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

Базы данных

Отчет по лабораторным работам

Работу выполнил:
Калашников Р.А.
Группа:
3530901/70203
Преподаватель:
Мяснов А.В.

Санкт-Петербург
2020

Содержание

1. Лабораторная работа №1. Проектирование базы данных

1.1. Цель работы

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

1.2. Программа работы

- Выбор задания (предметной области), описание набора данных и требований к хранимым данным в свободном формате в wiki своего проекта в GitLab.
- Формирование в свободном формате (предпочтительно в виде графической схемы) схемы БД.
- 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. Выполнение работы

Выбранная тема - стриминговый сервис видеоигр. В базе данных должны храниться:

- Информация о клиентах: персональные данные, приобретенные игры, а также активные и неактивные подписки;
- Информация об играх: цена, а также жанровая принадлежность;
- Информация об использовании компьютеров.

Таблицы базы данных:

- client - содержит информацию о клиентах. Столбцы:
  - nickname - имя пользователя, может быть неуникальным;
  - hash - хэш пароля;
  - email - почтовый адрес пользователя, должен быть уникальным.
- game - содержит информацию об играх
  - title - название гры;
  - price - цена игры.
- genre - содержит жанры игр
  - name - название жанра.

- subscription_plan - содержит информацию о подписочных планах
  - name - название плана;
  - price - цена плана за месяц.
- machine - содержит информацию о компьютерах
  - power_tier - мощность компьютера.
- owned_game - содержит информацию о приобретенных играх
  - client_id - id клиента, который приобрел игру;
  - game_id - id приобретенной игры;
  - purchase_date - дата приобретения.
- game_genre - содержит информацию о жанрах игр
  - game_id - id игры;
  - genre_id - id жанра.
- installed_game - содержит информацию об установленных на компьютерах играх
  - machine_id - id компьютера;
  - game_id - id игры.
- client_subscription_plan - содержит информацию о подписочных планах клиентов
  - client_id - id клиента;
  - subscription_plan_id - id плана подписки;
  - active_from - дата начала действия плана;
  - active_to - дата окончания действия плана.
- machine_usage - содержит информацию об использовании компьютеров
  - owned_game_id - id приобретенной игры;
  - machine_id - id компьютера;
  - in_use_from - время начала использования компьютера;
  - in_use_to - время окончания использования компьютера.
- available_machine_tier - содержит информацию о доступности компьютеров для подписочных планов
  - subscription_plan_id - id плана подписки;
  - machine_id - id компьютера;

Для id использовался тип serial, для строковых полей, таких как никнейм клиента и название игры, varchar, а для дат date или timestamp в зависимости от необходимой точности. Параметр ON DELETE для большинства внешних ключей был установлен CASCADE, так как предполагается, что при удалении объекта, на который ссылается запись в таблице, дальнейший доступ к ней нужен не будет. Исключением является поле owned_game_id в таблице machine_usage, где установлено SET NULL, так как даже при исчезновении игры, запись о сессии стоит оставить.

Скрипт создания таблиц в базе данных:

Листинг 1: Создание таблиц базы данных

```
CREATE TABLE client
(
    id SERIAL PRIMARY KEY,
    nickname CHARACTER VARYING(30) NOT NULL CHECK(nickname !=''),
    hash INTEGER NOT NULL,
    email CHARACTER VARYING(30) UNIQUE CHECK(email !='')
);

CREATE TABLE game
(
    id SERIAL PRIMARY KEY,
  title CHARACTER VARYING(30) NOT NULL UNIQUE CHECK(title !=''),
  price INTEGER NOT NULL
);

CREATE TABLE genre
(
    id SERIAL PRIMARY KEY,
  name CHARACTER VARYING(30) CHECK(name !='') UNIQUE
);

CREATE TABLE subscription_plan
(
    id SERIAL PRIMARY KEY,
  name CHARACTER VARYING(30) CHECK(name !='') UNIQUE,
  price INTEGER NOT NULL
);

CREATE TABLE machine
(
    id SERIAL PRIMARY KEY,
  power_tier INTEGER CHECK(power_tier > 0 AND power_tier < 4)
);

CREATE TABLE owned_game
(
    id SERIAL PRIMARY KEY,
  client_id bigint NOT NULL REFERENCES client(id) ON DELETE CASCADE,
  game_id bigint NOT NULL REFERENCES game(id) ON DELETE CASCADE,
  purchase_date DATE NOT NULL
);

CREATE TABLE game_genre
(
    id SERIAL PRIMARY KEY,
  game_id bigint NOT NULL REFERENCES game(id) ON DELETE CASCADE,
  genre_id bigint NOT NULL REFERENCES genre(id) ON DELETE CASCADE
);

CREATE TABLE installed_game
(
    id SERIAL PRIMARY KEY,
  machine_id bigint NOT NULL REFERENCES machine(id) ON DELETE CASCADE,
  game_id bigint NOT NULL REFERENCES game(id) ON DELETE CASCADE
);

CREATE TABLE client_subscription_plan
(
    id SERIAL PRIMARY KEY,
```

```
60    client_id bigint NOT NULL REFERENCES client(id) ON DELETE CASCADE,
61    subscription_plan_id bigint NOT NULL REFERENCES subscription_plan(id) ON
        ↪ DELETE CASCADE,
62    active_from DATE NOT NULL,
63    active_to DATE NOT NULL
64 );
65
66 CREATE TABLE machine_usage
67 (
68       id SERIAL PRIMARY KEY,
69    owned_game_id bigint NOT NULL REFERENCES owned_game(id) ON DELETE SET NULL,
70    machine_id bigint NOT NULL REFERENCES machine(id) ON DELETE CASCADE,
71    in_use_from TIMESTAMP NOT NULL,
72    in_use_to TIMESTAMP
73 );
74
75 CREATE TABLE available_machine_tier
76 (
77       id SERIAL PRIMARY KEY,
78    subscription_plan_id bigint NOT NULL REFERENCES subscription_plan(id) ON
        ↪ DELETE CASCADE,
79    machine_id bigint NOT NULL REFERENCES machine(id) ON DELETE CASCADE
80 );
```

