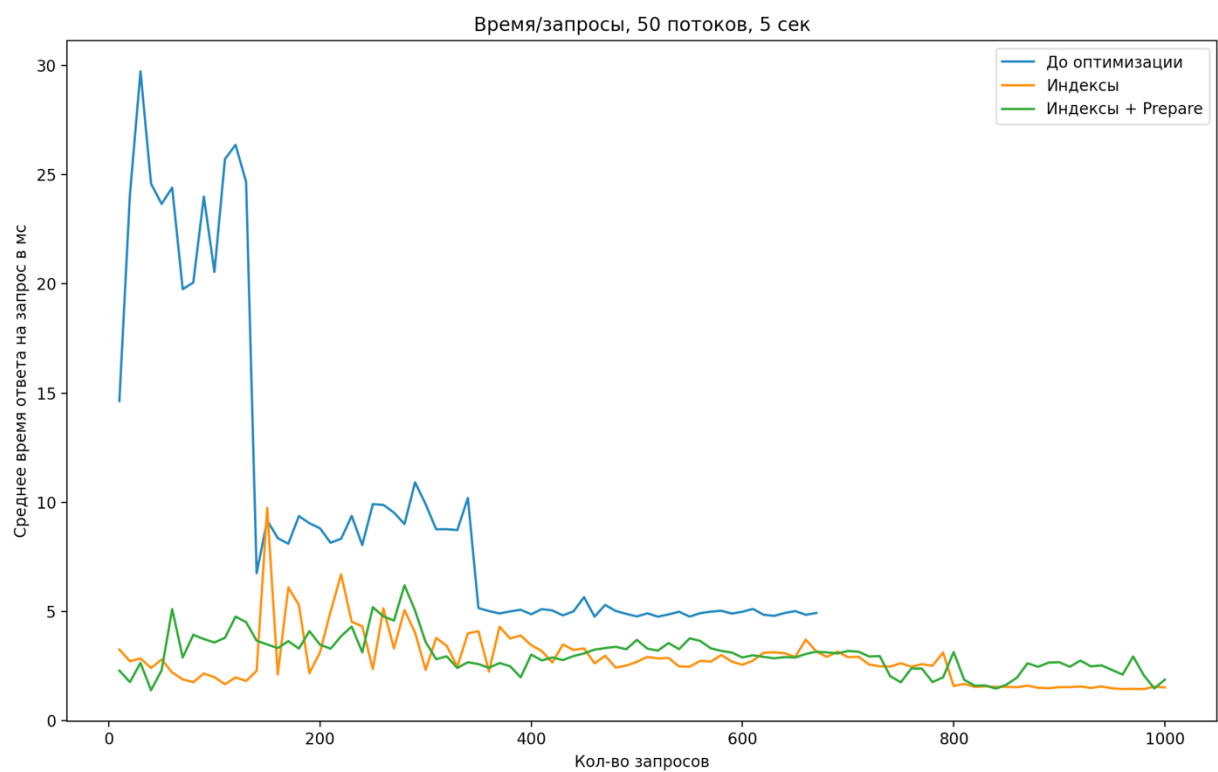


Рис. 4.3.3. График выполнения оптимизированных запросов для 50 потоков на обычной бд

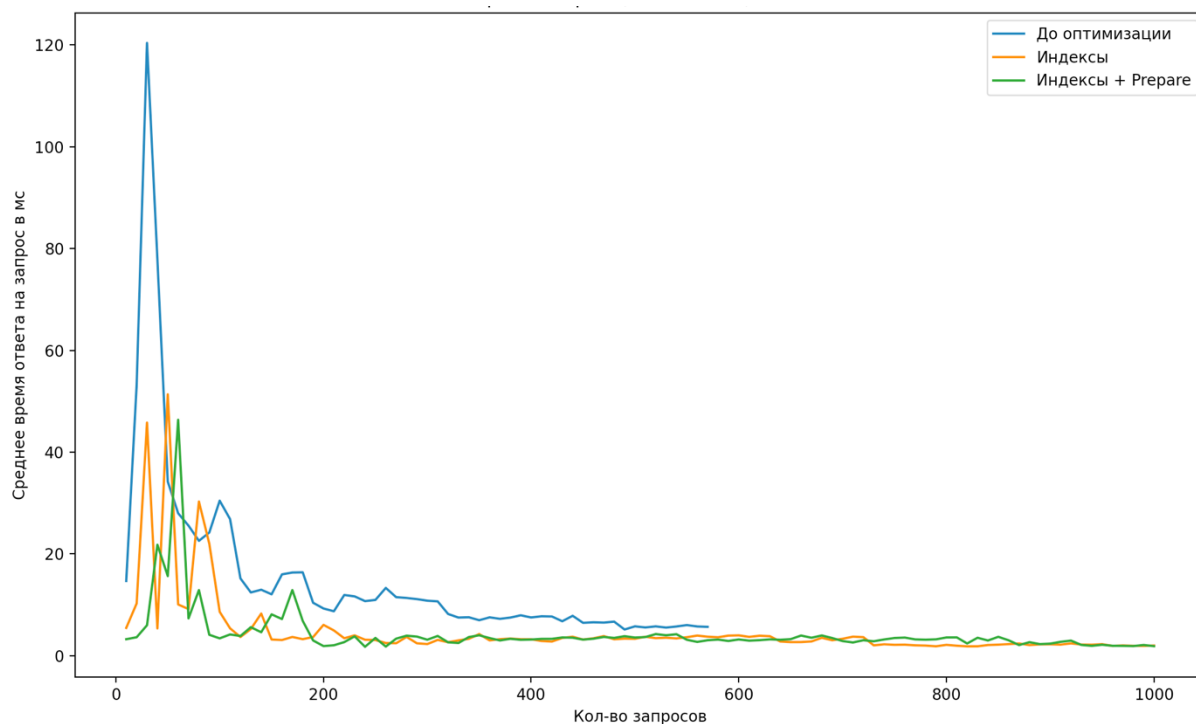


Рис. 4.3.4. График выполнения оптимизированных запросов для 97 потоков на обычной бд

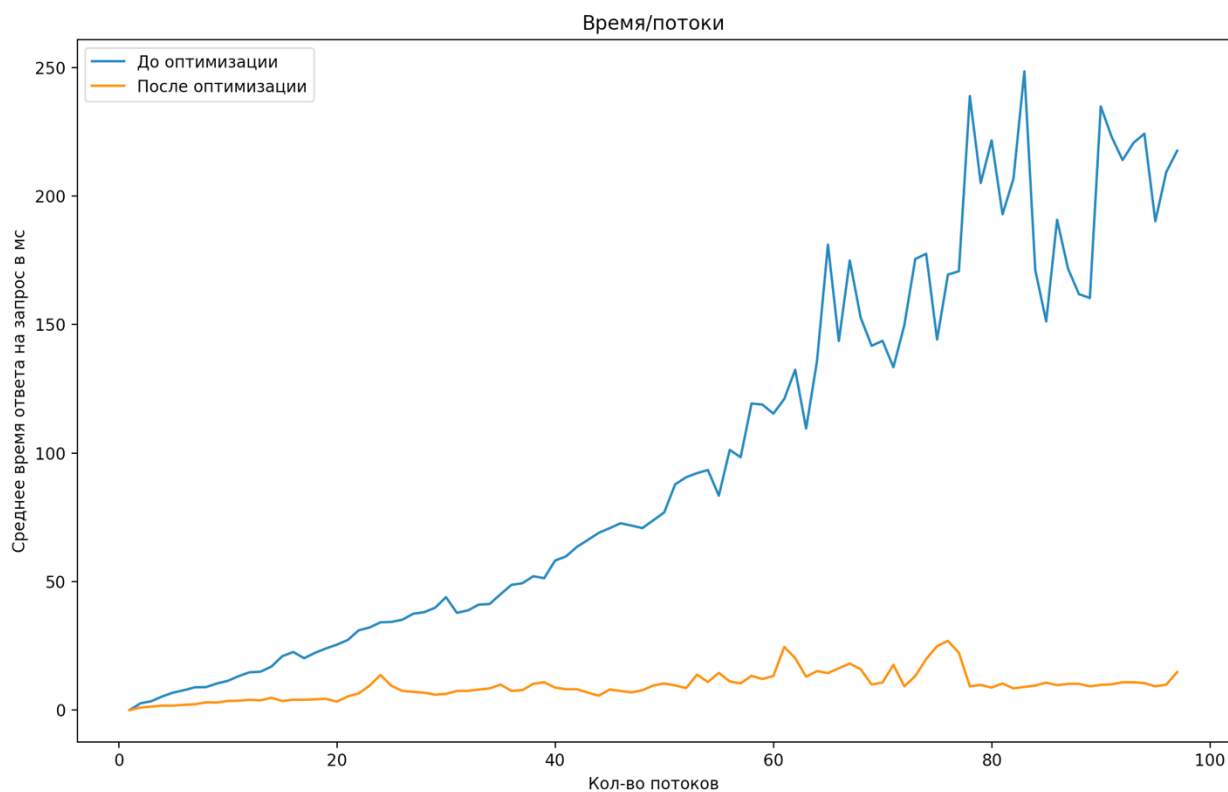


Рис. 4.3.5. График зависимости времени выполнения запросов от количества потоков на обычной бд до оптимизации запросов и после (index+Prepare)

Explain Analyze запроса 1 на обычной базе до и после оптимизации:

```
Unique (actual time=11.606..11.615 rows=3 loops=1)
-> GroupAggregate (actual time=11.605..11.611 rows=7 loops=1)
"
  Group Key: p.id, cap.name, m.person_id, m.enemy_id"
  Filter: ((count(((m.enemy_id = p.id) OR NULL::boolean)) <> 0) AND ((count(((m.person_id =
p.id) OR NULL::boolean)) / count(((m.enemy_id = p.id) OR NULL::boolean))) < 1))
  Rows Removed by Filter: 4
-> Sort (actual time=11.592..11.593 rows=11 loops=1)
"
  Sort Key: p.id, cap.name, m.person_id, m.enemy_id"
  Sort Method: quicksort Memory: 25kB
-> Nested Loop (actual time=2.578..11.561 rows=11 loops=1)
  Join Filter: ((p.id = m.person_id) OR (p.id = m.enemy_id))
  Rows Removed by Join Filter: 39941
-> Seq Scan on meetup m (actual time=0.012..1.917 rows=9988 loops=1)
-> Materialize (actual time=0.000..0.001 rows=4 loops=9988)
  -> Nested Loop (actual time=0.955..2.367 rows=4 loops=1)
    Join Filter: (p.class_of_person_id = cap.id)
    Rows Removed by Join Filter: 20
    -> Seq Scan on class_of_person cap (actual time=0.015..0.017 rows=6
loops=1)
    -> Materialize (actual time=0.072..0.389 rows=4 loops=6)
      -> Hash Join (actual time=0.429..2.326 rows=4 loops=1)
        Hash Cond: (p.user_id = ud.id)
        -> Seq Scan on person p (actual time=0.010..1.976 rows=2711
loops=1)
        Filter: (update_date <= (now() - '1 year'::interval))
        Rows Removed by Filter: 2296
      -> Hash (actual time=0.029..0.043 rows=1 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> Index Scan using user_data_nickname_key on user_data ud
(actual time=0.024..0.025 rows=1 loops=1)
          Index Cond: ((nickname)::text = 'Lamb5Wool'::text)
Planning Time: 0.389 ms
Execution Time: 11.736 ms

Индексы
Unique (actual time=0.107..0.115 rows=3 loops=1)
-> GroupAggregate (actual time=0.107..0.113 rows=7 loops=1)
"
  Group Key: p.id, cap.name, m.person_id, m.enemy_id"
  Filter: ((count(((m.enemy_id = p.id) OR NULL::boolean)) <> 0) AND ((count(((m.person_id =
p.id) OR NULL::boolean)) / count(((m.enemy_id = p.id) OR NULL::boolean))) < 1))
  Rows Removed by Filter: 4
-> Sort (actual time=0.099..0.099 rows=11 loops=1)
"
  Sort Key: p.id, cap.name, m.person_id, m.enemy_id"
  Sort Method: quicksort Memory: 25kB
-> Nested Loop (actual time=0.041..0.086 rows=11 loops=1)
  -> Nested Loop (actual time=0.028..0.040 rows=4 loops=1)
```

-> Nested Loop (actual time=0.025..0.032 rows=4 loops=1)
 -> Index Scan using user_nickname_idx on user_data ud (actual time=0.008..0.009 rows=1 loops=1)
 Index Cond: ((nickname)::text = 'Lamb5Wool'::text)
 -> Bitmap Heap Scan on person p (actual time=0.013..0.018 rows=4 loops=1)
 Recheck Cond: (user_id = ud.id)
 Filter: (update_date <= (now() - '1 year'::interval))
 Rows Removed by Filter: 2
 Heap Blocks: exact=6
 -> Bitmap Index Scan on person_user_idx (actual time=0.006..0.006 rows=6 loops=1)
 Index Cond: (user_id = ud.id)
 -> Index Scan using class_of_person_pkey on class_of_person cap (actual time=0.001..0.001 rows=1 loops=4)
 Index Cond: (id = p.class_of_person_id)
 -> Bitmap Heap Scan on meetup m (actual time=0.008..0.009 rows=3 loops=4)
 Recheck Cond: ((p.id = person_id) OR (p.id = enemy_id))
 Heap Blocks: exact=10
 -> BitmapOr (actual time=0.005..0.005 rows=0 loops=4)
 -> Bitmap Index Scan on meetup_person_idx (actual time=0.002..0.002 rows=1 loops=4)
 Index Cond: (person_id = p.id)
 -> Bitmap Index Scan on meetup_enemy_idx (actual time=0.003..0.003 rows=2 loops=4)
 Index Cond: (enemy_id = p.id)
 Planning Time: 0.479 ms
 Execution Time: 0.174 ms

C prepare

Unique (cost=42.69..43.25 rows=4 width=19) (actual time=0.023..0.023 rows=0 loops=1)
 -> GroupAggregate (cost=42.69..43.23 rows=4 width=19) (actual time=0.023..0.023 rows=0 loops=1)
 " Group Key: p.id, cap.name, m.person_id, m.enemy_id"
 Filter: (((count(((m.enemy_id = p.id) OR NULL::boolean)) <> 0) AND ((count(((m.person_id = p.id) OR NULL::boolean)) / count(((m.enemy_id = p.id) OR NULL::boolean))) < 1))
 -> Sort (cost=42.69..42.72 rows=12 width=19) (actual time=0.022..0.022 rows=0 loops=1)
 " Sort Key: p.id, cap.name, m.person_id, m.enemy_id"
 Sort Method: quicksort Memory: 25kB
 -> Nested Loop (cost=5.42..42.47 rows=12 width=19) (actual time=0.017..0.017 rows=0 loops=1)
 -> Nested Loop (cost=4.73..28.03 rows=3 width=11) (actual time=0.016..0.016 rows=0 loops=1)
 -> Nested Loop (cost=4.60..27.56 rows=3 width=8) (actual time=0.015..0.015 rows=0 loops=1)
 -> Index Scan using user_nickname_idx on user_data ud (cost=0.28..8.29 rows=1 width=4) (actual time=0.015..0.015 rows=0 loops=1)
 Index Cond: ((nickname)::text = (\$1)::text)

-> Bitmap Heap Scan on person p (cost=4.32..19.25 rows=2 width=12) (never executed)

Recheck Cond: (user_id = ud.id)
 Filter: (update_date <= (now() - '1 year'::interval))

-> Bitmap Index Scan on person_user_idx (cost=0.00..4.32 rows=5 width=0) (never executed)

Index Cond: (user_id = ud.id)

-> Index Scan using class_of_person_pkey on class_of_person cap (cost=0.13..0.15 rows=1 width=11) (never executed)

Index Cond: (id = p.class_of_person_id)

-> Bitmap Heap Scan on meetup m (cost=0.69..4.77 rows=5 width=8) (never executed)

Recheck Cond: ((p.id = person_id) OR (p.id = enemy_id))

-> BitmapOr (cost=0.69..0.69 rows=5 width=0) (never executed)

-> Bitmap Index Scan on meetup_person_idx (cost=0.00..0.34 rows=2 width=0) (never executed)

Index Cond: (person_id = p.id)

-> Bitmap Index Scan on meetup_enemy_idx (cost=0.00..0.34 rows=2 width=0) (never executed)

Index Cond: (enemy_id = p.id)

Planning Time: 0.016 ms
 Execution Time: 0.126 ms

Explain Analyze запроса 2 на обычной базе до и после оптимизации:

Запрос 2 база обыч

Sort (actual time=6.754..6.783 rows=409 loops=1)
 Sort Key: (count(m.*)) DESC
 Sort Method: quicksort Memory: 52kB

-> HashAggregate (actual time=6.599..6.666 rows=409 loops=1)
 Group Key: ud.nickname

-> Hash Join (actual time=2.536..6.200 rows=629 loops=1)
 Hash Cond: (p.user_id = ud.id)

-> Hash Join (actual time=2.049..5.462 rows=706 loops=1)
 Hash Cond: (m.person_id = p.id)

-> Seq Scan on meetup m (actual time=0.016..3.093 rows=706 loops=1)
 Filter: ((result = 'win'::meetup_result) AND (meetup_date < now()) AND (meetup_date >= (now() - '1 year'::interval)))
 Rows Removed by Filter: 9282

-> Hash (actual time=2.020..2.020 rows=5007 loops=1)
 Buckets: 8192 Batches: 1 Memory Usage: 258kB

-> Seq Scan on person p (actual time=0.008..1.021 rows=5007 loops=1)

-> Hash (actual time=0.481..0.481 rows=1001 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 55kB

-> Seq Scan on user_data ud (actual time=0.012..0.254 rows=1001 loops=1)

Planning Time: 0.574 ms
 Execution Time: 6.948 ms

Индексы

Sort (actual time=3.693..3.713 rows=409 loops=1)

Sort Key: (count(m.*)) DESC

Sort Method: quicksort Memory: 52kB

-> HashAggregate (actual time=3.491..3.566 rows=409 loops=1)

Group Key: ud.nickname

-> Hash Join (actual time=2.433..3.278 rows=629 loops=1)

Hash Cond: (p.user_id = ud.id)

-> Hash Join (actual time=1.973..2.638 rows=706 loops=1)

Hash Cond: (m.person_id = p.id)

-> Bitmap Heap Scan on meetup m (actual time=0.432..0.951 rows=706 loops=1)

Recheck Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date

< now()))

Filter: (result = 'win'::meetup_result)

Rows Removed by Filter: 1466

Heap Blocks: exact=74

-> Bitmap Index Scan on meetup_date_idx (actual time=0.403..0.403 rows=2172

loops=1)

Index Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date

< now()))

-> Hash (actual time=1.514..1.514 rows=5007 loops=1)

Buckets: 8192 Batches: 1 Memory Usage: 258kB

-> Seq Scan on person p (actual time=0.016..0.747 rows=5007 loops=1)

-> Hash (actual time=0.440..0.440 rows=1001 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 55kB

-> Seq Scan on user_data ud (actual time=0.014..0.218 rows=1001 loops=1)

Planning Time: 0.818 ms

Execution Time: 3.875 ms

C Prepare

Sort (cost=412.82..414.59 rows=708 width=18) (actual time=3.079..3.102 rows=413 loops=1)

Sort Key: (count(m.*)) DESC

Sort Method: quicksort Memory: 53kB

-> HashAggregate (cost=372.22..379.30 rows=708 width=18) (actual time=2.961..3.009 rows=413 loops=1)

Group Key: ud.nickname

-> Hash Join (cost=237.88..368.68 rows=708 width=58) (actual time=1.888..2.761 rows=666 loops=1)

Hash Cond: (p.user_id = ud.id)

-> Hash Join (cost=204.36..333.29 rows=708 width=52) (actual time=1.596..2.305 rows=725 loops=1)

Hash Cond: (m.person_id = p.id)

-> Bitmap Heap Scan on meetup m (cost=49.70..176.77 rows=708 width=52) (actual time=0.382..0.933 rows=725 loops=1)

Recheck Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date

< now()))

Filter: (result = \$1)

Rows Removed by Filter: 1394


```

        Heap Blocks: exact=74
        -> Bitmap Index Scan on meetup_date_idx (cost=0.00..49.52 rows=2123
width=0) (actual time=0.366..0.366 rows=2119 loops=1)
            Index Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date
< now()))
        -> Hash (cost=92.07..92.07 rows=5007 width=8) (actual time=1.203..1.203
rows=5007 loops=1)
            Buckets: 8192 Batches: 1 Memory Usage: 258kB
        -> Seq Scan on person p (cost=0.00..92.07 rows=5007 width=8) (actual
time=0.008..0.606 rows=5007 loops=1)
        -> Hash (cost=21.01..21.01 rows=1001 width=14) (actual time=0.287..0.287 rows=1001
loops=1)
            Buckets: 1024 Batches: 1 Memory Usage: 55kB
        -> Seq Scan on user_data ud (cost=0.00..21.01 rows=1001 width=14) (actual
time=0.013..0.152 rows=1001 loops=1)
Planning Time: 0.015 ms
Execution Time: 3.199 ms

```

Explain Analyze запроса 3 на обычной базе до и после оптимизации:

```

GroupAggregate (actual time=4.323..4.328 rows=9 loops=1)
  Group Key: i.name
  -> Sort (actual time=4.312..4.312 rows=10 loops=1)
      Sort Key: i.name
      Sort Method: quicksort Memory: 25kB
  -> Nested Loop (actual time=1.718..4.294 rows=10 loops=1)
      -> Index Only Scan using person_pkey on person p (actual time=0.013..0.015 rows=1
loops=1)
          Index Cond: (id = 1234)
          Heap Fetches: 1
      -> Nested Loop (actual time=1.704..4.277 rows=10 loops=1)
          -> Hash Join (actual time=1.678..4.226 rows=10 loops=1)
              Hash Cond: (ipi.inventory_person_id = ip.id)
              -> Seq Scan on inventory_person_items ipi (actual time=0.014..1.175 rows=9737
loops=1)
                  -> Hash (actual time=1.597..1.597 rows=3 loops=1)
                      Buckets: 1024 Batches: 1 Memory Usage: 9kB
                      -> Seq Scan on inventory_person ip (actual time=0.154..1.588 rows=3
loops=1)
                          Filter: (person_id = 1234)
                          Rows Removed by Filter: 12002
          -> Index Scan using item_pkey on item i (actual time=0.003..0.003 rows=1 loops=10)
              Index Cond: (id = ipi.item_id)
Planning Time: 0.482 ms
Execution Time: 4.384 ms

```

Индексы

```

GroupAggregate (actual time=0.100..0.105 rows=9 loops=1)

```

Group Key: i.name

-> Sort (actual time=0.093..0.093 rows=10 loops=1)

Sort Key: i.name

Sort Method: quicksort Memory: 25kB

-> Nested Loop (actual time=0.054..0.082 rows=10 loops=1)

-> Index Only Scan using person_pkey on person p (actual time=0.022..0.022 rows=1 loops=1)

Index Cond: (id = 1234)

Heap Fetches: 1

-> Nested Loop (actual time=0.028..0.055 rows=10 loops=1)

-> Nested Loop (actual time=0.022..0.037 rows=10 loops=1)

-> Bitmap Heap Scan on inventory_person ip (actual time=0.014..0.017 rows=3 loops=1)

Recheck Cond: (person_id = 1234)

Heap Blocks: exact=3

-> Bitmap Index Scan on inventory_person_person_idx (actual time=0.011..0.011 rows=3 loops=1)

Index Cond: (person_id = 1234)

-> Index Scan using inventory_person_item_inventory_person_item_idx on inventory_person_items ipi (actual time=0.003..0.005 rows=3 loops=3)

Index Cond: (inventory_person_id = ip.id)

-> Index Scan using item_pkey on item i (actual time=0.001..0.001 rows=1 loops=10)

Index Cond: (id = ipi.item_id)

Planning Time: 0.432 ms

Execution Time: 0.153 ms

C Prepare

GroupAggregate (cost=36.54..36.57 rows=2 width=18) (actual time=0.088..0.093 rows=9 loops=1)

Group Key: i.name

-> Sort (cost=36.54..36.54 rows=2 width=14) (actual time=0.082..0.083 rows=10 loops=1)

Sort Key: i.name

Sort Method: quicksort Memory: 25kB

-> Nested Loop (cost=5.00..36.53 rows=2 width=14) (actual time=0.042..0.069 rows=10 loops=1)

-> Index Only Scan using person_pkey on person p (cost=0.28..8.30 rows=1 width=4) (actual time=0.016..0.017 rows=1 loops=1)

Index Cond: (id = \$1)

Heap Fetches: 1

-> Nested Loop (cost=4.72..28.21 rows=2 width=18) (actual time=0.023..0.048 rows=10 loops=1)

-> Nested Loop (cost=4.59..27.90 rows=2 width=12) (actual time=0.019..0.034 rows=10 loops=1)

-> Bitmap Heap Scan on inventory_person ip (cost=4.30..11.27 rows=2 width=8) (actual time=0.010..0.011 rows=3 loops=1)

Recheck Cond: (person_id = \$1)

Heap Blocks: exact=3

-> Bitmap Index Scan on inventory_person_person_idx (cost=0.00..4.30 rows=2 width=0) (actual time=0.007..0.008 rows=3 loops=1)
 Index Cond: (person_id = \$1)
 -> Index Scan using inventory_person_item_inventory_person_item_idx on inventory_person_items ipi (cost=0.29..8.30 rows=1 width=12) (actual time=0.004..0.006 rows=3 loops=3)
 Index Cond: (inventory_person_id = ip.id)
 -> Index Scan using item_pkey on item i (cost=0.14..0.15 rows=1 width=14) (actual time=0.001..0.001 rows=1 loops=10)
 Index Cond: (id = ipi.item_id)
 Planning Time: 0.021 ms
 Execution Time: 0.157 ms

Explain Analyze запроса 4 на обычной базе до и после оптимизации:

GroupAggregate (actual time=11.838..11.839 rows=1 loops=1)
 Group Key: ps.person_id
 Filter: (count((ps.equiped OR NULL::boolean)) = 2)
 Rows Removed by Filter: 4
 -> Sort (actual time=11.821..11.821 rows=10 loops=1)
 Sort Key: ps.person_id
 Sort Method: quicksort Memory: 25kB
 -> Hash Join (actual time=7.946..11.802 rows=10 loops=1)
 Hash Cond: (ip.id = ipi.inventory_person_id)
 -> Hash Join (actual time=5.747..9.780 rows=200 loops=1)
 Hash Cond: (ip.person_id = ps.person_id)
 -> Seq Scan on inventory_person ip (actual time=0.022..1.759 rows=12005 loops=1)
 -> Hash (actual time=5.672..5.673 rows=88 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 12kB
 -> Hash Join (actual time=1.380..5.544 rows=88 loops=1)
 Hash Cond: (ps.skill_id = s.id)
 -> Hash Join (actual time=1.234..5.349 rows=498 loops=1)
 Hash Cond: (ps.person_id = p.id)
 -> Seq Scan on person_skill ps (actual time=0.016..1.684 rows=10008 loops=1)
 -> Hash (actual time=1.201..1.201 rows=249 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 17kB
 -> Seq Scan on person p (actual time=0.019..1.113 rows=249 loops=1)
 Filter: (health = 300)
 Rows Removed by Filter: 4758
 -> Hash (actual time=0.053..0.053 rows=2 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Hash Join (actual time=0.044..0.050 rows=2 loops=1)
 Hash Cond: (s.class_of_person_id = cop.id)
 -> Seq Scan on skill s (actual time=0.013..0.019 rows=13 loops=1)
 -> Hash (actual time=0.018..0.018 rows=1 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB

```

-> Seq Scan on class_of_person cop (actual time=0.013..0.015
rows=1 loops=1)
    Filter: ((name)::text = 'Archer'::text)
    Rows Removed by Filter: 5
-> Hash (actual time=1.958..1.958 rows=674 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 32kB
-> Seq Scan on inventory_person_items ipi (actual time=0.032..1.447 rows=674
loops=1)
    Filter: (item_id = 11)
    Rows Removed by Filter: 9063
Planning Time: 1.690 ms
Execution Time: 11.966 ms

Индексы
GroupAggregate (actual time=2.732..2.735 rows=1 loops=1)
  Group Key: ps.person_id
  Filter: (count((ps.equiped OR NULL)::boolean)) = 2)
  Rows Removed by Filter: 4
-> Sort (actual time=2.705..2.706 rows=10 loops=1)
  Sort Key: ps.person_id
  Sort Method: quicksort Memory: 25kB
-> Nested Loop (actual time=0.377..2.680 rows=10 loops=1)
  -> Nested Loop (actual time=0.196..2.094 rows=200 loops=1)
  -> Hash Join (actual time=0.191..1.461 rows=88 loops=1)
    Hash Cond: (ps.person_id = p.id)
  -> Nested Loop (actual time=0.035..1.056 rows=1684 loops=1)
    -> Hash Join (actual time=0.025..0.041 rows=2 loops=1)
      Hash Cond: (s.class_of_person_id = cop.id)
    -> Seq Scan on skill s (actual time=0.009..0.020 rows=13 loops=1)
    -> Hash (actual time=0.010..0.010 rows=1 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 9kB
    -> Seq Scan on class_of_person cop (actual time=0.006..0.007 rows=1
loops=1)
      Filter: ((name)::text = 'Archer'::text)
      Rows Removed by Filter: 5
    -> Index Scan using person_skill_skill_idx on person_skill ps (actual
time=0.009..0.384 rows=842 loops=2)
      Index Cond: (skill_id = s.id)
  -> Hash (actual time=0.132..0.132 rows=249 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 17kB
  -> Bitmap Heap Scan on person p (actual time=0.022..0.103 rows=249
loops=1)
    Recheck Cond: (health = 300)
    Heap Blocks: exact=42
  -> Bitmap Index Scan on person_health_idx (actual time=0.015..0.015
rows=249 loops=1)
    Index Cond: (health = 300)

```

-> Index Scan using inventory_person_person_idx on inventory_person ip (actual time=0.005..0.006 rows=2 loops=88)
 Index Cond: (person_id = ps.person_id)

-> Index Only Scan using inventory_person_item_inventory_person_item_idx on inventory_person_items ipi (actual time=0.002..0.002 rows=0 loops=200)
 Index Cond: ((inventory_person_id = ip.id) AND (item_id = 11))
 Heap Fetches: 2
 Planning Time: 1.193 ms
 Execution Time: 2.872 ms

C Prepare

GroupAggregate (cost=252.94..253.16 rows=1 width=4) (actual time=1.647..1.647 rows=1 loops=1)
 Group Key: ps.person_id
 Filter: (count((ps.equiped OR NULL)::boolean)) = 2)
 Rows Removed by Filter: 4

-> Sort (cost=252.94..252.97 rows=11 width=5) (actual time=1.638..1.638 rows=10 loops=1)
 Sort Key: ps.person_id
 Sort Method: quicksort Memory: 25kB

-> Nested Loop (cost=56.38..252.75 rows=11 width=5) (actual time=0.404..1.625 rows=10 loops=1)
 -> Nested Loop (cost=56.09..187.61 rows=199 width=9) (actual time=0.216..1.304 rows=200 loops=1)
 -> Hash Join (cost=55.81..156.08 rows=83 width=9) (actual time=0.209..0.999 rows=88 loops=1)
 Hash Cond: (ps.person_id = p.id)
 -> Nested Loop (cost=1.37..97.26 rows=1668 width=5) (actual time=0.042..0.707 rows=1684 loops=1)
 -> Hash Join (cost=1.09..2.27 rows=2 width=4) (actual time=0.029..0.033 rows=2 loops=1)
 Hash Cond: (s.class_of_person_id = cop.id)
 -> Seq Scan on skill s (cost=0.00..1.13 rows=13 width=8) (actual time=0.012..0.014 rows=13 loops=1)
 -> Hash (cost=1.07..1.07 rows=1 width=4) (actual time=0.010..0.011 rows=1 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Seq Scan on class_of_person cop (cost=0.00..1.07 rows=1 width=4) (actual time=0.006..0.007 rows=1 loops=1)
 Filter: ((name)::text = (\$1)::text)
 Rows Removed by Filter: 5

-> Index Scan using person_skill_skill_idx on person_skill ps (cost=0.29..39.15 rows=834 width=9) (actual time=0.009..0.225 rows=842 loops=2)
 Index Cond: (skill_id = s.id)
 -> Hash (cost=51.32..51.32 rows=249 width=4) (actual time=0.139..0.139 rows=249 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 17kB
 -> Bitmap Heap Scan on person p (cost=6.21..51.32 rows=249 width=4) (actual time=0.028..0.103 rows=249 loops=1)

```

Recheck Cond: (health = 300)
Heap Blocks: exact=42
-> Bitmap Index Scan on person_health_idx (cost=0.00..6.15 rows=249
width=0) (actual time=0.020..0.021 rows=249 loops=1)
    Index Cond: (health = 300)
-> Index Scan using inventory_person_person_idx on inventory_person ip
(cost=0.29..0.36 rows=2 width=8) (actual time=0.002..0.003 rows=2 loops=88)
    Index Cond: (person_id = ps.person_id)
-> Index Only Scan using inventory_person_item_inventory_person_item_idx on
inventory_person_items ipi (cost=0.29..0.32 rows=1 width=4) (actual time=0.001..0.001
rows=0 loops=200)
    Index Cond: ((inventory_person_id = ip.id) AND (item_id = 11))
    Heap Fetches: 2
Planning Time: 0.017 ms
Execution Time: 1.752 ms

```

Explain Analyze запроса 5 на обычной базе до и после оптимизации:

```

Sort (actual time=6.774..6.791 rows=225 loops=1)
Sort Key: (sum(ipi.amount)) DESC
Sort Method: quicksort Memory: 40kB
-> GroupAggregate (actual time=6.552..6.708 rows=225 loops=1)
    Group Key: ud.nickname
    -> Sort (actual time=6.530..6.558 rows=272 loops=1)
        Sort Key: ud.nickname
        Sort Method: quicksort Memory: 39kB
        -> Hash Join (actual time=1.816..6.296 rows=272 loops=1)
            Hash Cond: (p.user_id = ud.id)
            -> Nested Loop (actual time=1.520..5.770 rows=294 loops=1)
                -> Hash Join (actual time=1.502..4.189 rows=294 loops=1)
                    Hash Cond: (ip.id = ipi.inventory_person_id)
                    -> Seq Scan on inventory_person ip (actual time=0.007..1.265 rows=12005
loops=1)
                    -> Hash (actual time=1.487..1.488 rows=294 loops=1)
                        Buckets: 1024 Batches: 1 Memory Usage: 20kB
                        -> Seq Scan on inventory_person_items ipi (actual time=0.014..1.399
rows=294 loops=1)
                Filter: ((item_id = 11) AND (update_date < (now() -
'00:00:00'::interval)) AND (update_date >= (now() - '1 year'::interval)) AND (add_date < (now() -
'00:00:00'::interval)) AND (add_date >= (now() - '1 year'::interval)))
                Rows Removed by Filter: 9443
            -> Index Scan using person_pkey on person p (actual time=0.005..0.005 rows=1
loops=294)
                Index Cond: (id = ip.person_id)
            -> Hash (actual time=0.289..0.289 rows=1001 loops=1)
                Buckets: 1024 Batches: 1 Memory Usage: 55kB
                -> Seq Scan on user_data ud (actual time=0.013..0.146 rows=1001 loops=1)
Planning Time: 0.484 ms
Execution Time: 6.865 ms