Схема полученной базы данных представлена на рисунке 1.1

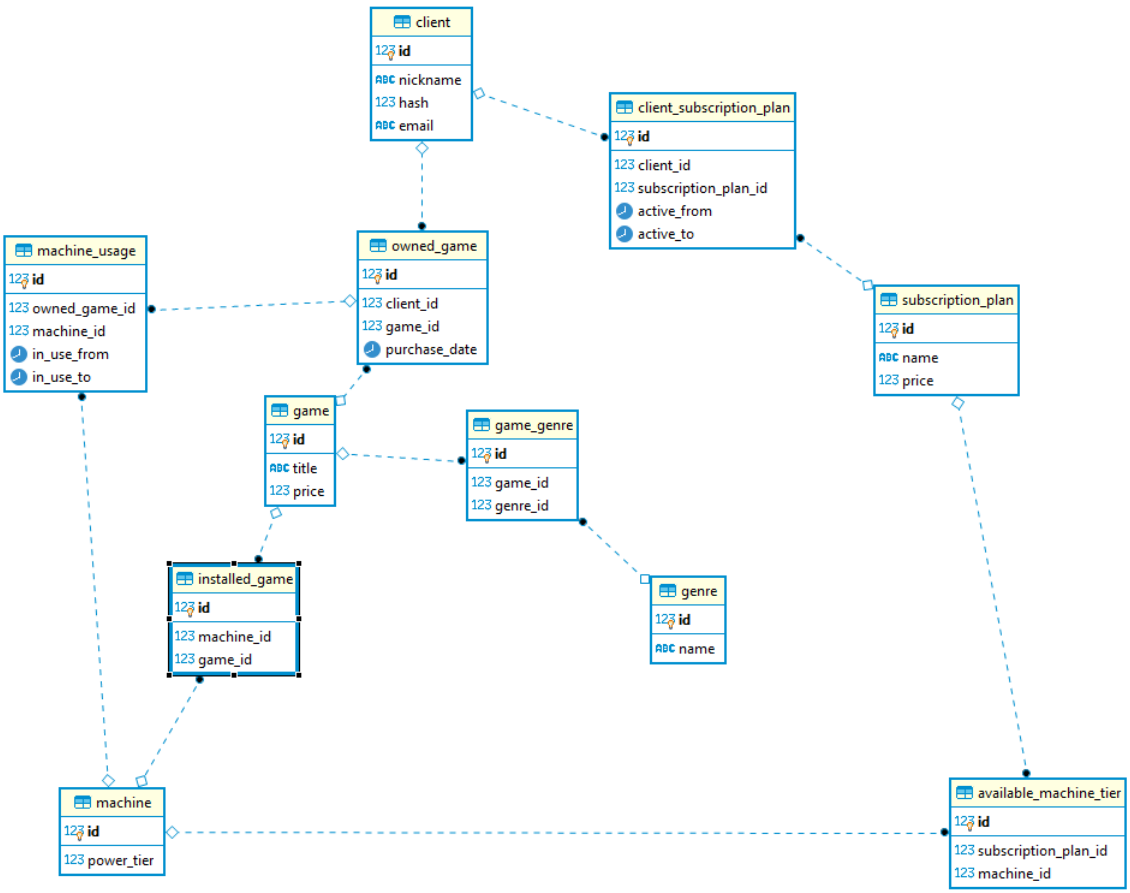

Рисунок 1.1. Схема базы данных

Для заполнения базы данных был написан скрипт, добавляющий по 3 строки в каждую таблицу. Для заполнения полей с внешними ключами был использован оператор SELECT.