```

Индексы

Sort (actual time=4.182..4.193 rows=223 loops=1)

Sort Key: (sum(ipi.amount)) DESC

Sort Method: quicksort Memory: 40kB

-> HashAggregate (actual time=4.096..4.128 rows=223 loops=1)

Group Key: ud.nickname

-> Hash Join (actual time=3.670..3.970 rows=270 loops=1)

Hash Cond: (ud.id = p.user_id)

-> Seq Scan on user_data ud (actual time=0.008..0.139 rows=1001 loops=1)

-> Hash (actual time=3.652..3.652 rows=270 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 19kB

-> Nested Loop (actual time=0.279..3.545 rows=292 loops=1)

-> Hash Join (actual time=0.273..2.654 rows=292 loops=1)

Hash Cond: (ip.id = ipi.inventory_person_id)

-> Seq Scan on inventory_person ip (actual time=0.005..1.155 rows=12005

loops=1)

-> Hash (actual time=0.263..0.263 rows=292 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 20kB

-> Bitmap Heap Scan on inventory_person_items ipi (actual

time=0.115..0.220 rows=292 loops=1)

Recheck Cond: ((item_id = 11) AND (add_date < now()) AND (add_date
>= (now() - '1 year'::interval)) AND (update_date < now()) AND (update_date >= (now() - '1
year'::interval)))

Heap Blocks: exact=74

-> Bitmap Index Scan on

inventory_person_items_item_addupdate_date_idx (actual time=0.106..0.106 rows=292
loops=1)

Index Cond: ((item_id = 11) AND (add_date < now()) AND (add_date
>= (now() - '1 year'::interval)) AND (update_date < now()) AND (update_date >= (now() - '1
year'::interval)))

-> Index Scan using person_pkey on person p (actual time=0.003..0.003 rows=1
loops=292)

Index Cond: (id = ip.person_id)

Planning Time: 0.658 ms

Execution Time: 4.307 ms

C Prepare

Sort (cost=17.32..17.33 rows=1 width=18) (actual time=2.165..2.186 rows=223 loops=1)

Sort Key: (sum(ipi.amount)) DESC

Sort Method: quicksort Memory: 40kB

-> GroupAggregate (cost=17.29..17.31 rows=1 width=18) (actual time=2.007..2.121 rows=223
loops=1)

Group Key: ud.nickname

-> Sort (cost=17.29..17.30 rows=1 width=14) (actual time=2.001..2.016 rows=270 loops=1)

Sort Key: ud.nickname

Sort Method: quicksort Memory: 39kB

```

-> Nested Loop (cost=1.13..17.28 rows=1 width=14) (actual time=0.066..1.874
rows=270 loops=1)
  -> Nested Loop (cost=0.86..16.98 rows=1 width=8) (actual time=0.045..1.412
rows=292 loops=1)
    -> Nested Loop (cost=0.57..16.66 rows=1 width=8) (actual time=0.038..0.870
rows=292 loops=1)
      -> Index Scan using inventory_person_items_item_addupdate_date_idx on
inventory_person_items ipi (cost=0.29..8.35 rows=1 width=8) (actual time=0.029..0.232
rows=292 loops=1)
        Index Cond: ((item_id = 11) AND (add_date < $1) AND (add_date >= ($1 -
'1 year'::interval)) AND (update_date < $1) AND (update_date >= ($1 - '1 year'::interval)))
      -> Index Scan using inventory_person_pkey on inventory_person ip
(cost=0.29..8.30 rows=1 width=8) (actual time=0.002..0.002 rows=1 loops=292)
        Index Cond: (id = ipi.inventory_person_id)
    -> Index Scan using person_pkey on person p (cost=0.28..0.32 rows=1 width=8)
(actual time=0.002..0.002 rows=1 loops=292)
      Index Cond: (id = ip.person_id)
  -> Index Scan using user_data_pkey on user_data ud (cost=0.28..0.31 rows=1
width=14) (actual time=0.001..0.001 rows=1 loops=292)
    Index Cond: (id = p.user_id)
Planning Time: 0.025 ms
Execution Time: 2.276 ms

```

В результате проведения экспериментов с большой базой данных были построены графики для 1, 50 и 97 (макс подключений для бд) потоков. Единица времени равна 20 секунда.. Шаг изменения количества запросов равен 10. Графики представлены на рисунках 4.3.6–4.3.12.

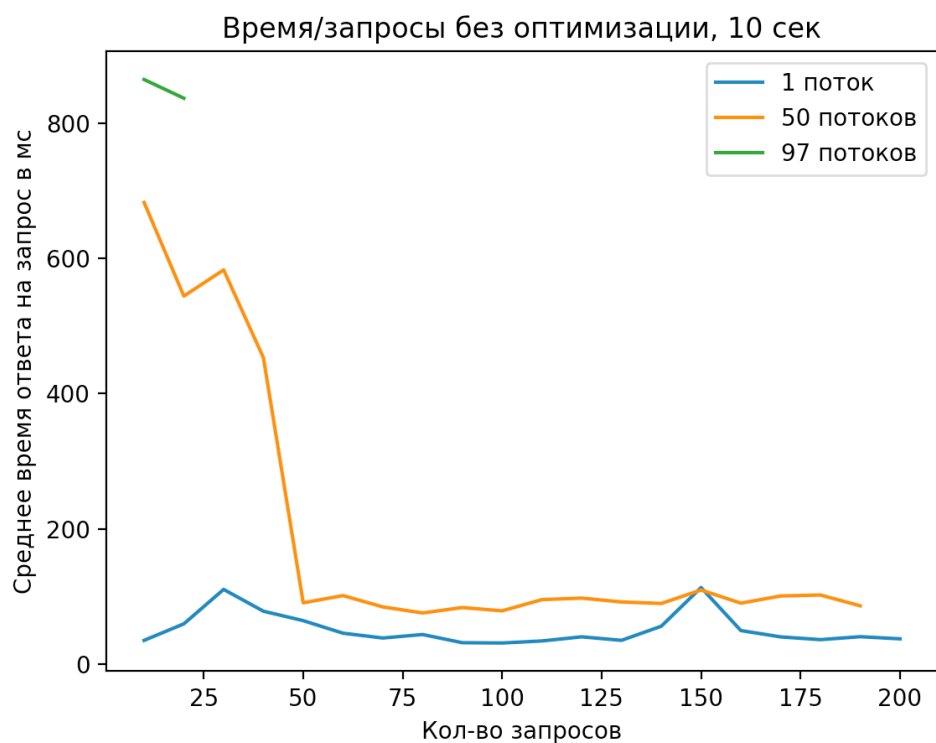


Рис. 4.3.6. Плохой график выполнения запросов без оптимизации для 1, 50 и 97 потоков на большой бд с единицей времени 10 сек

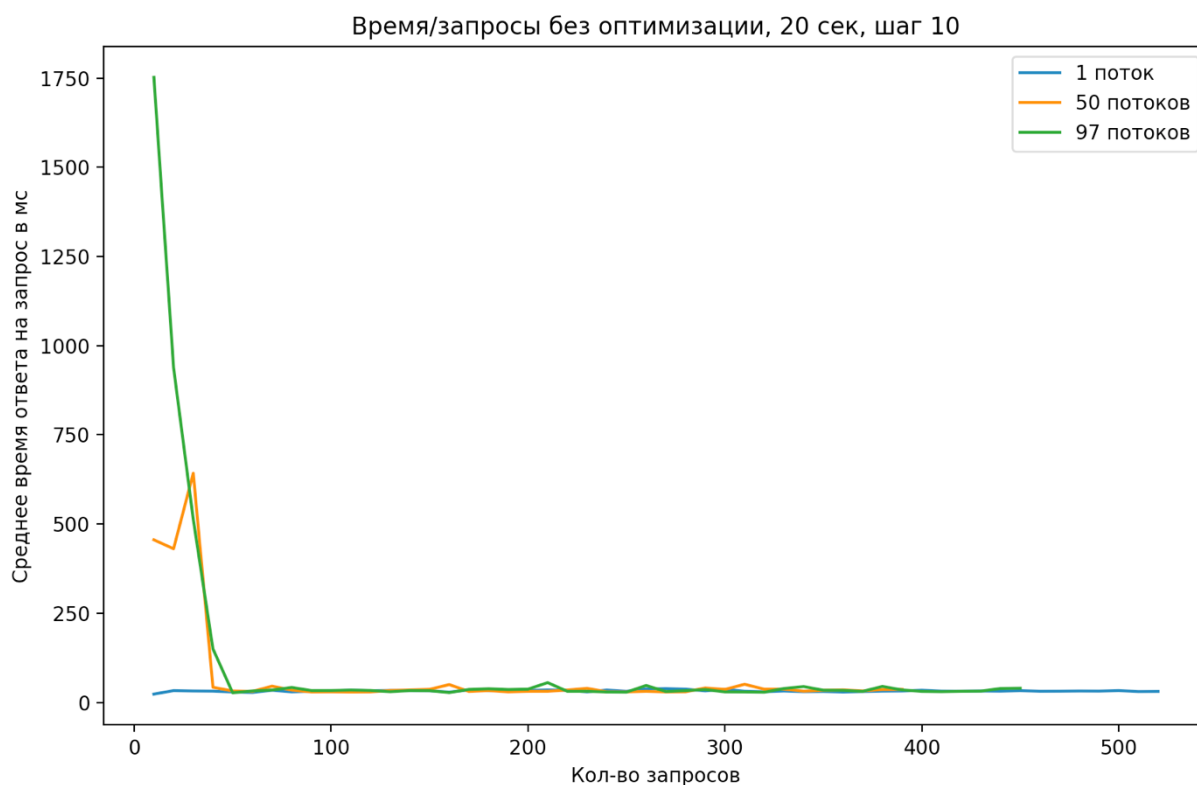


Рис. 4.3.7. График выполнения запросов без оптимизации для 1, 50 и 97 потоков на большой бд с единицей времени 20 сек

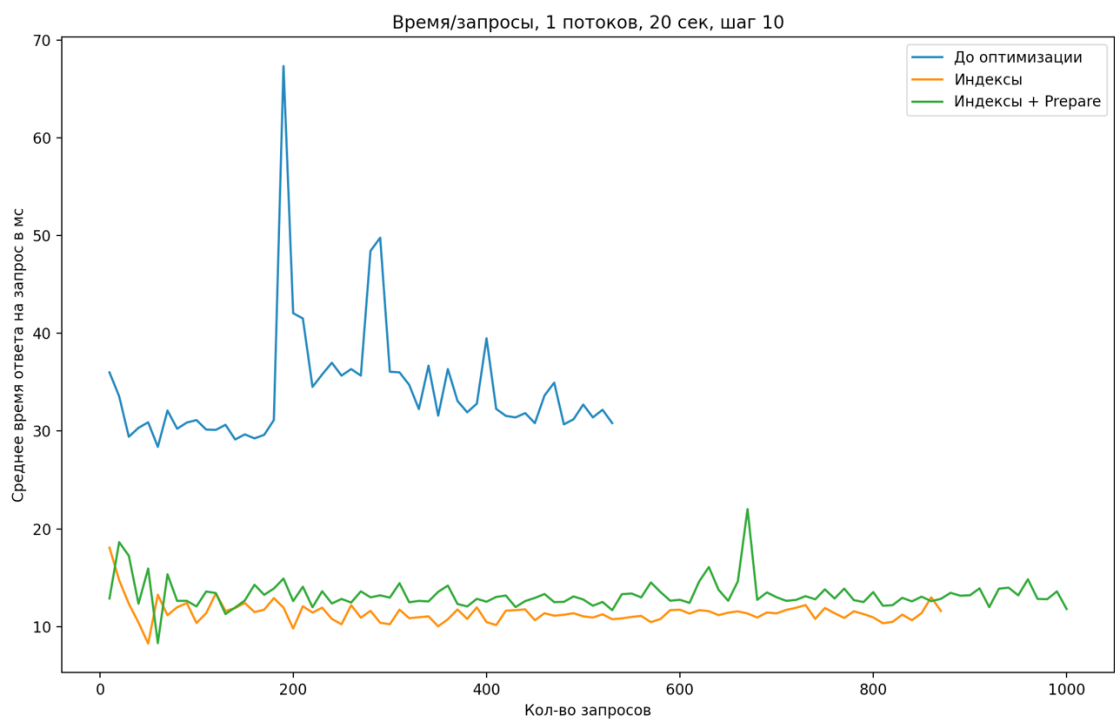


Рис. 4.3.8. График выполнения оптимизированных запросов для 1 потока на большой бд

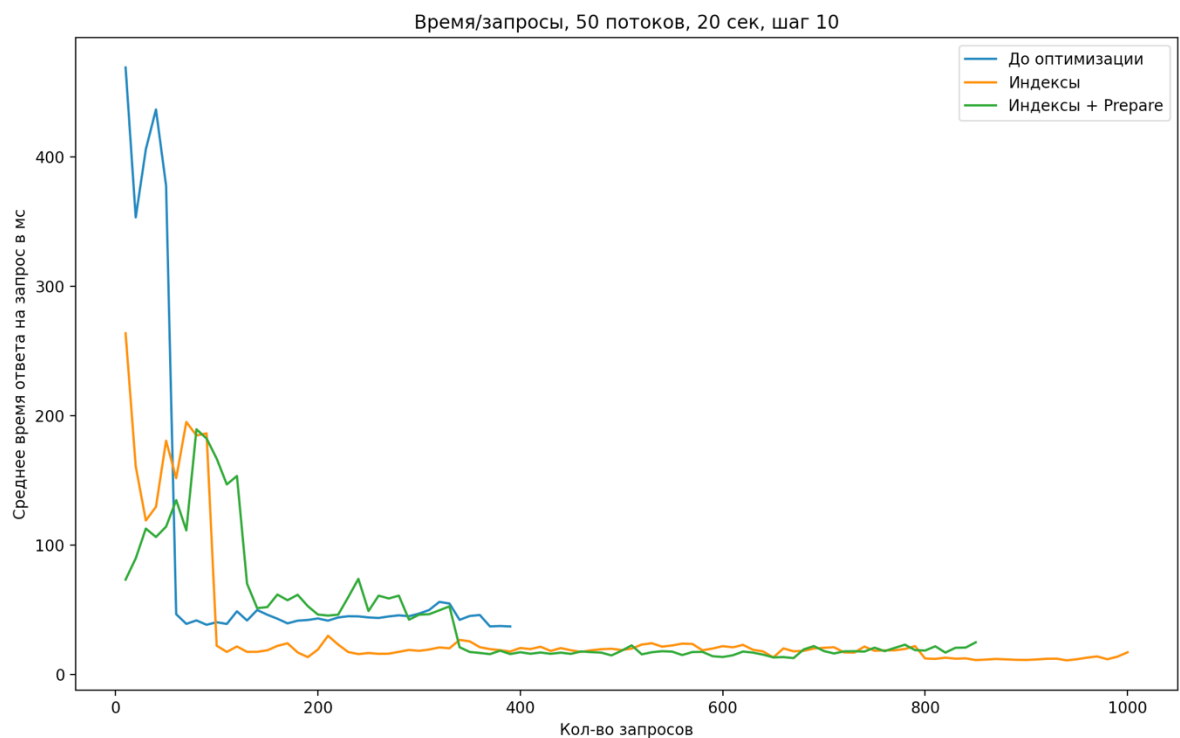


Рис. 4.3.9. График выполнения оптимизированных запросов для 50 потоков на большой бд

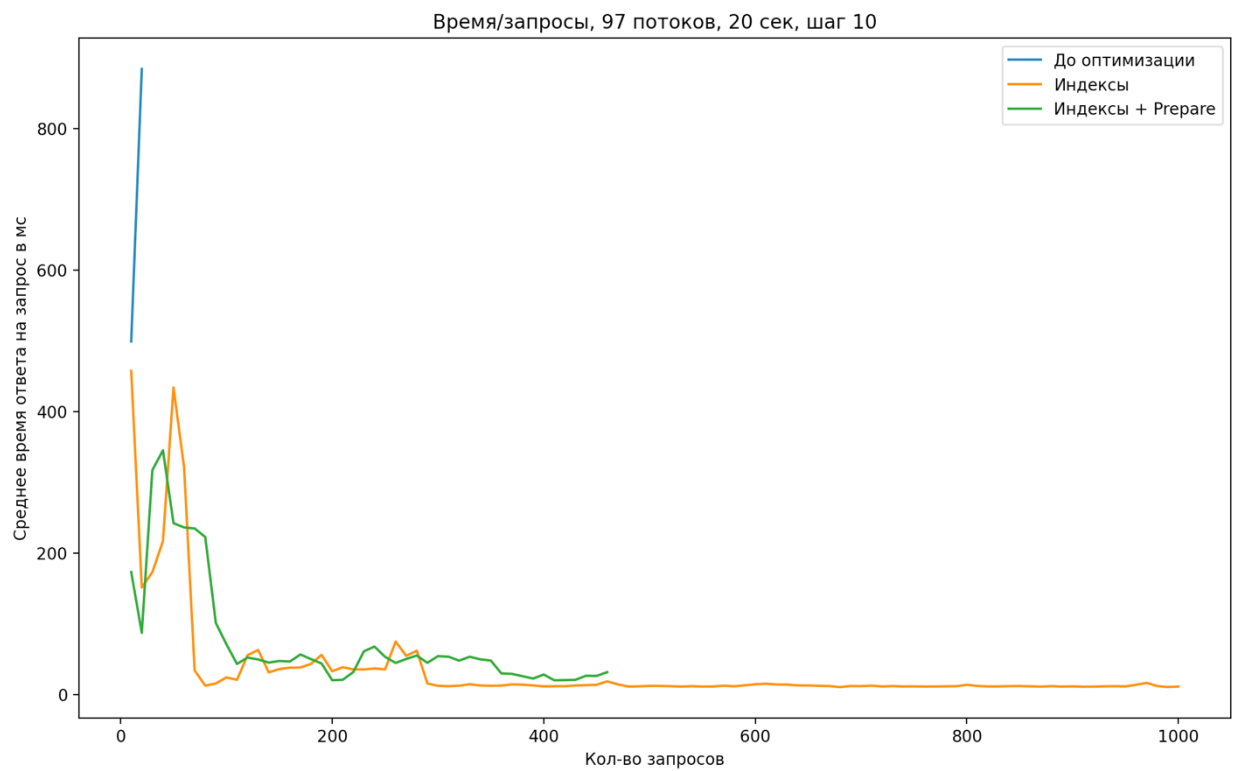


Рис. 4.3.10. График выполнения оптимизированных запросов для 97 потоков на большой бд