Листинг 2: Заполнение базы данных

```
INSERT INTO client (nickname, hash, email)
VALUES ('example_client', 5, 'example1@mail.ru'),
('new_client', 28, 'new@gmail.ru');

INSERT INTO game (title, price)
VALUES ('DOOM', 2000),
('Mario', 4500),
('Trails of cold steel', 1000);

INSERT INTO genre (name)
VALUES ('action'),
('arcade'),
('rpg');

INSERT INTO machine (power_tier)
VALUES (1),
(2),
(3);

INSERT INTO subscription_plan (name, price)
VALUES ('low-tier plan', 100),
('middle-tier plan', 200),
('high-tier plan', 300);

INSERT INTO available_machine_tier (subscription_plan_id, machine_id)
VALUES ((SELECT id from subscription_plan WHERE name = 'low-tier plan'), (SELECT
    ↪  id from machine WHERE power_tier = 1)),
((SELECT id from subscription_plan WHERE name = 'middle-tier plan'), (SELECT id
    ↪ from machine WHERE power_tier = 2)),
((SELECT id from subscription_plan WHERE name = 'high-tier plan'), (SELECT id
    ↪ from machine WHERE power_tier = 3));

INSERT INTO client_subscription_plan (client_id, subscription_plan_id,
    ↪ active_from, active_to)
VALUES ((SELECT id from client WHERE email = 'example1@mail.ru'), (SELECT id
    ↪ from subscription_plan WHERE name = 'low-tier plan'), '2019-06-01', '
    ↪ 2020-06-01'),
((SELECT id from client WHERE email = 'new@gmail.ru'), (SELECT id from
    ↪ subscription_plan WHERE name = 'middle-tier plan'), '2019-04-02', '
    ↪ 2020-04-02');

INSERT INTO game_genre (game_id, genre_id)
VALUES ((SELECT id from game WHERE title = 'DOOM'), (SELECT id from genre WHERE
    ↪ name = 'action')),
((SELECT id from game WHERE title = 'Mario'), (SELECT id from genre WHERE name =
    ↪  'arcade')),
((SELECT id from game WHERE title = 'Trails of cold steel'), (SELECT id from
    ↪ genre WHERE name = 'rpg'));

INSERT INTO installed_game (machine_id, game_id)
VALUES ((SELECT id from machine WHERE power_tier = 3), (SELECT id from game
    ↪ WHERE title = 'DOOM')),
((SELECT id from machine WHERE power_tier = 1), (SELECT id from game WHERE title
    ↪  = 'Mario')),
((SELECT id from machine WHERE power_tier = 2), (SELECT id from game WHERE title
```

```
    ↪ = 'Trails of cold steel'));
43
44 INSERT INTO owned_game (client_id, game_id, purchase_date)
45 VALUES ((SELECT id from client WHERE email = 'example1@mail.ru'), (SELECT id
    ↪ from game WHERE title = 'DOOM'), '2019-06-02'),
46 ((SELECT id from client WHERE email = 'new@gmail.ru'), (SELECT id from game
    ↪ WHERE title = 'Mario'), '2019-07-12'),
47 ((SELECT id from client WHERE email = 'new@gmail.ru'), (SELECT id from game
    ↪ WHERE title = 'Trails of cold steel'), '2019-08-27');
48
49 INSERT INTO machine_usage (owned_game_id, machine_id, in_use_from, in_use_to)
50 VALUES ((SELECT id from owned_game WHERE client_id = (SELECT id from client
    ↪ WHERE email = 'example1@mail.ru') AND game_id = (SELECT id from game WHERE
    ↪  title = 'DOOM')), (SELECT id from machine WHERE power_tier = 3), '
    ↪ 2019-07-22 19:10:25-07', '2019-07-22 23:20:25-02'),
51 ((SELECT id from owned_game WHERE client_id = (SELECT id from client WHERE email
    ↪ = 'new@gmail.ru') AND game_id = (SELECT id from game WHERE title = 'Mario
    ↪ ')), (SELECT id from machine WHERE power_tier = 1), '2019-10-02
    ↪ 12:00:25-07', '2019-10-02 23:20:25-02'),
52 ((SELECT id from owned_game WHERE client_id = (SELECT id from client WHERE email
    ↪ = 'new@gmail.ru') AND game_id = (SELECT id from game WHERE title = '
    ↪ Trails of cold steel')), (SELECT id from machine WHERE power_tier = 2), '
    ↪ 2019-11-13 22:54:23-07', '2019-11-13 23:42:15-01');
```

Индивидуальное задание внесение изменений в структуру базы данных: добавить историю изменения цен на игры. При этом не потерять данные о текущих ценах после реорганизации структуры БД.

Для выполнения данного задания было решено удалить поле цены из таблицы game и создать дополнительную таблицу:

game_price:

- game_id - id игры;
- price - цена игры;
- price_set_date - время установления цены;
- price_end_date - время снятия цены. Если поле - NULL, значит, что цена является действующей.

Далее представлен код, который создает данную таблицу, а также переносит в нее данные о ценах игр, находящиеся в базе, а также удаляет поле price из таблицы game.

Листинг 3: Изменение структуры базы данных

```
1  CREATE TABLE game_price
2  (
3    id SERIAL PRIMARY KEY,
4    game_id bigint NOT NULL REFERENCES game(id) ON DELETE CASCADE,
5    price INTEGER,
6    price_set_date TIMESTAMP,
7    price_end_date TIMESTAMP
8  );
9
10 INSERT INTO game_price (game_id) SELECT id from game;
11 UPDATE game_price SET price = (SELECT price from game WHERE game.id = game_price
    ↪ .game_id);
12 UPDATE game_price SET price_set_date = current_timestamp;
13
14 ALTER TABLE game
15 DROP price
```

Полученная в результате выполнения работы схема базы данных показана на следующем рисунке.