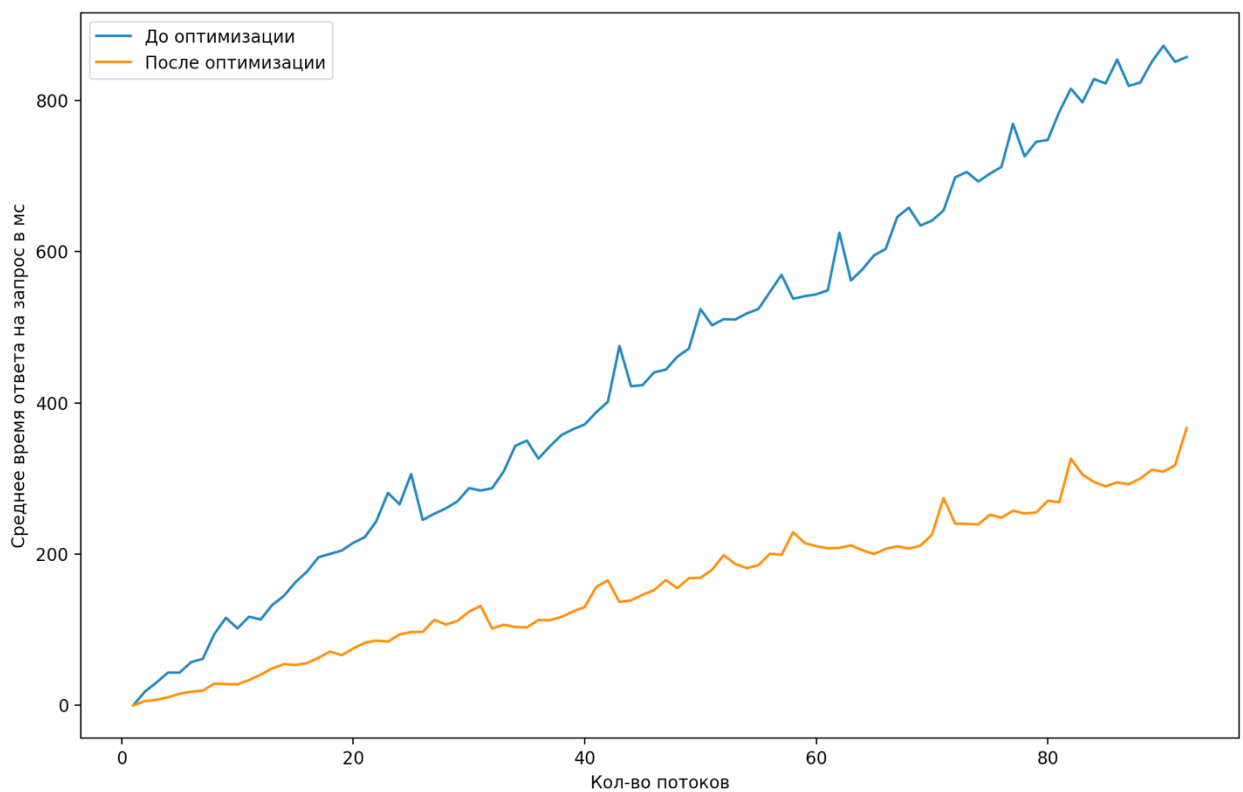


Рис. 4.3.11. График зависимости времени выполнения запросов от количества потоков на большой бд до оптимизации запросов и после (index+Prepare) при 100 запросах для 1 потока

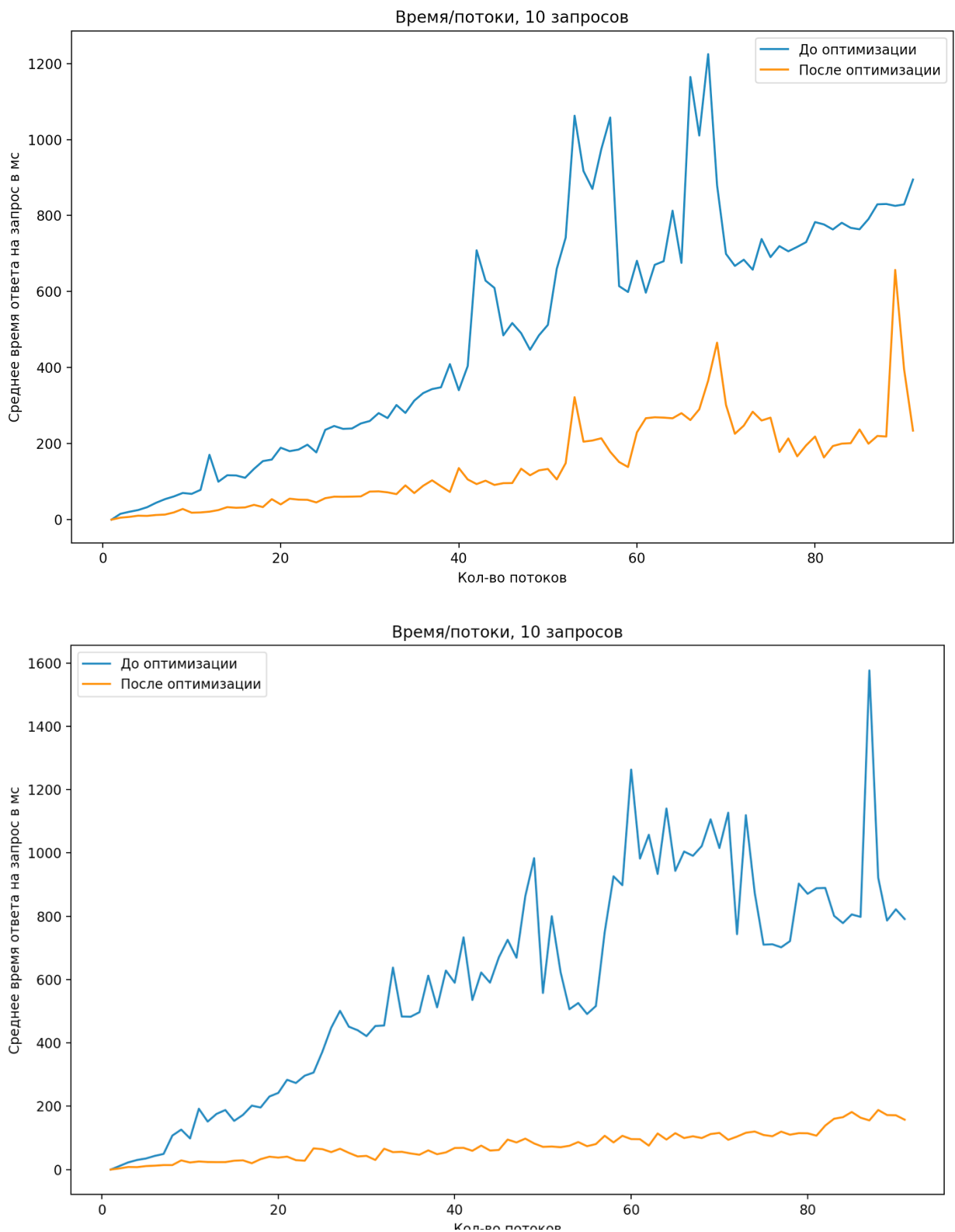


Рис. 4.3.12. График зависимости времени выполнения запросов от количества потоков на большой бд до оптимизации запросов и после (index+Prepare) при 10 запросах для 1 потока

Explain Analyze запроса 1 на большой базе до и после оптимизации:

```
Unique (actual time=34.250..34.265 rows=1 loops=1)
-> GroupAggregate (actual time=34.247..34.258 rows=6 loops=1)
"   Group Key: p.id, cap.name, m.person_id, m.enemy_id"
   Filter: ((count(((m.enemy_id = p.id) OR NULL::boolean)) <> 0) AND ((count(((m.person_id =
p.id) OR NULL::boolean)) / count(((m.enemy_id = p.id) OR NULL::boolean))) < 1))
   Rows Removed by Filter: 10
-> Sort (actual time=34.214..34.216 rows=16 loops=1)
"   Sort Key: p.id, cap.name, m.person_id, m.enemy_id"
   Sort Method: quicksort  Memory: 26kB
-> Nested Loop (actual time=1.925..34.101 rows=16 loops=1)
   Join Filter: ((p.id = m.person_id) OR (p.id = m.enemy_id))
   Rows Removed by Join Filter: 99987
-> Nested Loop (actual time=1.834..8.468 rows=1 loops=1)
   -> Hash Join (actual time=1.779..8.413 rows=1 loops=1)
       Hash Cond: (p.user_id = ud.id)
       -> Seq Scan on person p (actual time=0.054..7.422 rows=9980 loops=1)
           Filter: (update_date <= (now() - '1 year'::interval))
           Rows Removed by Filter: 10026
       -> Hash (actual time=0.106..0.106 rows=1 loops=1)
           Buckets: 1024  Batches: 1  Memory Usage: 9kB
           -> Index Scan using user_data_nickname_key on user_data ud (actual
time=0.093..0.095 rows=1 loops=1)
               Index Cond: ((nickname)::text = 'redispel'::text)
           -> Index Scan using class_of_person_pkey on class_of_person cap (actual
time=0.015..0.015 rows=1 loops=1)
               Index Cond: (id = p.class_of_person_id)
       -> Seq Scan on meetup m (actual time=0.054..18.852 rows=100003 loops=1)
Planning Time: 1.561 ms
Execution Time: 34.434 ms
```

Индексы

```
Unique (actual time=0.119..0.129 rows=1 loops=1)
-> GroupAggregate (actual time=0.117..0.125 rows=6 loops=1)
"   Group Key: p.id, cap.name, m.person_id, m.enemy_id"
   Filter: ((count(((m.enemy_id = p.id) OR NULL::boolean)) <> 0) AND ((count(((m.person_id =
p.id) OR NULL::boolean)) / count(((m.enemy_id = p.id) OR NULL::boolean))) < 1))
   Rows Removed by Filter: 10
-> Sort (actual time=0.108..0.110 rows=16 loops=1)
"   Sort Key: p.id, cap.name, m.person_id, m.enemy_id"
   Sort Method: quicksort  Memory: 26kB
-> Nested Loop (actual time=0.067..0.090 rows=16 loops=1)
   -> Nested Loop (actual time=0.043..0.047 rows=1 loops=1)
       -> Nested Loop (actual time=0.037..0.040 rows=1 loops=1)
```

```

-> Index Scan using user_nickname_idx on user_data ud (actual
time=0.014..0.015 rows=1 loops=1)
    Index Cond: ((nickname)::text = 'redispel'::text)
-> Bitmap Heap Scan on person p (actual time=0.018..0.020 rows=1 loops=1)
    Recheck Cond: (user_id = ud.id)
    Filter: (update_date <= (now() - '1 year'::interval))
    Rows Removed by Filter: 1
    Heap Blocks: exact=2
-> Bitmap Index Scan on person_user_idx (actual time=0.008..0.008
rows=2 loops=1)
    Index Cond: (user_id = ud.id)
-> Index Scan using class_of_person_pkey on class_of_person cap (actual
time=0.005..0.005 rows=1 loops=1)
    Index Cond: (id = p.class_of_person_id)
-> Bitmap Heap Scan on meetup m (actual time=0.020..0.035 rows=16 loops=1)
    Recheck Cond: ((p.id = person_id) OR (p.id = enemy_id))
    Heap Blocks: exact=15
-> BitmapOr (actual time=0.013..0.013 rows=0 loops=1)
-> Bitmap Index Scan on meetup_person_idx (actual time=0.008..0.008
rows=10 loops=1)
    Index Cond: (person_id = p.id)
-> Bitmap Index Scan on meetup_enemy_idx (actual time=0.004..0.004
rows=6 loops=1)
    Index Cond: (enemy_id = p.id)
Planning Time: 1.394 ms
Execution Time: 0.289 ms

C Prepare
Unique (cost=25.79..26.25 rows=3 width=90) (actual time=0.118..0.128 rows=1 loops=1)
-> GroupAggregate (cost=25.79..26.24 rows=3 width=90) (actual time=0.118..0.127 rows=6
loops=1)
    "    Group Key: p.id, cap.name, m.person_id, m.enemy_id"
    Filter: (((count(((m.enemy_id = p.id) OR NULL)::boolean)) <> 0) AND ((count(((m.person_id =
p.id) OR NULL)::boolean)) / count(((m.enemy_id = p.id) OR NULL)::boolean))) < 1))
    Rows Removed by Filter: 10
-> Sort (cost=25.79..25.81 rows=10 width=90) (actual time=0.109..0.109 rows=16 loops=1)
    "    Sort Key: p.id, cap.name, m.person_id, m.enemy_id"
    Sort Method: quicksort  Memory: 26kB
-> Nested Loop (cost=5.89..25.62 rows=10 width=90) (actual time=0.069..0.089
rows=16 loops=1)
-> Nested Loop (cost=4.74..20.17 rows=1 width=82) (actual time=0.049..0.052
rows=1 loops=1)
-> Nested Loop (cost=4.59..20.00 rows=1 width=8) (actual time=0.044..0.047
rows=1 loops=1)
-> Index Scan using user_nickname_idx on user_data ud (cost=0.29..8.30
rows=1 width=4) (actual time=0.019..0.020 rows=1 loops=1)
    Index Cond: ((nickname)::text = ($1)::text)

```

```

-> Bitmap Heap Scan on person p (cost=4.30..11.68 rows=1 width=12)
(actual time=0.018..0.020 rows=1 loops=1)
    Recheck Cond: (user_id = ud.id)
    Filter: (update_date <= (now() - '1 year'::interval))
    Rows Removed by Filter: 1
    Heap Blocks: exact=2
-> Bitmap Index Scan on person_user_idx (cost=0.00..4.30 rows=2
width=0) (actual time=0.009..0.009 rows=2 loops=1)
    Index Cond: (user_id = ud.id)
-> Index Scan using class_of_person_pkey on class_of_person cap
(cost=0.15..0.17 rows=1 width=82) (actual time=0.003..0.003 rows=1 loops=1)
    Index Cond: (id = p.class_of_person_id)
-> Bitmap Heap Scan on meetup m (cost=1.15..5.33 rows=12 width=8) (actual
time=0.016..0.030 rows=16 loops=1)
    Recheck Cond: ((p.id = person_id) OR (p.id = enemy_id))
    Heap Blocks: exact=15
-> BitmapOr (cost=1.15..1.15 rows=12 width=0) (actual time=0.012..0.012
rows=0 loops=1)
    -> Bitmap Index Scan on meetup_person_idx (cost=0.00..0.58 rows=7
width=0) (actual time=0.007..0.007 rows=10 loops=1)
        Index Cond: (person_id = p.id)
    -> Bitmap Index Scan on meetup_enemy_idx (cost=0.00..0.57 rows=5
width=0) (actual time=0.004..0.004 rows=6 loops=1)
        Index Cond: (enemy_id = p.id)
Planning Time: 0.018 ms
Execution Time: 0.235 ms

```

Explain Analyze запроса 2 на большой базе до и после оптимизации:

```

Sort (actual time=74.516..75.202 rows=4334 loops=1)
  Sort Key: (count(m.*)) DESC
  Sort Method: quicksort  Memory: 490kB
-> HashAggregate (actual time=71.331..72.917 rows=4334 loops=1)
  Group Key: ud.nickname
-> Hash Join (actual time=15.908..63.992 rows=8446 loops=1)
  Hash Cond: (p.user_id = ud.id)
-> Hash Join (actual time=10.306..51.862 rows=8446 loops=1)
  Hash Cond: (m.person_id = p.id)
-> Seq Scan on meetup m (actual time=0.092..34.691 rows=8446 loops=1)
  Filter: ((result = 'loose'::meetup_result) AND (meetup_date < now()) AND
(meetup_date >= (now() - '1 year'::interval)))
  Rows Removed by Filter: 91557
-> Hash (actual time=9.822..9.823 rows=20006 loops=1)
  Buckets: 32768  Batches: 1  Memory Usage: 1030kB
-> Seq Scan on person p (actual time=0.030..4.043 rows=20006 loops=1)
-> Hash (actual time=5.514..5.514 rows=9876 loops=1)
  Buckets: 16384  Batches: 1  Memory Usage: 588kB
-> Seq Scan on user_data ud (actual time=0.035..2.743 rows=9876 loops=1)

```

Planning Time: 0.813 ms
Execution Time: 76.178 ms

Индексы

Sort (actual time=48.051..48.929 rows=4312 loops=1)

Sort Key: (count(m.*)) DESC

Sort Method: quicksort Memory: 489kB

-> HashAggregate (actual time=45.541..46.747 rows=4312 loops=1)

Group Key: ud.nickname

-> Hash Join (actual time=18.041..39.642 rows=8369 loops=1)

Hash Cond: (p.user_id = ud.id)

-> Hash Join (actual time=14.474..30.965 rows=8369 loops=1)

Hash Cond: (m.person_id = p.id)

-> Bitmap Heap Scan on meetup m (actual time=6.942..18.046 rows=8369 loops=1)

Recheck Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date

< now()))

Filter: (result = 'loose'::meetup_result)

Rows Removed by Filter: 16698

Heap Blocks: exact=736

-> Bitmap Index Scan on meetup_date_idx (actual time=6.800..6.800

rows=25067 loops=1)

Index Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date

< now()))

-> Hash (actual time=7.322..7.322 rows=20006 loops=1)

Buckets: 32768 Batches: 1 Memory Usage: 1030kB

-> Seq Scan on person p (actual time=0.028..3.153 rows=20006 loops=1)

-> Hash (actual time=3.521..3.521 rows=9876 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 588kB

-> Seq Scan on user_data ud (actual time=0.011..1.727 rows=9876 loops=1)

Planning Time: 0.542 ms

Execution Time: 50.023 ms

C Prepare

Sort (cost=3679.55..3700.73 rows=8471 width=18) (actual time=45.759..46.351 rows=4312 loops=1)

Sort Key: (count(m.*)) DESC

Sort Method: quicksort Memory: 489kB

-> HashAggregate (cost=3042.18..3126.89 rows=8471 width=18) (actual time=43.092..44.501 rows=4312 loops=1)

Group Key: ud.nickname

-> Hash Join (cost=1584.02..2999.83 rows=8471 width=58) (actual time=18.357..38.486 rows=8369 loops=1)

Hash Cond: (p.user_id = ud.id)

-> Hash Join (cost=1156.81..2550.37 rows=8471 width=52) (actual time=14.267..30.063 rows=8369 loops=1)

Hash Cond: (m.person_id = p.id)


```

-> Bitmap Heap Scan on meetup m (cost=540.67..1912.00 rows=8471 width=52)
(actual time=5.301..16.314 rows=8369 loops=1)
    Recheck Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date
< now()))
    Filter: (result = $1)
    Rows Removed by Filter: 16698
    Heap Blocks: exact=736
    -> Bitmap Index Scan on meetup_date_idx (cost=0.00..538.56 rows=25413
width=0) (actual time=5.177..5.177 rows=25067 loops=1)
        Index Cond: ((meetup_date >= (now() - '1 year'::interval)) AND (meetup_date
< now()))
    -> Hash (cost=366.06..366.06 rows=20006 width=8) (actual time=8.820..8.821
rows=20006 loops=1)
        Buckets: 32768 Batches: 1 Memory Usage: 1030kB
    -> Seq Scan on person p (cost=0.00..366.06 rows=20006 width=8) (actual
time=0.015..3.685 rows=20006 loops=1)
    -> Hash (cost=303.76..303.76 rows=9876 width=14) (actual time=4.069..4.070
rows=9876 loops=1)
        Buckets: 16384 Batches: 1 Memory Usage: 588kB
    -> Seq Scan on user_data ud (cost=0.00..303.76 rows=9876 width=14) (actual
time=0.013..2.064 rows=9876 loops=1)
Planning Time: 0.016 ms
Execution Time: 47.573 ms

```

Explain Analyze запроса 3 на большой базе до и после оптимизации:

```

GroupAggregate (actual time=33.976..33.992 rows=6 loops=1)
  Group Key: i.name
  -> Sort (actual time=33.945..33.948 rows=6 loops=1)
      Sort Key: i.name
      Sort Method: quicksort Memory: 25kB
      -> Nested Loop (actual time=11.263..33.870 rows=6 loops=1)
          -> Index Only Scan using person_pkey on person p (actual time=0.009..0.019 rows=1
loops=1)
              Index Cond: (id = 120007)
              Heap Fetches: 1
          -> Nested Loop (actual time=11.250..33.835 rows=6 loops=1)
              -> Hash Join (actual time=11.197..33.549 rows=6 loops=1)
                  Hash Cond: (ipi.inventory_person_id = ip.id)
                  -> Seq Scan on inventory_person_items ipi (actual time=0.008..13.047
rows=146194 loops=1)
                      -> Hash (actual time=2.033..2.034 rows=1 loops=1)
                          Buckets: 1024 Batches: 1 Memory Usage: 9kB
                      -> Seq Scan on inventory_person ip (actual time=0.013..2.028 rows=1
loops=1)
                          Filter: (person_id = 120007)
                          Rows Removed by Filter: 22005
                  -> Index Scan using item_pkey on item i (actual time=0.030..0.030 rows=1 loops=6)

```

Index Cond: (id = ipi.item_id)

Planning Time: 0.448 ms

Execution Time: 34.128 ms

Индексы

GroupAggregate (actual time=0.069..0.071 rows=6 loops=1)

Group Key: i.name

-> Sort (actual time=0.063..0.063 rows=6 loops=1)

Sort Key: i.name

Sort Method: quicksort Memory: 25kB

-> Nested Loop (actual time=0.038..0.054 rows=6 loops=1)

-> Nested Loop (actual time=0.032..0.040 rows=6 loops=1)

-> Nested Loop (actual time=0.014..0.015 rows=1 loops=1)

-> Index Scan using inventory_person_person_idx on inventory_person ip

(actual time=0.007..0.009 rows=1 loops=1)

Index Cond: (person_id = 120007)

-> Index Only Scan using person_pkey on person p (actual time=0.005..0.005 rows=1 loops=1)

Index Cond: (id = 120007)

Heap Fetches: 1

-> Bitmap Heap Scan on inventory_person_items ipi (actual time=0.013..0.019 rows=6 loops=1)

Recheck Cond: (inventory_person_id = ip.id)

Heap Blocks: exact=6

-> Bitmap Index Scan on inventory_person_item_inventory_person_item_idx (actual time=0.007..0.007 rows=6 loops=1)

Index Cond: (inventory_person_id = ip.id)

-> Index Scan using item_pkey on item i (actual time=0.002..0.002 rows=1 loops=6)

Index Cond: (id = ipi.item_id)

Planning Time: 0.458 ms

Execution Time: 0.125 ms

C Prepare

GroupAggregate (cost=48.93..49.05 rows=7 width=18) (actual time=0.073..0.076 rows=6 loops=1)

Group Key: i.name

-> Sort (cost=48.93..48.94 rows=7 width=14) (actual time=0.068..0.069 rows=6 loops=1)

Sort Key: i.name

Sort Method: quicksort Memory: 25kB

-> Nested Loop (cost=5.18..48.83 rows=7 width=14) (actual time=0.044..0.058 rows=6 loops=1)

-> Nested Loop (cost=5.05..47.76 rows=7 width=8) (actual time=0.040..0.047 rows=6 loops=1)

-> Nested Loop (cost=0.57..16.62 rows=1 width=4) (actual time=0.023..0.024 rows=1 loops=1)

-> Index Scan using inventory_person_person_idx on inventory_person ip (cost=0.29..8.30 rows=1 width=8) (actual time=0.015..0.016 rows=1 loops=1)

```

      Index Cond: (person_id = $1)
    -> Index Only Scan using person_pkey on person p (cost=0.29..8.30 rows=1
width=4) (actual time=0.006..0.007 rows=1 loops=1)
      Index Cond: (id = $1)
      Heap Fetches: 1
    -> Bitmap Heap Scan on inventory_person_items ipi (cost=4.47..31.07 rows=7
width=12) (actual time=0.013..0.019 rows=6 loops=1)
      Recheck Cond: (inventory_person_id = ip.id)
      Heap Blocks: exact=6
    -> Bitmap Index Scan on inventory_person_item_inventory_person_item_idx
(cost=0.00..4.47 rows=7 width=0) (actual time=0.009..0.009 rows=6 loops=1)
      Index Cond: (inventory_person_id = ip.id)
    -> Index Scan using item_pkey on item i (cost=0.14..0.15 rows=1 width=14) (actual
time=0.001..0.001 rows=1 loops=6)
      Index Cond: (id = ipi.item_id)
Planning Time: 0.021 ms
Execution Time: 0.138 ms

```

Explain Analyze запроса 4 на большой базе до и после оптимизации:

```

GroupAggregate (actual time=67.829..67.868 rows=20 loops=1)
  Group Key: ps.person_id
  Filter: (count((ps.equiped OR NULL)::boolean)) = 2)
  Rows Removed by Filter: 46
  -> Sort (actual time=67.814..67.825 rows=138 loops=1)
    Sort Key: ps.person_id
    Sort Method: quicksort Memory: 31kB
    -> Nested Loop (actual time=29.680..67.648 rows=138 loops=1)
      -> Hash Join (actual time=29.610..54.379 rows=3512 loops=1)
        Hash Cond: (ipi.inventory_person_id = ip.id)
      -> Seq Scan on inventory_person_items ipi (actual time=0.030..18.952 rows=10523
loops=1)
        Filter: (item_id = 11)
        Rows Removed by Filter: 135671
      -> Hash (actual time=29.524..29.524 rows=7310 loops=1)
        Buckets: 8192 (originally 1024) Batches: 1 (originally 1) Memory Usage: 407kB
      -> Hash Join (actual time=14.435..26.444 rows=7310 loops=1)
        Hash Cond: (ip.person_id = ps.person_id)
      -> Seq Scan on inventory_person ip (actual time=0.016..3.309 rows=22006
loops=1)
        Filter: (item_id = 11)
        Rows Removed by Filter: 135671
      -> Hash (actual time=14.391..14.391 rows=6634 loops=1)
        Buckets: 8192 (originally 1024) Batches: 1 (originally 1) Memory Usage:
304kB
      -> Hash Join (actual time=0.102..12.362 rows=6634 loops=1)
        Hash Cond: (ps.skill_id = s.id)
      -> Seq Scan on person_skill ps (actual time=0.017..5.962 rows=40012
loops=1)
        Filter: (item_id = 11)
        Rows Removed by Filter: 135671
      -> Hash (actual time=0.071..0.071 rows=2 loops=1)

```

Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Hash Join (actual time=0.064..0.069 rows=2 loops=1)
 Hash Cond: (s.class_of_person_id = cop.id)
 -> Seq Scan on skill s (actual time=0.010..0.014 rows=12 loops=1)
 -> Hash (actual time=0.033..0.033 rows=1 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Index Scan using class_of_person_name_key on class_of_person cop (actual time=0.021..0.024 rows=1 loops=1)
 Index Cond: ((name)::text = 'Archer'::text)
 -> Index Scan using person_pkey on person p (actual time=0.003..0.003 rows=0 loops=3512)
 Index Cond: (id = ps.person_id)
 Filter: (health = 300)
 Rows Removed by Filter: 1
 Planning Time: 1.646 ms
 Execution Time: 68.130 ms

Индексы

GroupAggregate (actual time=25.469..25.568 rows=20 loops=1)
 Group Key: ps.person_id
 Filter: (count((ps.equiped OR NULL)::boolean)) = 2)
 Rows Removed by Filter: 46
 -> Sort (actual time=25.429..25.447 rows=138 loops=1)
 Sort Key: ps.person_id
 Sort Method: quicksort Memory: 31kB
 -> Nested Loop (actual time=0.161..25.310 rows=138 loops=1)
 -> Nested Loop (actual time=0.139..23.511 rows=310 loops=1)
 -> Nested Loop (actual time=0.130..21.910 rows=288 loops=1)
 -> Nested Loop (actual time=0.114..5.163 rows=6634 loops=1)
 -> Hash Join (actual time=0.085..0.118 rows=2 loops=1)
 Hash Cond: (s.class_of_person_id = cop.id)
 -> Seq Scan on skill s (actual time=0.035..0.051 rows=12 loops=1)
 -> Hash (actual time=0.030..0.030 rows=1 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Index Scan using class_of_person_name_key on class_of_person cop (actual time=0.020..0.022 rows=1 loops=1)
 Index Cond: ((name)::text = 'Archer'::text)
 -> Index Scan using person_skill_skill_idx on person_skill ps (actual time=0.018..1.991 rows=3317 loops=2)
 Index Cond: (skill_id = s.id)
 -> Index Scan using person_pkey on person p (actual time=0.002..0.002 rows=0 loops=6634)
 Index Cond: (id = ps.person_id)
 Filter: (health = 300)
 Rows Removed by Filter: 1
 -> Index Scan using inventory_person_person_idx on inventory_person ip (actual time=0.004..0.005 rows=1 loops=288)

Index Cond: (person_id = ps.person_id)
 -> Index Only Scan using inventory_person_item_inventory_person_item_idx on
 inventory_person_items ipi (actual time=0.005..0.005 rows=0 loops=310)
 Index Cond: ((inventory_person_id = ip.id) AND (item_id = 11))
 Heap Fetches: 0
 Planning Time: 2.276 ms
 Execution Time: 25.794 ms

C Prepare
 GroupAggregate (cost=147.00..147.02 rows=1 width=4) (actual time=23.106..23.156 rows=20
 loops=1)
 Group Key: ps.person_id
 Filter: (count((ps.equiped OR NULL)::boolean)) = 2)
 Rows Removed by Filter: 46
 -> Sort (cost=147.00..147.00 rows=1 width=5) (actual time=23.081..23.096 rows=138
 loops=1)
 Sort Key: ps.person_id
 Sort Method: quicksort Memory: 31kB
 -> Nested Loop (cost=9.46..146.99 rows=1 width=5) (actual time=0.241..22.958 rows=138
 loops=1)
 -> Nested Loop (cost=9.04..145.42 rows=3 width=9) (actual time=0.208..21.211
 rows=310 loops=1)
 -> Nested Loop (cost=8.76..144.42 rows=3 width=9) (actual time=0.193..19.609
 rows=288 loops=1)
 -> Nested Loop (cost=8.47..119.42 rows=70 width=5) (actual time=0.170..4.792
 rows=6634 loops=1)
 -> Hash Join (cost=8.18..25.01 rows=1 width=4) (actual time=0.140..0.169
 rows=2 loops=1)
 Hash Cond: (s.class_of_person_id = cop.id)
 -> Seq Scan on skill s (cost=0.00..15.40 rows=540 width=8) (actual
 time=0.058..0.072 rows=12 loops=1)
 -> Hash (cost=8.17..8.17 rows=1 width=4) (actual time=0.058..0.059
 rows=1 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Index Scan using class_of_person_name_key on class_of_person
 cop (cost=0.15..8.17 rows=1 width=4) (actual time=0.047..0.049 rows=1 loops=1)
 Index Cond: ((name)::text = (\$1)::text)
 -> Index Scan using person_skill_skill_idx on person_skill ps
 (cost=0.29..61.07 rows=3334 width=9) (actual time=0.018..1.832 rows=3317 loops=2)
 Index Cond: (skill_id = s.id)
 -> Index Scan using person_pkey on person p (cost=0.29..0.36 rows=1 width=4)
 (actual time=0.002..0.002 rows=0 loops=6634)
 Index Cond: (id = ps.person_id)
 Filter: (health = 300)
 Rows Removed by Filter: 1
 -> Index Scan using inventory_person_person_idx on inventory_person ip
 (cost=0.29..0.32 rows=1 width=8) (actual time=0.004..0.005 rows=1 loops=288)
 Index Cond: (person_id = ps.person_id)

```
-> Index Only Scan using inventory_person_item_inventory_person_item_idx on
inventory_person_items ipi (cost=0.42..0.51 rows=1 width=4) (actual time=0.005..0.005
rows=0 loops=310)
    Index Cond: ((inventory_person_id = ip.id) AND (item_id = 11))
    Heap Fetches: 0
Planning Time: 0.065 ms
Execution Time: 23.427 ms
```

Explain Analyze запроса 5 на большой базе до и после оптимизации:

```
Sort (actual time=83.488..84.021 rows=5363 loops=1)
  Sort Key: (sum(ipi.amount)) DESC
  Sort Method: quicksort Memory: 560kB
-> Finalize HashAggregate (actual time=80.399..81.951 rows=5363 loops=1)
  Group Key: ud.nickname
-> Gather (actual time=72.475..76.296 rows=6397 loops=1)
  Workers Planned: 1
  Workers Launched: 1
-> Partial HashAggregate (actual time=65.517..66.962 rows=3198 loops=2)
  Group Key: ud.nickname
-> Hash Join (actual time=31.896..61.657 rows=4018 loops=2)
  Hash Cond: (p.user_id = ud.id)
-> Hash Join (actual time=24.266..50.436 rows=4452 loops=2)
  Hash Cond: (ip.person_id = p.id)
-> Hash Join (actual time=11.619..33.966 rows=4452 loops=2)
  Hash Cond: (ipi.inventory_person_id = ip.id)
-> Parallel Seq Scan on inventory_person_items ipi (actual
time=0.043..18.288 rows=4452 loops=2)
  Filter: ((item_id = 11) AND (update_date < (now() +
'00:00:00'::interval)) AND (update_date >= (now() - '1 year'::interval)) AND (add_date < (now()
+ '00:00:00'::interval)) AND (add_date >= (now() - '1 year'::interval)))
  Rows Removed by Filter: 68646
-> Hash (actual time=11.322..11.322 rows=22006 loops=2)
  Buckets: 32768 Batches: 1 Memory Usage: 1116kB
-> Seq Scan on inventory_person ip (actual time=0.120..4.069
rows=22006 loops=2)
-> Hash (actual time=12.293..12.293 rows=20006 loops=2)
  Buckets: 32768 Batches: 1 Memory Usage: 1030kB
-> Seq Scan on person p (actual time=0.151..4.981 rows=20006 loops=2)
-> Hash (actual time=7.407..7.407 rows=9876 loops=2)
  Buckets: 16384 Batches: 1 Memory Usage: 588kB
-> Seq Scan on user_data ud (actual time=0.147..3.671 rows=9876 loops=2)
Planning Time: 0.871 ms
Execution Time: 85.592 ms
```