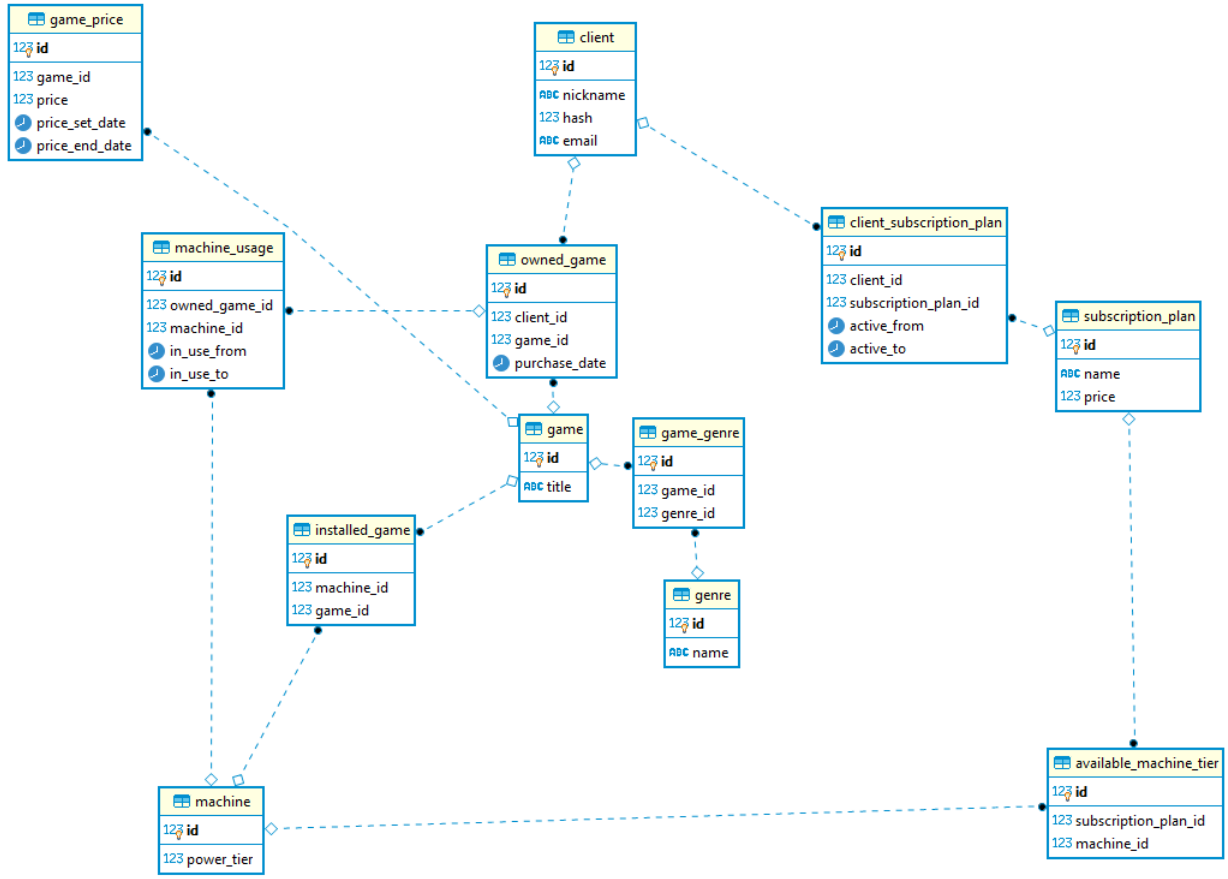

Рисунок 1.2. Схема базы данных после изменения

2. Лабораторная работа №2. Генерация тестовых данных

2.1. Цель работы

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

2.2. Программа работы

- Реализация в виде программы параметризируемого 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. Выполнение работы

Для написания генератора тестовых данных был выбран язык Kotlin. В качестве драйвера был использован JDBC. Входные данные для генератора, такие как данные для подключения к базе данных и количество записей в таблицах, которое необходимо сгенерировать, задаются в .properties файле, вид которого представлен в листинге 4.

Листинг 4: Вид файла properties

```
username=
password=
database_address=
database_name=
amount_of_clients=1000
amount_of_games=1000
amount_of_machines=1000
amount_of_owned_games=1000
amount_of_installed_games=1000
amount_of_client_subscription_plans=3
amount_of_machine_usages=300
```

Данные таблиц, не являющихся справочными, было решено генерировать случайно. Помимо чисел основные два типа данных, которые нужно генерировать это строка переменной длины и дата. Созданные для этого функции показаны в следующих листингах. В качестве входных параметров передаются длина строки и две даты, являющиеся границами генерации, соответственно.

Листинг 5: Генерация случайной строки

```
fun generateRandomString(len: Int): String {
    val sb = StringBuilder("")
    val AlphaNumericString = ("ABCDEFGHIJKLMNOPQRSTUVWXYZ"
            + "0123456789"
            + "abcdefghijklmnopqrstuvxyz")

    for (i in 0 until len) {
        val index = (0..AlphaNumericString.length - 1).random()
        sb.append(AlphaNumericString[index])
    }
    return sb.toString()
}
```