Индексы

```
Sort (actual time=71.166..71.914 rows=5363 loops=1)
```

Sort Key: (sum(ipi.amount)) DESC
 Sort Method: quicksort Memory: 560kB
 -> HashAggregate (actual time=68.532..69.860 rows=5363 loops=1)
 Group Key: ud.nickname
 -> Hash Join (actual time=27.922..62.433 rows=8037 loops=1)
 Hash Cond: (p.user_id = ud.id)
 -> Hash Join (actual time=23.018..51.717 rows=8903 loops=1)
 Hash Cond: (ip.person_id = p.id)
 -> Hash Join (actual time=12.043..34.014 rows=8903 loops=1)
 Hash Cond: (ipi.inventory_person_id = ip.id)
 -> Bitmap Heap Scan on inventory_person_items ipi (actual time=0.859..15.771 rows=8903 loops=1)
 Recheck Cond: (item_id = 11)
 Filter: ((update_date < (now() - '00:00:00'::interval)) AND (update_date >= (now() - '1 year'::interval)) AND (add_date < (now() - '00:00:00'::interval)) AND (add_date >= (now() - '1 year'::interval)))
 Rows Removed by Filter: 1620
 Heap Blocks: exact=1380
 -> Bitmap Index Scan on inventory_person_item_idx (actual time=0.671..0.671 rows=10523 loops=1)
 Index Cond: (item_id = 11)
 -> Hash (actual time=10.904..10.905 rows=22006 loops=1)
 Buckets: 32768 Batches: 1 Memory Usage: 1116kB
 -> Seq Scan on inventory_person ip (actual time=0.059..3.898 rows=22006 loops=1)
 -> Hash (actual time=10.736..10.736 rows=20006 loops=1)
 Buckets: 32768 Batches: 1 Memory Usage: 1030kB
 -> Seq Scan on person p (actual time=0.042..4.764 rows=20006 loops=1)
 -> Hash (actual time=4.815..4.815 rows=9876 loops=1)
 Buckets: 16384 Batches: 1 Memory Usage: 588kB
 -> Seq Scan on user_data ud (actual time=0.039..2.489 rows=9876 loops=1)
 Planning Time: 1.440 ms
 Execution Time: 73.224 ms

Work_mem=500Mb
 Sort (actual time=76.443..77.607 rows=5363 loops=1)
 Sort Key: (sum(ipi.amount)) DESC
 Sort Method: quicksort Memory: 560kB
 -> HashAggregate (actual time=72.138..74.448 rows=5363 loops=1)
 Group Key: ud.nickname
 -> Hash Join (actual time=28.427..65.623 rows=8037 loops=1)
 Hash Cond: (p.user_id = ud.id)
 -> Hash Join (actual time=22.243..53.073 rows=8903 loops=1)
 Hash Cond: (ip.person_id = p.id)
 -> Hash Join (actual time=10.571..34.381 rows=8903 loops=1)
 Hash Cond: (ipi.inventory_person_id = ip.id)
 -> Bitmap Heap Scan on inventory_person_items ipi (actual time=1.053..17.312 rows=8903 loops=1)

Recheck Cond: (item_id = 11)
 Filter: ((update_date < (now() - '00:00:00'::interval)) AND (update_date >= (now() - '1 year'::interval)) AND (add_date < (now() - '00:00:00'::interval)) AND (add_date >= (now() - '1 year'::interval)))
 Rows Removed by Filter: 1620
 Heap Blocks: exact=1380
 -> Bitmap Index Scan on inventory_person_item_idx (actual time=0.882..0.882 rows=10523 loops=1)
 Index Cond: (item_id = 11)
 -> Hash (actual time=9.272..9.273 rows=22006 loops=1)
 Buckets: 32768 Batches: 1 Memory Usage: 1116kB
 -> Seq Scan on inventory_person ip (actual time=0.034..3.398 rows=22006 loops=1)
 -> Hash (actual time=11.438..11.438 rows=20006 loops=1)
 Buckets: 32768 Batches: 1 Memory Usage: 1030kB
 -> Seq Scan on person p (actual time=0.035..4.625 rows=20006 loops=1)
 -> Hash (actual time=6.142..6.143 rows=9876 loops=1)
 Buckets: 16384 Batches: 1 Memory Usage: 588kB
 -> Seq Scan on user_data ud (actual time=0.065..3.024 rows=9876 loops=1)
 Planning Time: 2.041 ms
 Execution Time: 80.837 ms

C Prepare

Sort (cost=18.50..18.51 rows=1 width=18) (actual time=125.417..126.271 rows=5363 loops=1)
 Sort Key: (sum(ipi.amount)) DESC
 Sort Method: quicksort Memory: 560kB
 -> GroupAggregate (cost=18.47..18.49 rows=1 width=18) (actual time=120.487..124.070 rows=5363 loops=1)
 Group Key: ud.nickname
 -> Sort (cost=18.47..18.48 rows=1 width=14) (actual time=120.460..121.449 rows=8037 loops=1)
 Sort Key: ud.nickname
 Sort Method: quicksort Memory: 617kB
 -> Nested Loop (cost=1.28..18.46 rows=1 width=14) (actual time=0.043..114.375 rows=8037 loops=1)
 -> Nested Loop (cost=1.00..18.08 rows=1 width=8) (actual time=0.038..85.766 rows=8903 loops=1)
 -> Nested Loop (cost=0.71..17.69 rows=1 width=8) (actual time=0.032..51.241 rows=8903 loops=1)
 -> Index Scan using inventory_person_items_item_addupdate_date_idx on inventory_person_items ipi (cost=0.42..9.38 rows=1 width=8) (actual time=0.025..15.550 rows=8903 loops=1)
 Index Cond: ((item_id = 11) AND (add_date < \$1) AND (add_date >= (\$1 - '1 year'::interval)) AND (update_date < \$1) AND (update_date >= (\$1 - '1 year'::interval)))
 -> Index Scan using inventory_person_pkey on inventory_person ip (cost=0.29..8.30 rows=1 width=8) (actual time=0.003..0.003 rows=1 loops=8903)
 Index Cond: (id = ipi.inventory_person_id)


```
-> Index Scan using person_pkey on person p (cost=0.29..0.40 rows=1 width=8)
(actual time=0.003..0.003 rows=1 loops=8903)
      Index Cond: (id = ip.person_id)
-> Index Scan using user_data_pkey on user_data ud (cost=0.29..0.38 rows=1
width=14) (actual time=0.003..0.003 rows=1 loops=8903)
      Index Cond: (id = p.user_id)
Planning Time: 0.026 ms
Execution Time: 126.596 ms
```

4.4 Выводы

В ходе выполнения данной лабораторной работы было проведено ознакомление со структурой и инструментами, помогающими при оптимизации запросов пользователей к БД. Получились следующие результаты: prepare ускоряет работу БД, но не так сильно как indexes. Совмещение этих двух методов оптимизации позволяет существенно повысить быстродействие базы данных в сравнении с результатами, полученными при отсутствии всякой оптимизации. С увеличением количества потоков наблюдается явное увеличение затрат времени на запросы. Также прослеживается увеличение времени обработки запросов с увеличением количества запросов в единицу времени во всех проведенных экспериментах.

5. Листинги

Листинг 5.1. Скрипт изменения бд для лабораторной работы №1

```
-----ОБЪЕДИНЕНИЕ PERSON И ENEMY
ALTER TABLE Person ALTER COLUMN user_id SET DEFAULT NULL;
ALTER TABLE Enemy ADD COLUMN user_id INTEGER NULL DEFAULT NULL;
ALTER TABLE Person ADD is_enemy BOOLEAN NOT NULL DEFAULT 'f';
ALTER TABLE Person ALTER COLUMN is_enemy SET DEFAULT 't';
INSERT INTO Person(class_of_person_id, health, experience, user_id,
update_date)
SELECT class_of_person_id, health, experience, user_id, update_date FROM Enemy;

-----ОБЪЕДИНЕНИЕ INVENTORY_PERSON И
INVENTORY_ENEMY
INSERT INTO inventory_person(inventory_size) SELECT inventory_size FROM
inventory_enemy;
UPDATE inventory_person SET person_id = id;

-----ОБЪЕДИНЕНИЕ INVENTORY_PERSON_ITEMS И
INVENTORY_PERSON_ENEMY
ALTER TABLE inventory_enemy_items ADD COLUMN new_id INTEGER;
UPDATE inventory_enemy_items SET new_id=inventory_enemy_id+3;
```

```

INSERT INTO inventory_person_items(inventory_person_id, item_id, add_date,
is_deleted, amount, update_date)
SELECT new_id, item_id, add_date, is_deleted, amount, update_date FROM
inventory_enemy_items;

-----ИЗМЕНЕНИЕ MEETUP
ALTER TABLE meetup ADD COLUMN new_enemy_id INTEGER REFERENCES Person(id) ON
DELETE SET NULL;
UPDATE meetup SET new_enemy_id=enemy_id+3;
ALTER TABLE meetup DROP COLUMN enemy_id;
ALTER TABLE meetup RENAME COLUMN new_enemy_id TO enemy_id;

-----ДОБАВЛЕНИЕ PERSON_SKILL
CREATE TABLE person_skill (
person_id INTEGER REFERENCES person(id) ON DELETE CASCADE,
skill_id INTEGER REFERENCES skill(id) ON DELETE CASCADE,
equiped BOOL NOT NULL
);

insert into person_skill VALUES (1, 1, 'f'),
                                (1, 2, 'f'),
                                (2, 3, 'f'),
                                (2, 4, 'f'),
                                (3, 5, 'f'),
                                (3, 6, 'f');

-----ИЗМЕНЕНИЕ SKILL
ALTER TABLE skill DROP COLUMN equiped;

-----УДАЛЕНИЕ ЛИШНИХ ТАБЛИЦ
DROP TABLE inventory_enemy_items;
DROP TABLE inventory_enemy;
DROP TABLE enemy;
SELECT * FROM information_schema.tables
WHERE table_schema = 'public'

```

Листинг 5.2. Код генератора для лабораторной работы №2

```

import datetime
import psycopg2
import random
import argparse
import configparser

config = configparser.ConfigParser()
config.read('config.ini', encoding='utf-8')

'''
connection = psycopg2.connect(
    dbname=config.get("database_for_game", "dbname"),
    user=config.get("postgres", "user"),
    password=config.get("6559", "password"))
'''

connection = psycopg2.connect(dbname='database_for_game_big',
                             user='postgres',
                             password='6559',
                             host='localhost')

cursor = connection.cursor()

letters = 'qwertyuiopasdfghjklzxcvbnm'

```

```

today = datetime.datetime.today()

item_ids = ()
inventory_person_ids = ()
class_of_person_ids = ()
user_data_ids = ()
person_ids = ()

item_array = []
class_of_person_array = []
skill_array = []
user_data_array = []
person_array = []
inventory_person_array = []
meetup_array = []
person_skill_array = []
inventory_person_items_update_array = []
inventory_person_items_insert_array = []
inventory_person_items_update_tuple = []

items = open("input/items.txt", "r").readlines()
classes = open("input/classes.txt", "r").readlines()
skills = open("input/skills.txt", "r").readlines()
nicknames = open("input/nicknames.txt", "r").readlines()
email_domains = open("input/email_domains.txt", "r").readlines()

def add_on_update(inventory_person_id, item_id, current_item):
    amount_item = int(current_item[4]) + 1
    update_date = random_date(current_item[5], today)
    inventory_person_items_update_array.append([amount_item,
                                                update_date,
                                                inventory_person_id[0],
                                                item_id[0]])

def amount_of_insert_and_updates_for_person(inventory_person_id, insert_list,
update_list):
    update_items_dictionary = {}
    insert_items = 0
    for element in update_list:
        if element[2] == inventory_person_id:
            if update_items_dictionary.get(element[3]) is not None:
                if update_items_dictionary[element[3]] < element[0]:
                    update_items_dictionary[element[3]] = element[0]
            else:
                update_items_dictionary[element[3]] = element[0]

    for element in insert_list:
        if element[0] == inventory_person_id and
update_items_dictionary.get(element[1]) is None:
            insert_items += 1

    total_amount = insert_items + sum(update_items_dictionary.values())
    return total_amount

def list_to_tuple(update_list):
    list_temp = []
    for element in update_list:
        list_temp.append(tuple(element))
    return tuple(list_temp)

```

```

def random_string(length):
    rnd_str = ''
    for i in range(int(length)):
        rnd_str += random.choice(letters)
    return rnd_str

def random_date(start_date: object, end_date: object) -> object:
    date_delta = end_date - start_date
    days_delta = date_delta.days
    if days_delta == 0:
        return today
    else:
        days_random = random.randrange(days_delta)
        rnd_date = start_date + datetime.timedelta(days=days_random)
        return rnd_date

def generate_item():
    global item_ids
    cursor.execute('SELECT name FROM item')
    item_names = cursor.fetchall()
    for element in items:
        element = element.replace("\n", "")
        if element not in item_names:
            description = random_string(random.randint(5, 15))
            item_array.append((element, description))

    if item_array:
        temp_list = ','.join(['%s'] * len(item_array))
        cursor.execute(
            'INSERT INTO \"item\"(name, description) VALUES {} ON CONFLICT DO
NOTHING'.format(temp_list), item_array)
        cursor.execute('SELECT id FROM item')
        item_ids = cursor.fetchall()

def generate_class_of_person():
    global class_of_person_ids
    cursor.execute('SELECT name FROM class_of_person')
    class_name = cursor.fetchall()
    for element in classes:
        element = element.replace("\n", "")
        if element not in class_name:
            description = random_string(random.randint(5, 15))
            class_of_person_array.append((element, description))

    if class_of_person_array:
        temp_list = ','.join(['%s'] * len(class_of_person_array))
        cursor.execute('INSERT INTO \"class_of_person\"(name, description)
VALUES {} ON CONFLICT DO NOTHING'
            .format(temp_list), class_of_person_array)
        cursor.execute('SELECT id FROM class_of_person')
        class_of_person_ids = cursor.fetchall()
        generate_skill()

def generate_skill():
    cursor.execute('SELECT name FROM skill')
    skill_name = cursor.fetchall()
    for index, element in enumerate(skills):
        element = element.replace("\n", "")
        if element not in skill_name:

```

```

        description = random_string(random.randint(5, 15))
        cost = 50 * (index % 2 + 1)
        cursor.execute('SELECT id FROM class_of_person WHERE
name=\'{}\'.format(element.split()[0]))
        class_of_person_id = cursor.fetchone()[0]
        skill_array.append((element, description, cost,
class_of_person_id))

    if skill_array:
        temp_list = ','.join(['%s'] * len(skill_array))
        cursor.execute('INSERT INTO skill(name, description, cost,
class_of_person_id) VALUES {} '
                        'ON CONFLICT DO NOTHING'.format(temp_list),
skill_array)

def generate_person_skill():
    cursor.execute('SELECT person_id FROM person_skill')
    person_id_in_person_skill = cursor.fetchall()

    for element in person_ids:
        if element not in person_id_in_person_skill:
            person_id = element[0]
            cursor.execute('SELECT class_of_person_id FROM person WHERE
id=\'{}\'.format(person_id))
            person_class = cursor.fetchall()
            cursor.execute('SELECT id FROM skill WHERE
class_of_person_id={}'.format(person_class[0][0]))
            class_skills = cursor.fetchall()
            for skill_temp in class_skills:
                skill_id = skill_temp[0]
                is_equiped = bool(random.getrandbits(1))
                person_skill_array.append((person_id, skill_id, is_equiped))

    if person_skill_array:
        temp_list = ','.join(['%s'] * len(person_skill_array))
        cursor.execute('INSERT INTO person_skill VALUES {}'
                        .format(temp_list), person_skill_array)

def generate_user_data(amount):
    global user_data_ids
    for i in range(int(amount)):
        nickname = random.choice(nicknames)
        nickname = nickname.replace("\n", "")
        password = random_string(random.randint(10, 16))
        email_domain = random.choice(email_domains)
        email_domain = email_domain.replace("\n", "")
        email = random_string(random.randint(10, 20)) + email_domain
        user_data_array.append((email, password, nickname[:20]))

    if user_data_array:
        temp_list = ','.join(['%s'] * len(user_data_array))
        cursor.execute('INSERT INTO user_data(email, password, nickname)
VALUES {} ON CONFLICT DO NOTHING'
                        .format(temp_list), user_data_array)
        cursor.execute('SELECT id FROM user_data')
        user_data_ids = cursor.fetchall()

def generate_person(amount):
    global person_ids
    global user_data_ids
    for i in range(int(amount)):