Листинг 6: Генерация случайной даты

```
fun generateRandomDate(dateFromString: String, dateToString: String): String {
    var parts = dateToString.split("-")
    val dateTo = LocalDate.of(parts[0].toInt(), parts[1].toInt(), parts[2].toInt
    ↪ ())
    parts = dateFromString.split("-")
    val dateFrom = LocalDate.of(parts[0].toInt(), parts[1].toInt(), parts[2].
    ↪ toInt())
    val randomDate =
        LocalDate.ofEpochDay(dateFrom.toEpochDay() + (1..dateTo.toEpochDay() -
    ↪ dateFrom.toEpochDay() + 1).random())
```

```
    return StringBuilder("%04d".format(randomDate.year)).append("-")
        .append("%02d".format(randomDate.monthValue))
        .append("-").append("%02d".format(randomDate.dayOfMonth)).toString()
}
```

Функция main() выполняет подключение к базе данных, выборку необходимых для генерации данных, а также функции для генерации данных для каждой таблицы. Каждая из таких функций, используя функции, представленные выше, и выбранные из базы данные генерирует новые вхождения. Код функции main() представлен дальше, а полный код находится в репозитории `http://gitlab.icc.spbstu.ru/kalashnikov.ra/videogame-streaming-ser`

Листинг 7: Функция main()

```
fun main() {
    val start = System.currentTimeMillis()
    val properties = Properties()
    val propertiesFile = System.getProperty("user.dir") + "\\file.properties"
    val inputStream = FileInputStream(propertiesFile)
    properties.load(inputStream)
    val username = properties.getProperty("username")
    val password = properties.getProperty("password")
    lateinit var conn: Connection
    val connectionProps = Properties()
    connectionProps["user"] = username
    connectionProps["password"] = password
    try {
        Class.forName("org.postgresql.Driver")
        conn = DriverManager.getConnection(
            "jdbc:" + "postgresql" + "://" +
                    properties.getProperty("database_address") + "/" +
                    properties.getProperty("database_name"),
            connectionProps
        )
    } catch (ex: SQLException) {
        ex.printStackTrace()
    } catch (ex: Exception) {
        ex.printStackTrace()
    }

    createClientEntry(conn, properties.getProperty("amount_of_clients").toInt())

    val clientIds = mutableListOf<String>()
    val statement = conn.createStatement()
    var resultSet = statement.executeQuery("SELECT id from client")
    while (resultSet.next()) {
        clientIds.add(resultSet.getString("id"))
    }

    createGameEntry(conn, properties.getProperty("amount_of_games").toInt())

    val gameIds = mutableListOf<String>()
    resultSet = statement.executeQuery("SELECT id from game")
    while (resultSet.next()) {
        gameIds.add(resultSet.getString("id"))
    }

    createMachineEntry(conn, properties.getProperty("amount_of_machines").toInt
    ↪ ())

    val machineIds = mutableListOf<String>()
    resultSet = statement.executeQuery("SELECT id, power_tier from machine")
```

```
48      while (resultSet.next()) {
49          machineIds.add(resultSet.getString("id"))
50      }
51
52      createGenreEntry(conn)
53
54      val genreIds = mutableListOf<String>()
55      resultSet = statement.executeQuery("SELECT id from genre")
56      while (resultSet.next()) {
57          genreIds.add(resultSet.getString("id"))
58      }
59
60      createOwnedGameEntry(conn, properties.getProperty("amount_of_owned_games").
    ↪ toInt(), gameIds, clientIds)
61
62      val gameWithoutGenreIds = mutableListOf<String>()
63      resultSet = statement.executeQuery("SELECT id from game WHERE id NOT IN (
    ↪ SELECT game_id from game_genre)")
64      while (resultSet.next()) {
65          gameWithoutGenreIds.add(resultSet.getString("id"))
66      }
67
68      createGameGenreEntries(conn, genreIds, gameWithoutGenreIds)
69
70      createSubscriptionPlanEntry(conn)
71
72      val machineWithoutPlanIds = mutableListOf<String>()
73      val machinePowerTiers = mutableListOf<String>()
74      resultSet =
75          statement.executeQuery("SELECT id, power_tier from machine WHERE id NOT
    ↪ IN (SELECT machine_id from available_machine_tier)")
76      while (resultSet.next()) {
77          machineWithoutPlanIds.add(resultSet.getString("id"))
78          machinePowerTiers.add(resultSet.getString("power_tier"))
79      }
80      createAvailableMachinesEntry(conn, machineWithoutPlanIds, machinePowerTiers)
81
82      createInstalledGameEntries(conn, properties.getProperty("
    ↪ amount_of_installed_games").toInt(), gameIds, machineIds)
83
84      createClientSubscriptionEntry(
85          conn,
86          properties.getProperty("amount_of_client_subscription_plans").toInt(),
87          clientIds
88      )
89
90      val clientSubPlans = mutableMapOf<String, MutableList<Triple<String, String,
    ↪  String>>>()
91      resultSet =
92          statement.executeQuery("SELECT subscription_plan_id, client_id,
    ↪ active_from, active_to from client_subscription_plan")
93      while (resultSet.next()) {
94          if (clientSubPlans[resultSet.getString("client_id")] == null) {
95              clientSubPlans[resultSet.getString("client_id")] = mutableListOf(
96                  Triple(
97                      resultSet.getString("subscription_plan_id"),
98                      resultSet.getString("active_from"),
99                      resultSet.getString("active_to")
100                 )
101             )
```

```
102             } else {
103                 clientSubPlans[resultSet.getString("client_id")]!!.add(
104                     Triple(
105                         resultSet.getString("subscription_plan_id"),
106                         resultSet.getString("active_from"),
107                         resultSet.getString("active_to")
108                     )
109                 )
110             }
111         }
112 
113 
114         val gameWithoutPriceIds = mutableListOf<String>()
115         resultSet = statement.executeQuery("SELECT id from game WHERE id NOT IN (
    ↪ SELECT game_id from game_price)")
116         while (resultSet.next()) {
117             gameWithoutPriceIds.add(resultSet.getString("id"))
118         }
119 
120         createGamePriceEntries(conn, gameWithoutPriceIds)
121 
122         val machineAvailability = mutableMapOf<String, String>()
123         resultSet = statement.executeQuery("SELECT machine_id, subscription_plan_id
    ↪  from available_machine_tier")
124         while (resultSet.next()) {
125             machineAvailability[resultSet.getString("machine_id")] = resultSet.
    ↪ getString("subscription_plan_id")
126         }
127 
128 
129         createMachineUsageEntry(
130             conn,
131             properties.getProperty("amount_of_machine_usages").toInt(),
132             machineIds,
133             machineAvailability,
134             clientSubPlans
135         )
136         System.out.println("Generation took: " + (System.currentTimeMillis() - start
    ↪ ) / 1000 + " seconds")
137 }
```

На рисунке 2.1 представлены примеры сгенерированных клиентов, а на рисунке 2.2 примеры сгенерированных игровых сессий.

	id [PK] integer	nickname character varying (30)	hash character varying (100)	email character varying (30)
6	7	SHCu83FiHRfu6YSCN1jla	1910750464	yN1uz7l@m5N.com
7	8	o5	933188220	m9zPMIP6myolEFq0kefS9t...
8	9	kfSh9	434019495	HX3ox6nAjG1u88Kh2zlWX4...
9	10	0eHNgNVDLiYAR18EuuhaR...	70481029	XmtmrRHnVGd4QqTJNn@Rl...
10	11	1tJzMnRXFuyYPzelyhW8mF	1985826953	ZYB3CWsfPnixy6ciKuGXOC...
11	12	BE	31614455	HBnkG@j6itAV5J3WQ.com
12	13	5IEPl	1085297248	7XQ0UuBEjHU5NxoM@AiFZ...
13	14	RZgPbJYWgg8QrlNsaeJ	2097806238	WuGDYRZdhhYWNtzsFS85...

Рисунок 2.1. Пример сгенерированных данных для таблицы client

	id [PK] integer	owned_game_id bigint	machine_id bigint	in_use_from timestamp without time zone	in_use_to timestamp without time zone
5	5	1187	2987	2020-02-09 06:52:22	2020-02-09 07:15:37
6	6	4852	1846	2016-04-23 08:18:20	2016-04-23 21:59:36
7	7	4708	4385	2020-10-09 10:23:24	2020-10-09 11:35:03
8	8	4597	3921	2014-04-22 20:40:34	2014-04-22 23:31:06
9	9	2161	1567	2016-08-09 13:05:54	2016-08-09 22:30:37
10	10	1401	1541	2017-08-24 04:50:08	2017-08-24 12:12:39
11	11	2025	589	2014-06-02 07:13:20	2014-06-02 16:13:22
12	12	3443	1032	2020-11-15 18:05:56	2020-11-15 23:26:25
13	13	2277	309	2019-10-17 07:47:24	2019-10-17 15:46:04

Рисунок 2.2. Пример сгенерированных данных для таблицы machine_usage

Далее представлена таблица времени генерации данных при разных параметрах для каждой таблицы. Время указано в миллисекундах без учета времени преднамеренной выборки данных.

Таблица 2.1

Длительность генерации данных

Длительность генерации	100 новых записей	1000 новых записей	10000 новых зап
amount_of_clients	60	101	848
amount_of_games	17	42	362
amount_of_machines	2	23	89
amount_of_owned_games	59	271	1237
amount_of_installed_games	7	60	814
amount_of_client_subscription_plans	35	119	931
amount_of_machine_usages	45	191	799

В следующей таблице показано полное время генерации в миллисекундах при изменении значения всех параметров, при условии, что на момент заполнения в базе данных было незначительное количество записей.

Таблица 2.2

Зависимость длительности генерации от значения всех параметров

Значение всех параметров	Время генерации
100	440
1000	1032
10000	4984
20000	11603

3. Лабораторная работа №3. Язык SQL-DML

3.1. Цель работы

Познакомить студентов 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.

3.3. Выполнение работы

Далее представлены запросы с результатами выполнения некоторых из них:

- Сделайте выборку всех данных из каждой таблицы

Листинг 8: Текст запросов

```
SELECT * FROM available_machine_tier;
SELECT * FROM client;
SELECT * FROM client_subscription_plan;
SELECT * FROM game;
SELECT * FROM game_genre;
SELECT * FROM genre;
SELECT * FROM installed_game;
SELECT * FROM machine;
SELECT * FROM machine_usage;
SELECT * FROM owned_game;
SELECT * FROM subscription_plan;
SELECT * FROM game_price;
```

- Сделайте выборку данных из одной таблицы при нескольких условиях, с использованием логических операций, LIKE, BETWEEN, IN (не менее 3-х разных примеров)