```

```

class_of_person_id = random.choice(class_of_person_ids)[0]
health = 10 * random.randint(10, 30)
experience = 5 * random.randint(0, 19)
update_date = random_date(today - datetime.timedelta(days=2 * 365),
                           today)

is_enemy = True

cursor.execute('SELECT id FROM user_data')
user_data_ids = cursor.fetchall()

user_id = [None]

if i % 10 != 0:
    user_id = random.choice(user_data_ids)
    is_enemy = False
    person_array.append((class_of_person_id, health, experience,
user_id[0], update_date, is_enemy))

if person_array:
    temp_list = ','.join(['%s'] * len(person_array))
    cursor.execute(
        'INSERT INTO person (class_of_person_id, health, experience,
user_id, update_date, is_enemy) '
        'VALUES {}'.format(temp_list), person_array)

    cursor.execute('SELECT id FROM person')
    person_ids = cursor.fetchall()
    generate_inventory_person(amount, True)
    generate_person_skill()

def generate_inventory_person(amount, from_person):
    global person_ids
    global inventory_person_ids
    inventory_person_array.clear()
    cursor.execute('SELECT person_id FROM inventory_person')
    person_id_in_inventory_person = cursor.fetchall()

    if from_person:
        for element in person_ids:
            if element not in person_id_in_inventory_person:
                person_id = element[0]
                inventory_size = 5 * random.randint(2, 8)
                inventory_person_array.append((person_id, inventory_size))
    else:
        cursor.execute('SELECT id FROM person')
        person_ids = cursor.fetchall()
        for i in range(int(amount)):
            person_id = random.choice(person_ids)
            inventory_size = 5 * random.randint(2, 8)
            inventory_person_array.append((person_id, inventory_size))

    if inventory_person_array:
        temp_list = ','.join(['%s'] * len(inventory_person_array))
        cursor.execute('INSERT INTO inventory_person(person_id,
inventory_size) VALUES {}'.
                        .format(temp_list), inventory_person_array)
        cursor.execute('SELECT id FROM inventory_person')
        inventory_person_ids = cursor.fetchall()

def generate_meetup(amount):
    global person_ids
    global user_data_ids

```

```

        cursor.execute('SELECT id FROM user_data')
        user_data_ids = cursor.fetchall()
        cursor.execute('SELECT id FROM person')
        person_ids = cursor.fetchall()

        for i in range(int(amount)):
            person_id = None
            enemy_id = None
            persons_of_user = None

            while person_id is None:
                user_data_id = random.choice(user_data_ids)
                cursor.execute('SELECT id FROM person WHERE
user_id=\''{}\'''.format(user_data_id[0]))
                persons_of_user = cursor.fetchall()
                if persons_of_user:
                    person_id = random.choice(persons_of_user)

            while enemy_id is None:
                temp_enemy_id = random.choice(person_ids)
                if temp_enemy_id not in persons_of_user and temp_enemy_id !=
person_id:
                    enemy_id = temp_enemy_id

            cursor.execute('SELECT update_date FROM person WHERE
id=\''{}\'''.format(person_id[0]))
            person_update_date = cursor.fetchall()
            cursor.execute('SELECT update_date FROM person WHERE
id=\''{}\'''.format(enemy_id[0]))
            enemy_update_date = cursor.fetchall()
            if person_update_date <= enemy_update_date:
                meetup_date = person_update_date
            else:
                meetup_date = enemy_update_date

            result = random.choice(('win', 'loose', 'draw'))
            meetup_array.append((person_id[0], result, meetup_date[0][0],
enemy_id[0]))

            if meetup_array:
                temp_list = ','.join(['%s'] * len(meetup_array))
                cursor.execute('INSERT INTO meetup (person_id, result, meetup_date,
enemy_id) VALUES {}'.format(temp_list), meetup_array)

def generate_inventory_person_items(amount):
    global inventory_person_ids
    cursor.execute('SELECT id FROM inventory_person')
    inventory_person_ids = cursor.fetchall()

    for i in range(int(amount)):
        inventory_person_id = random.choice(inventory_person_ids)
        item_id = random.choice(item_ids)

        # check plenum
        cursor.execute('SELECT * FROM inventory_person WHERE
id=\''{}\'''.format(inventory_person_id[0]))
        inventory_person_tuple = cursor.fetchone()
        cursor.execute('SELECT amount FROM inventory_person_items WHERE
inventory_person_id=\''{}\'''.format(inventory_person_id[0]))
        all_amount = cursor.fetchall()

```

```

        current_amount_of_items = sum(element[0] for element in all_amount) +
\
        amount_of_insert_and_updates_for_person(inventory_person_id[0],
inventory_person_items_insert_array,
inventory_person_items_update_array)

        if current_amount_of_items < inventory_person_tuple[2]:
            cursor.execute('SELECT * FROM inventory_person_items WHERE
inventory_person_id=\'{}\'' AND item_id=\'{}\'''.
                format(inventory_person_id[0], item_id[0]))
            current_item = cursor.fetchone()

            cursor.execute('SELECT update_date FROM person WHERE
id=\'{}\'''.format(inventory_person_tuple[1]))
            person_update_date = cursor.fetchone()

            is_deleted = False
            is_in_insert_array = False

            for element in inventory_person_items_insert_array:
                if inventory_person_id[0] == element[0] and item_id[0] ==
element[1]:
                    is_in_insert_array = True

            if (current_item and not bool(current_item[3])) or
is_in_insert_array:
                # если такой айтем уже в инвентаре игрока - апдейтим его
                if not current_item:
                    for index, element in
enumerate(inventory_person_items_insert_array):
                        if inventory_person_id[0] == element[0] and
item_id[0] == element[1]:
                            current_item = (inventory_person_id[0],
                                item_id[0],
                                element[2],
                                is_deleted,
                                1,
                                element[2])

                if inventory_person_items_update_array:
                    flag = False
                    # перебираем все кортежи на апдейт и если наш есть -
amount+1
                    for index, element in
enumerate(inventory_person_items_update_array):
                        if inventory_person_id[0] == element[2] and
item_id[0] == element[3]:
                            flag = True
                            inventory_person_items_update_array[index][0] +=
1
                    # если кортежа на апдейт ещё нет в списке, но список не
пуст - добавляем этот кортеж
                    if not flag:
                        add_on_update(inventory_person_id, item_id,
current_item)
                    # если список пустой - добавляем кортеж на апдейт
                    else:
                        add_on_update(inventory_person_id, item_id, current_item)

                # если нету такого айтема в инвентаре у игрока - добавляем его
туда
            else:

```



```

        add_date = random_date(person_update_date[0], today)
        amount_item = 1
        update_date = add_date

inventory_person_items_insert_array.append((inventory_person_id[0],
                                             item_id[0],
                                             add_date,
                                             is_deleted,
                                             amount_item,
                                             update_date))

    if inventory_person_items_insert_array:
        temp_list = ','.join(['%s'] *
len(inventory_person_items_insert_array))
        cursor.execute(
            'INSERT INTO inventory_person_items VALUES {}'.format(temp_list),
inventory_person_items_insert_array)

    if inventory_person_items_update_array:
        temp_list = ','.join(['%s'] *
len(inventory_person_items_update_array))
        cursor.execute('UPDATE inventory_person_items AS t ' \
            'SET amount = c.amount, update_date = c.update_date ' \
            'FROM (VALUES {}) as c(amount, update_date,
inventory_person_id, item_id) ' \
            'WHERE c.inventory_person_id = t.inventory_person_id
AND c.item_id = t.item_id'
            .format(temp_list),
list_to_tuple(inventory_person_items_update_array))

def generate_data(args):
    generate_item()
    generate_class_of_person()

    if args.user_data is not None:
        generate_user_data(arguments.user_data)

    if args.persons is not None:
        generate_person(arguments.persons)

    if args.inventory_person is not None:
        generate_inventory_person(arguments.inventory_person, False)

    if args.inventory_person_items is not None:
        generate_inventory_person_items(arguments.inventory_person_items)

    if args.meetup is not None:
        generate_meetup(arguments.meetup)

if __name__ == '__main__':
    args = argparse.ArgumentParser(description="Details of data generation")
    args.add_argument('-u', action="store", dest="user_data")
    args.add_argument('-p', action="store", dest="persons")
    args.add_argument('-inv', action="store", dest="inventory_person")
    args.add_argument('-i', action="store", dest="inventory_person_items")
    args.add_argument('-m', action="store", dest="meetup")
    arguments = args.parse_args()

    generate_data(arguments)

    connection.commit()

```

```
cursor.close()
connection.close()
```

Листинг 5.3. Общие запросы для лабораторной работы №3

```
--1. Вывод данных из таблиц
create or replace view task1_1 as SELECT * FROM class_of_person;
create or replace view task1_2 as SELECT * FROM person;
create or replace view task1_3 as SELECT * FROM user_data;
create or replace view task1_4 as SELECT * FROM person_skill;
create or replace view task1_5 as SELECT * FROM skill;
create or replace view task1_6 as SELECT * FROM meetup;
create or replace view task1_7 as SELECT * FROM inventory_person;
create or replace view task1_8 as SELECT * FROM inventory_person_items;
create or replace view task1_9 as SELECT * FROM item;

--2. Сделать выборку данных из одной таблицы при нескольких условиях, с
использованием логических операций,
--**LIKE**, **BETWEEN**, **IN**
create or replace view task2_1 as SELECT * FROM person WHERE health BETWEEN
150 AND 170;
create or replace view task2_2 as SELECT * FROM user_data WHERE nickname LIKE
'dec%';
create or replace view task2_3 as SELECT * FROM inventory_person_items WHERE
inventory_person_id IN ('2', '212', '1212');

--3. Создать в запросе вычисляемое поле
create or replace view task3 as SELECT inventory_person_id, add_date,
update_date, extract(epoch FROM update_date -
add_date)/3600/24 AS delta_days FROM inventory_person_items WHERE add_date !=
update_date ORDER BY delta_days;

--4. Сделать выборку всех данных с сортировкой по нескольким полям
create or replace view task4 as SELECT * FROM inventory_person_items ORDER BY
item_id, amount;

--5. Создать запрос, вычисляющий несколько совокупных характеристик таблиц
create or replace view task5 as SELECT ROUND(AVG(health), 2) AS "AVG Hp",
MAX(experience) AS "MAX Exp" FROM person;

--6. Сделать выборку данных из связанных таблиц
create or replace view task6 as SELECT
    c.name as "Class name", s.name as "Skill name", s.cost
FROM class_of_person c
    INNER JOIN skill s on s.class_of_person_id = c.id;

create or replace view task6_2 as SELECT
    p.id, ud.nickname, c.name
FROM
    person p
    INNER JOIN user_data ud ON p.user_id = ud.id
    INNER JOIN class_of_person c ON p.class_of_person_id = c.id
ORDER BY ud.nickname;

--7. Создать запрос, рассчитывающий совокупную характеристику с использованием
группировки,
--наложите ограничение на результат группировки
create or replace view task7 as SELECT
    inventory_person_id, COUNT(ipi.inventory_person_id), SUM(amount)
FROM inventory_person_items ipi
GROUP BY ipi.inventory_person_id HAVING SUM(amount)>5
ORDER BY ipi.inventory_person_id
```

```
--8. Придумать и реализовать пример использования вложенного запроса
create or replace view task8 as SELECT COUNT(*)
FROM person p
WHERE class_of_person_id = (SELECT cop.id FROM class_of_person cop WHERE
cop.name = 'Archer');

--9. С помощью команды **INSERT** добавить в каждую таблицу по одной записи.
CREATE OR REPLACE PROCEDURE task9() LANGUAGE plpgsql AS
$$ BEGIN
    INSERT INTO public.user_data(email, password, nickname) VALUES
('task_email@ya.ru', 'lol02olo', 'task_nickname') ON CONFLICT DO NOTHING;
    INSERT INTO public.person(class_of_person_id, health, experience,
user_id, update_date, is_enemy) VALUES (1, 100, 50, 1, now(), False) ON
CONFLICT DO NOTHING;
    INSERT INTO public.meetup(person_id, result, meetup_date, enemy_id)
VALUES (1, 'draw', now(), 2) ON CONFLICT DO NOTHING;
    INSERT INTO public.class_of_person(name, description) VALUES
('task_class', 'aaaolo') ON CONFLICT DO NOTHING;
    INSERT INTO public.skill(name, description, cost, class_of_person_id)
VALUES ('task_skill', 'ldsdaolo', 50, 6) ON CONFLICT DO NOTHING;
    INSERT INTO public.person_skill(person_id, skill_id, equiped) VALUES (1,
4, True) ON CONFLICT DO NOTHING;
    INSERT INTO public.inventory_person(person_id, inventory_size) VALUES (1,
1000) ON CONFLICT DO NOTHING;
    INSERT INTO public.item(name, description) VALUES ('test_item',
'dsdaw32') ON CONFLICT DO NOTHING;
    INSERT INTO public.inventory_person_items(inventory_person_id, item_id,
add_date, is_deleted, amount, update_date) VALUES (1, 4, now(), True, 1,
now()) ON CONFLICT DO NOTHING;
END $$;
CALL task9();

--10. С помощью оператора **UPDATE** измените значения нескольких полей у
всех записей,
--отвечающих заданному условию
CREATE OR REPLACE PROCEDURE task10() LANGUAGE plpgsql AS
$$ BEGIN
    UPDATE person SET health = 200, update_date = now() WHERE health = 170
AND user_id = 2;
END $$;
CALL task10();

--11. С помощью оператора **DELETE** удалить запись,
--имеющую максимальное (минимальное) значение некоторой совокупной
характеристики
CREATE OR REPLACE PROCEDURE task11() LANGUAGE plpgsql AS
$$ BEGIN
DELETE FROM person WHERE id = (SELECT person_id
                                FROM (SELECT person_id, count(*) AS
meetup_count
                                FROM meetup
                                GROUP BY person_id
                                ORDER BY meetup_count DESC LIMIT 1) AS
most_pupular_meetup_person);
END $$;
CALL task11();

--12. С помощью оператора **DELETE** удалить записи в главной таблице,
--на которые не ссылается подчиненная таблица (используя вложенный запрос)
CREATE OR REPLACE PROCEDURE task12() LANGUAGE plpgsql AS
$$ BEGIN
    DELETE FROM class_of_person cop WHERE cop.id NOT IN (SELECT DISTINCT
class_of_person_id FROM person);
```

```
END $$;  
CALL task12();
```

Листинг 5.4. Код программы нагрузки бд для лабораторной работы №4

```
Execute.py  
  
import threading  
import time  
  
import matplotlib.pyplot as plt  
from const import Data  
from dbFunctions import DBFunctions  
  
dbFunction = DBFunctions()  
dbFunction.connect_to_database()  
const_and_data = Data()  
  
use_prepare = False  
threads_and_time = True  
plot_x = []  
plot_y = []  
results_d_threads = []  
unit_of_time = 20  
  
def main():  
    global plot_x  
    global plot_y  
    global use_prepare  
    global threads_and_time  
    min_req = 10  
    max_req = 1000  
    step = 10  
  
    # dbFunction.drop_index_if_exists()  
    # threads_and_time = False  
    # threads_count = 1  
    # print("Смотрим зависимость при 1 потоке")  
    # x1, y1 = get_requests_and_time_dependency(min_req, max_req, step,  
False, threads_count)  
    #  
    # threads_count = 50  
    # print("Смотрим зависимость при 50 потоках")  
    # x2, y2 = get_requests_and_time_dependency(min_req, max_req, step,  
False, threads_count)  
    #  
    # threads_count = 97  
    # print("Смотрим зависимость при 97 потоках")  
    # x3, y3 = get_requests_and_time_dependency(min_req, max_req, step,  
False, threads_count)  
    #  
    # plt.title('Время/запросы без оптимизации, 10 сек')  
    # plt.plot(x1, y1, label='1 поток')  
    # plt.plot(x2, y2, label='50 потоков')  
    # plt.plot(x3, y3, label='97 потоков')  
    # plt.xlabel('Кол-во запросов')  
    # plt.ylabel('Среднее время ответа на запрос в мс')  
    # plt.legend()  
    # plt.show()  
  
#####
```

```

dbFunction.drop_index_if_exists()
threads_and_time = False
threads_count = 50
x1, y1 = get_requests_and_time_dependency(min_req, max_req, step, False,
threads_count)

dbFunction.add_indexes()
threads_count = 50
x2, y2 = get_requests_and_time_dependency(min_req, max_req, step, False,
threads_count)

use_prepare = True
threads_count = 50
x3, y3 = get_requests_and_time_dependency(min_req, max_req, step, False,
threads_count)

plt.title('Время/запросы, 50 потоков, 20 сек, шаг 10')
plt.plot(x1, y1, label='До оптимизации')
plt.plot(x2, y2, label='Индексы')
plt.plot(x3, y3, label='Индексы + Prepare')
plt.xlabel('Кол-во запросов')
plt.ylabel('Среднее время ответа на запрос в мс')
plt.legend()
plt.show()

#####

# dbFunction.drop_index_if_exists()
# threads_and_time = True
# threads_count = 97
# x1, y1 = get_threads_and_time_dependency(min_req, max_req, step,
threads_count, False)
#
# dbFunction.add_indexes()
# use_prepare = True
# x2, y2 = get_threads_and_time_dependency(min_req, max_req, step,
threads_count, False)
#
# plt.title('Время/потоки')
# plt.plot(x1, y1, label='До оптимизации')
# plt.plot(x2, y2, label='После оптимизации')
# plt.xlabel('Кол-во потоков')
# plt.ylabel('Среднее время ответа на запрос в мс')
# plt.legend()
# plt.show()

def get_requests_and_time_dependency(min_req, max_req, step, need_to_plot,
threads_count):
    global threads_result

    threads_result = []
    for t in range(threads_count):
        qt = QueryThread(min_req, max_req, step)
        qt.start()

    while threading.activeCount() > 1:
        time.sleep(1)

    number_with_max_req_in_sec = 0
    max_req_in_sec = 0
    for i in range(1, len(threads_result)):
        values = threads_result[i - 1]