Листинг 9: Текст запросов

```
SELECT title FROM game WHERE title like 'Trails%';
SELECT * FROM game_price WHERE price BETWEEN 1000 AND 3000;
SELECT * FROM game WHERE title like 'Trails%' AND (SELECT price FROM
    ↪ game_price WHERE game.id = game_id) in (1000, 2000, 3000);
```

	title character varying (30)
1	Trails of cold steel

Рисунок 3.1. Результат выполнения первого запроса

	id [PK] integer	game_id bigint	price integer	price_set_date timestamp without time zone	price_end_date timestamp without time zone
1	1	1	2000	2020-05-28 23:55:13.451032	[null]
2	3	3	1000	2020-05-28 23:55:13.451032	[null]
3	6	6	2880	2019-05-03 00:00:00	[null]
4	7	7	2220	2019-08-14 00:00:00	[null]
5	12	12	1190	2010-10-11 00:00:00	[null]
6	15	15	2650	2018-01-05 00:00:00	[null]
7	21	21	2060	2018-05-29 00:00:00	[null]
8	24	24	1530	2010-05-05 00:00:00	[null]

Рисунок 3.2. Результат выполнения второго запроса

	id [PK] integer	title character varying (30)
1	3	Trails of cold steel

Рисунок 3.3. Результат выполнения третьего запроса

- Создайте в запросе вычисляемое поле

Листинг 10: Текст запросов

```
SELECT MAX(price) FROM game_price;
SELECT AVG(price) FROM game_price WHERE price_end_date IS NULL;
SELECT COUNT(in_use_FROM) FROM machine_usage WHERE in_use_FROM BETWEEN '
    ↪ 2019-07-21' AND '2019-12-1';
```

	max integer
1	8000

Рисунок 3.4. Результат выполнения первого запроса

	avg numeric
1	4022.3365980411752948

Рисунок 3.5. Результат выполнения второго запроса

	count bigint
1	243

Рисунок 3.6. Результат выполнения третьего запроса

- Сделайте выборку всех данных с сортировкой по нескольким полям

Листинг 11: Текст запросов

```
SELECT * FROM machine_usage
ORDER BY in_use_FROM DESC, in_use_to ASC;
SELECT * FROM client
ORDER BY nickname ASC, email DESC;
```

	id [PK] integer	owned_game_id bigint	machine_id bigint	in_use_from timestamp without time zone	in_use_to timestamp without time zone
1	4760	4569	4511	2023-12-01 21:38:31	2023-12-01 22:24:11
2	3724	3089	1651	2023-11-08 18:39:12	2023-11-08 23:12:46
3	2224	3276	1603	2023-11-02 00:01:17	2023-11-02 22:41:16
4	495	2838	3706	2023-10-22 03:33:00	2023-10-22 11:20:06
5	1676	3261	1424	2023-10-10 03:26:35	2023-10-10 16:30:24
6	3808	3413	3629	2023-10-07 13:22:39	2023-10-07 23:27:48
7	3607	1058	3255	2023-09-28 22:45:21	2023-09-28 23:38:15
8	1128	2666	4401	2023-09-28 04:04:55	2023-09-28 05:33:37

Рисунок 3.7. Результат выполнения первого запроса

	id [PK] integer	nickname character varying (30)	hash character varying (100)	email character varying (30)
1	3636	0	483417951	Z41DAIWORn@S.com
2	774	0	1150189287	SymzYgg9W3@ASVIWqi2P...
3	3016	0	1379157090	pWMcsI7@mHX2sUJI8q2ha...
4	3400	0	1558800462	NObOK5PV@GxCsQhi5i.com
5	62	0	1974961356	IhfgIxWUsUTc@GNEsM71W...
6	671	0	682472930	ciG@frNQRx5onnj.com
7	1413	0	1141584660	8LKgBOL41vJQc3amHsUjA...
8	212	0	1813657237	3lv9VtDR@JO3fx0ke.com

Рисунок 3.8. Результат выполнения второго запроса

- Создайте запрос, вычисляющий несколько совокупных характеристик таблиц

Листинг 12: Текст запросов

```
SELECT AVG(price) AS avg_price, MIN(price) AS min_price, MAX(price) AS
    ↪ max_price FROM game_price WHERE price_end_date IS NULL;
```

	avg_price numeric	min_price integer	max_price integer
1	4022.3365980411752948	200	8000

Рисунок 3.9. Результат выполнения запроса

- Сделайте выборку данных из связанных таблиц (не менее двух примеров). Для соединения таблиц в следующих запросах используется INNER JOIN

Листинг 13: Текст запросов

```
SELECT game.title, game_price.price
  FROM game
  INNER JOIN game_price ON game.id = game_price.game_id;

SELECT client.nickname, game.title
  FROM owned_game
  INNER JOIN client ON owned_game.client_id = client.id
  INNER JOIN game ON owned_game.game_id = game.id;
```

	title character varying (30)	price integer
1	DOOM	2000
2	Mario	4500
3	Trails of cold steel	1000
4	1xt9IGlNfAp	7250
5	25QlYHjpvjQSqAbgX	7190
6	LbClcvhtd9ag	2880
7	JxUtzW59VGB	2220
8	NmLjZdVxKlH3NZDuRmeN4...	5220

Рисунок 3.10. Результат выполнения первого запроса

	nickname character varying (30)	title character varying (30)
1	roll	DOOM
2	new_client	Mario
3	new_client	Trails of cold steel
4	YFvhghcjJtzPrLo9	aKcX0eE4JVh
5	z5YhdifcTQAyIU	ZcCWcxn0xmPAEQtuXMaB4...
6	EUQfH83gtyktJ7cJym8xvpY...	vDoEnskykYc8Y76bKYB2AD...
7	z7A7dqqSRPD5BgqYYdQ2iX...	HOzoxTmzuHHj2t
8	Og0lfcdcbkJy9DuKXn9yAXU	mGdgKn1RbtTxRKYLh1RrPZ...