```

```

        if len(values[0]) > max_req_in_sec:
            max_req_in_sec = len(values[0])
            number_with_max_req_in_sec = i

    values = threads_result[number_with_max_req_in_sec - 1]
    print("Смотрим результаты ", str(number_with_max_req_in_sec), " потока")
    plot_x = values[0]
    plot_y = values[1]

    if need_to_plot:
        plt.plot(plot_x, plot_y, linewidth=2.0, label='Количество потоков = '
+ str(threads_count), color="red")
        plt.xlabel('Количество запросов в секунду')
        plt.ylabel('Время ответа на один запрос, мс')
        plt.legend()
        plt.show()
    return plot_x, plot_y

def get_threads_and_time_dependency(min_queries, max_queries, step,
num_threads, need_to_plot):
    plot_x = [k for k in range(1, num_threads + 1)]
    plot_y = []

    global cur_num_threads

    for cur_num_threads in range(1, num_threads + 1):
        results_d_threads.clear()
        cur_threads_sum = 0
        print("count time for " + str(cur_num_threads) + " threads")
        for t in range(cur_num_threads):
            dbt = QueryThread(min_queries, max_queries, step)
            dbt.start()

            while threading.activeCount() > 1:
                time.sleep(1)

        for f in range(1, len(results_d_threads)):
            cur_threads_sum += results_d_threads[f - 1]

        plot_y.append(cur_threads_sum/cur_num_threads)

    if need_to_plot:
        plt.plot(plot_x, plot_y, linewidth=2.0)
        plt.xlabel('Количество потоков')
        plt.ylabel('Время ответа на один запрос, мс')
        plt.show()
    return plot_x, plot_y

class QueryThread(threading.Thread):
    def __init__(self, min_queries, max_queries, step):

        threading.Thread.__init__(self)
        self.DBFunctions = DBFunctions()
        self.DBFunctions.connect_to_database()
        self.max_queries = max_queries
        self.min_queries = min_queries
        self.step = step

        if use_prepare:
            self.DBFunctions.prepare_queries()

```

```

        self.DBFunctions.connection.commit()

    def run(self):
        if not threads_and_time:
            thread_failed_with_time = False
            res_x = []
            res_y = []
            target_RPS = self.min_queries
            while target_RPS <= self.max_queries:
                if thread_failed_with_time:
                    break
                executed_RPS = 0
                exec_sum = 0
                start_time = time.localtime().tm_sec

                while executed_RPS < target_RPS:
                    executed_RPS += 1
                    exec_time = self.execute()
                    exec_sum += exec_time
                    run_queries_time = time.localtime().tm_sec - start_time

                    if run_queries_time >= unit_of_time or executed_RPS ==
target_RPS:
                        if unit_of_time - run_queries_time > 0:
                            time.sleep(unit_of_time - run_queries_time)
                        if executed_RPS < target_RPS:
                            thread_failed_with_time = True
                            break
                        else:
                            res_x.append(executed_RPS)
                            res_y.append(exec_sum / executed_RPS)

                            target_RPS += self.step
                            thread_res = (res_x, res_y)
                            threads_result.append(thread_res)

                else:
                    exec_sum = 0
                    executed_req = 0
                    for j in range(self.min_queries, self.max_queries + 1,
self.step):
                        exec_time = self.execute()
                        exec_sum += exec_time
                        executed_req += 1

                        results_d_threads.append(exec_sum / executed_req)

            def execute(self):
                if not use_prepare:
                    result =
self.DBFunctions.execute_random_query_and_get_time(const_and_data)
                    return result

                else:
                    result =
self.DBFunctions.execute_random_query_with_optimisation_and_get_time(const_an
d_data)
                    return result

if __name__ == "__main__":
    main()

```

```
dbFunctions.py
```

```
from random import Random
import psycopg2
```

```
class DBFunctions():
```

```
    def __init__(self):
        self.connection = None
```

```
    def connect_to_database(self):
        self.connection = psycopg2.connect(dbname='database_for_game_big',
                                           user='postgres',
                                           password='6559',
                                           host='localhost')

        self.cursor = self.connection.cursor()
        self.connection.autocommit = True
```

```
    def get_data_for_request(self, nickname, battle_result, id_of_person,
class_of_person, item_date):
```

```
        self.cursor.execute("SELECT nickname FROM user_data;")
        result = self.cursor.fetchall()
        for i in range(0, len(result)):
            nickname.append(result[i][0])
```

```
        self.cursor.execute("SELECT result FROM meetup;")
        result = self.cursor.fetchall()
        for i in range(0, len(result)):
            battle_result.append(result[i][0])
```

```
        self.cursor.execute("SELECT id FROM person;")
        result = self.cursor.fetchall()
        for i in range(0, len(result)):
            id_of_person.append(result[i][0])
```

```
        self.cursor.execute("SELECT name FROM class_of_person;")
        result = self.cursor.fetchall()
        for i in range(0, len(result)):
            class_of_person.append(result[i][0])
```

```
        self.cursor.execute("SELECT add_date FROM inventory_person_items;")
        result = self.cursor.fetchall()
        for i in range(0, len(result)):
            item_date.append(result[i][0])
```

```
    def execute_random_query_and_get_time(self, const_and_data):
        random = Random().randint(1, 5)
```

```
        # Запрос 1
```

```
        # Вывести информацию обо всех персонажах юзера не активных на
протяжении года
```

```
        # на которо нападали больше чем напал сам персонаж (параметр -
никнейм)
```

```
        query_1 = "SELECT DISTINCT p.id, cap.name AS class FROM person p " \
                  "INNER JOIN class_of_person cap ON p.class_of_person_id =
```

```
cap.id " \
```

```
                  "INNER JOIN user_data ud ON p.user_id = ud.id " \
```

```
                  "INNER JOIN meetup m on p.id = m.person_id OR p.id =
```

```
m.enemy_id " \
```

```
                  "WHERE ud.nickname = %(nickname)s " \
```

```
                  "AND p.update_date <= now() - '1 year'::interval " \
```

```
                  "GROUP BY p.id, cap.name, m.person_id, m.enemy_id " \
```



```

        "HAVING count(m.enemy_id = p.id OR NULL) != 0 " \
        "AND count(m.person_id = p.id OR NULL)/count(m.enemy_id =
p.id OR NULL) < 1;"

# Запрос 2
# Вывести ники юзеров-лидеров и кол-во требуемых результатов
(победа/поражение/ничья) в инициированных
# сражениях за последний год (параметр - тип результата сражения)
query_2 = "SELECT ud.nickname, count(m.*) AS results FROM user_data
ud " \
        "INNER JOIN person p ON ud.id = p.user_id " \
        "INNER JOIN meetup m ON p.id = m.person_id " \
        "AND m.result = %(battle_result)s " \
        "AND m.meetup_date >= now() - '1 year'::interval " \
        "AND m.meetup_date < now() " \
        "GROUP BY ud.nickname " \
        "ORDER BY results DESC;"

# Запрос 3
# Вывести и стакнуть артефакты из всех инвентарей персонажа (параметр
- id персонажа)
query_3 = "SELECT i.name, SUM(amount) FROM inventory_person_items ipi
" \
        "INNER JOIN item i on i.id = ipi.item_id " \
        "INNER JOIN inventory_person ip on ipi.inventory_person_id
= ip.id " \
        "INNER JOIN person p on ip.person_id = p.id " \
        "WHERE p.id = %(id_of_person)s " \
        "GROUP BY i.name;"

# Запрос 4
# Вывести персонажей определённого класса которые прошли всю игру
# (вкачали оба скилла, макс жизней, получили редкий айтем) (параметр
- имя класса)
query_4 = "SELECT ps.person_id FROM person_skill ps " \
        "INNER JOIN skill s ON ps.skill_id = s.id " \
        "INNER JOIN class_of_person cop on s.class_of_person_id =
cop.id " \
        "INNER JOIN person p ON ps.person_id = p.id " \
        "INNER JOIN inventory_person ip ON p.id = ip.person_id " \
        "INNER JOIN inventory_person_items ipi on ip.id =
ipi.inventory_person_id " \
        "WHERE cop.name = %(class_of_person)s AND health = 300 AND
ipi.item_id = 11 " \
        "GROUP BY ps.person_id HAVING count(equiped=true OR
NULL)=2;"

# Запрос 5
# Вывести рейтинг игроков (ники) по полученным за определённый месяц
редким артефактам
# для всех персонажей (параметр - даты)
query_5 = "SELECT ud.nickname, sum(ipi.amount) FROM
inventory_person_items ipi " \
        "INNER JOIN inventory_person ip on ip.id =
ipi.inventory_person_id " \
        "INNER JOIN person p on p.id = ip.person_id " \
        "INNER JOIN user_data ud on p.user_id = ud.id " \
        "WHERE ipi.update_date < %(item_date)s + '1 year'::interval
" \
        "AND ipi.update_date >= %(item_date)s " \
        "AND ipi.add_date < %(item_date)s + '1 year'::interval " \
        "AND ipi.add_date >= %(item_date)s " \
        "AND ipi.item_id = 11 " \
        "GROUP BY ud.nickname " \

```

```

        "ORDER BY sum(ipi.amount) DESC;"

    queries = (query_1, query_2, query_3, query_4, query_5)

    if random == 1:
        nickname = const_and_data.nickname[Random().randint(0,
(len(const_and_data.nickname) - 1))]
        self.cursor.execute("EXPLAIN ANALYZE " + queries[random - 1],
{"nickname": nickname})

    if random == 2:
        battle_result = const_and_data.battle_result[Random().randint(0,
len(const_and_data.battle_result) - 1)]
        self.cursor.execute("EXPLAIN ANALYZE " + queries[random - 1],
{"battle_result": battle_result})

    if random == 3:
        id_of_person = const_and_data.id_of_person[Random().randint(0,
len(const_and_data.id_of_person) - 1)]
        self.cursor.execute("EXPLAIN ANALYZE " + queries[random - 1],
{"id_of_person": id_of_person})

    if random == 4:
        class_of_person =
const_and_data.class_of_person[Random().randint(0,
(len(const_and_data.class_of_person) - 1))]
        self.cursor.execute("EXPLAIN ANALYZE " + queries[random - 1],
{"class_of_person": class_of_person})

    if random == 5:
        item_date = const_and_data.item_date[Random().randint(0,
(len(const_and_data.item_date) - 1))]
        self.cursor.execute("EXPLAIN ANALYZE " + queries[random - 1],
{"item_date": item_date})

    res = self.cursor.fetchall()
    self.connection.commit()
    return float(res[-1][0].split(" ")[2])

    def add_indexes(self):
        self.cursor.execute("CREATE INDEX IF NOT EXISTS meetup_person_idx ON
meetup(person_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS meetup_enemy_idx ON
meetup(enemy_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS meetup_date_idx ON
meetup(meetup_date);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS
inventory_person_item_inventory_person_idx "
"ON
inventory_person_items(inventory_person_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS
inventory_person_person_idx ON inventory_person(person_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS
person_skill_skill_idx ON person_skill(skill_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS person_health_idx ON
person(health);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS
inventory_person_items_item_addupdate_date_idx "
"ON inventory_person_items(item_id, add_date,
update_date);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS person_user_idx ON
person(user_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS person_class_idx ON
person(class_of_person_id);")

```

```

        self.cursor.execute("CREATE INDEX IF NOT EXISTS user_nickname_idx ON
user_data(nickname);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS
inventory_person_item_inventory_person_idx "
                            "ON
inventory_person_items(inventory_person_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS
inventory_person_item_inventory_person_item_idx "
                            "ON inventory_person_items(inventory_person_id,
item_id);")
        self.cursor.execute("CREATE INDEX IF NOT EXISTS
inventory_person_item_idx ON inventory_person_items(item_id);")

    def drop_index_if_exists(self):
        self.cursor.execute("DROP INDEX IF EXISTS meetup_person_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS meetup_enemy_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS meetup_date_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS
inventory_person_item_inventory_person_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS
inventory_person_person_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS person_skill_skill_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS person_health_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS
inventory_person_items_item_addupdate_date_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS person_user_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS person_class_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS user_nickname_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS
inventory_person_item_inventory_person_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS
inventory_person_item_inventory_person_item_idx;")
        self.cursor.execute("DROP INDEX IF EXISTS
inventory_person_item_idx;")

    def prepare_queries(self):
        # Вывести информацию обо всех персонажах юзера не активных на
        # протяжении года
        # на которо нападали больше чем напал сам персонаж (параметр -
        # никнейм)
        self.cursor.execute("PREPARE query1 (varchar(20)) AS "
                            "SELECT DISTINCT p.id, cap.name AS class FROM
person p "
                            "INNER JOIN class_of_person cap ON
p.class_of_person_id = cap.id "
                            "INNER JOIN user_data ud ON p.user_id = ud.id "
                            "INNER JOIN meetup m on p.id = m.person_id OR
p.id = m.enemy_id "
                            "WHERE ud.nickname = $1 "
                            "AND p.update_date <= now() - '1 year'::interval
"
                            "GROUP BY p.id, cap.name, m.person_id, m.enemy_id
"
                            "HAVING count(m.enemy_id = p.id OR NULL) != 0 "
                            "AND count(m.person_id = p.id OR
NULL)/count(m.enemy_id = p.id OR NULL) < 1;")

        # Вывести ники юзеров-лидеров и кол-во требуемых результатов
        (победа/поражение/ничья) в инициированных
        # сражениях за последний год (параметр - тип результата сражения)
        self.cursor.execute("PREPARE query2 AS "
                            "SELECT ud.nickname, count(m.*) AS results FROM
user_data ud "
                            "INNER JOIN person p ON ud.id = p.user_id ")

```

```

"INNER JOIN meetup m ON p.id = m.person_id "
"AND m.result = $1 "
"AND m.meetup_date >= now() - '1 year'::interval
"
"AND m.meetup_date < now() "
"GROUP BY ud.nickname "
"ORDER BY results DESC;")

# Вывести и стакнуть артефакты из всех инвентарей персонажа (параметр
- id персонажа)
self.cursor.execute("PREPARE query3 (bigint) AS "
"SELECT i.name, SUM(amount) FROM
inventory_person_items ipi "
"INNER JOIN item i on i.id = ipi.item_id "
"INNER JOIN inventory_person ip on
ipi.inventory_person_id = ip.id "
"INNER JOIN person p on ip.person_id = p.id "
"WHERE p.id = $1 "
"GROUP BY i.name;")

# Вывести персонажей определённого класса которые прошли всю игру
# (вкачали оба скилла, макс жизней, получили редкий айтем) (параметр
- имя класса)
self.cursor.execute("PREPARE query4 (varchar(30)) AS "
"SELECT ps.person_id FROM person_skill ps "
"INNER JOIN skill s ON ps.skill_id = s.id "
"INNER JOIN class_of_person cop on
s.class_of_person_id = cop.id "
"INNER JOIN person p ON ps.person_id = p.id "
"INNER JOIN inventory_person ip ON p.id =
ip.person_id "
"INNER JOIN inventory_person_items ipi on ip.id =
ipi.inventory_person_id "
"WHERE cop.name = $1 AND health = 300 AND
ipi.item_id = 11 "
"GROUP BY ps.person_id HAVING count(equiped=true
OR NULL)=2;")

# Вывести рейтинг игроков (ники) по полученным за определённый месяц
редким артефактам
# для всех персонажей (параметр - даты)
self.cursor.execute("PREPARE query5 AS "
"SELECT ud.nickname, sum(ipi.amount) FROM
inventory_person_items ipi "
"INNER JOIN inventory_person ip on ip.id =
ipi.inventory_person_id "
"INNER JOIN person p on p.id = ip.person_id "
"INNER JOIN user_data ud on p.user_id = ud.id "
"WHERE ipi.update_date < $1::timestamp + '1
year'::interval "
"AND ipi.update_date >= $1 "
"AND ipi.add_date < $1 + '1 year'::interval "
"AND ipi.add_date >= $1 "
"AND ipi.item_id = 11 "
"GROUP BY ud.nickname "
"ORDER BY sum(ipi.amount) DESC;")

self.connection.commit()

def execute_random_query_with_optimisation_and_get_time(self,
const_and_data):
    random = Random().randint(1, 5)

    if random == 1:

```

```

        nickname = const_and_data.nickname[Random().randint(0,
(len(const_and_data.nickname) - 1))]
        self.cursor.execute("EXPLAIN ANALYZE EXECUTE query1
('{}');".format(nickname))

        if random == 2:
            battle_result = const_and_data.battle_result[Random().randint(0,
(len(const_and_data.battle_result) - 1))]
            self.cursor.execute("EXPLAIN ANALYZE EXECUTE query2
('{}');".format(battle_result))

            if random == 3:
                id_of_person = const_and_data.id_of_person[Random().randint(0,
(len(const_and_data.id_of_person) - 1))]
                self.cursor.execute("EXPLAIN ANALYZE EXECUTE query3
('{}');".format(id_of_person))

            if random == 4:
                class_of_person = \
                    const_and_data.class_of_person[Random().randint(0,
(len(const_and_data.class_of_person) - 1))]
                self.cursor.execute("EXPLAIN ANALYZE EXECUTE query4
('{}');".format(class_of_person))

            if random == 5:
                item_date = const_and_data.item_date[Random().randint(0,
(len(const_and_data.item_date) - 1))]
                self.cursor.execute("EXPLAIN ANALYZE EXECUTE query5
('{}');".format(item_date))

        res = self.cursor.fetchall()
        return float(res[-1][0].split(" ")[2])

```

const.py

from dbFunctions import DBFunctions

class Data:

```

    def __init__(self):
        self.nickname = list()
        self.battle_result = list()
        self.id_of_person = list()
        self.class_of_person = list()
        self.item_date = list()

        dbFunctions = DBFunctions()
        dbFunctions.connect_to_database()
        dbFunctions.get_data_for_request(self.nickname,
                                         self.battle_result,
                                         self.id_of_person,
                                         self.class_of_person,
                                         self.item_date)

        self.second_and_requests = dict()
        self.millisecond_and_requests = []
        self.threads_vs_time = dict()
        for d in range(1, 1000000):
            self.second_and_requests[d] = []

```