Рисунок 3.11. Результат выполнения второго запроса

- Создайте запрос, рассчитывающий совокупную характеристику с использованием группировки, наложите ограничение на результат группировки

Листинг 14: Текст запросов

```
SELECT client.nickname, COUNT(game.title)
  FROM owned_game
  INNER JOIN client ON  owned_game.client_id = client.id
  INNER JOIN game ON owned_game.game_id = game.id
  GROUP BY nickname HAVING COUNT(title) >= 1
  ORDER BY COUNT(title) DESC;
```

	nickname character varying (30)	count bigint
1	8	13
2	lel	11
3	destroyer1337	10
4	c	8
5	t	7
6	e	7
7	E	7
8	r	7

Рисунок 3.12. Результат выполнения запроса

- Придумайте и реализуйте пример использования вложенного запроса

Листинг 15: Текст запросов

```
SELECT * FROM machine_usage
  WHERE owned_game_id IN (SELECT id FROM owned_game
                WHERE game_id in (SELECT game_id FROM game_genre
                          WHERE genre_id = (SELECT id FROM genre
                                    WHERE name = 'action'))));
```

	id [PK] integer	owned_game_id bigint	machine_id bigint	in_use_from timestamp without time zone	in_use_to timestamp without time zone
1	1	1	3	2019-07-22 19:10:25	2019-07-22 23:20:25
2	4	522	2010	2019-08-28 15:58:12	2019-08-28 16:58:50
3	5	1187	2987	2020-02-09 06:52:22	2020-02-09 07:15:37
4	6	4852	1846	2016-04-23 08:18:20	2016-04-23 21:59:36
5	8	4597	3921	2014-04-22 20:40:34	2014-04-22 23:31:06
6	9	2161	1567	2016-08-09 13:05:54	2016-08-09 22:30:37
7	13	2277	309	2019-10-17 07:47:24	2019-10-17 15:46:04
8	14	1281	3884	2012-01-03 10:38:34	2012-01-03 13:00:02

Рисунок 3.13. Результат выполнения запроса

- С помощью оператора INSERT добавьте в каждую таблицу по одной записи

Листинг 16: Текст запросов

```
INSERT INTO client (nickname, hash, email)
  VALUES ('new_client', 33, 'new_cl@gmail.ru');
INSERT INTO game (title)
  VALUES ('new_game');
INSERT INTO genre (name)
  VALUES ('new_genre');
INSERT INTO machine (power_tier)
  VALUES (1);
INSERT INTO subscription_plan (name, price)
  VALUES ('premium-tier plan', 1000);
INSERT INTO available_machine_tier (subscription_plan_id, machine_id)
  VALUES ((SELECT id from subscription_plan WHERE name = 'premium-tier plan
    ↪ '), (SELECT id from machine WHERE power_tier = 3));
INSERT INTO client_subscription_plan (client_id, subscription_plan_id,
    ↪ active_from, active_to)
  VALUES ((SELECT id from client WHERE email = 'new_cl@gmail.ru'), (SELECT
    ↪ id from subscription_plan WHERE name = 'low-tier plan'), '2019-07-21'
    ↪ , '2020-07-21');
INSERT INTO game_genre (game_id, genre_id)
  VALUES ((SELECT id from game WHERE title = 'new_game'), (SELECT id from
    ↪ genre WHERE name = 'new_genre'));
INSERT INTO installed_game (machine_id, game_id)
  VALUES ((SELECT id from machine WHERE power_tier = 3 LIMIT 1), (SELECT id
    ↪  from game WHERE title = 'new_game'));
INSERT INTO owned_game (client_id, game_id, purchase_date)
  VALUES ((SELECT id from client WHERE email = 'new_cl@gmail.ru'), (SELECT
    ↪ id from game WHERE title = 'new_game'), '2019-07-03');
INSERT INTO machine_usage (owned_game_id, machine_id, in_use_from,
    ↪ in_use_to)
  VALUES ((SELECT id from owned_game WHERE client_id = (SELECT id from
    ↪ client WHERE email = 'new_cl@gmail.ru') AND game_id = (SELECT id from
    ↪  game WHERE title = 'new_game')), (SELECT id from machine WHERE
    ↪ power_tier = 3 LIMIT 1), '2019-08-12 19:10:25-07', '2019-08-12
    ↪ 23:20:25-02');
INSERT INTO game_price (game_id, price, price_set_date, price_end_date)
  VALUES ((SELECT id from game WHERE title = 'new_game'), 1000, '2015-01-12
    ↪ 00:00:00-00', null);
```

- С помощью оператора UPDATE измените значения нескольких полей у всех записей, отвечающих заданному условию

Листинг 17: Текст запросов

```
UPDATE client
  SET nickname = 'changed_nickname', hash = 111111
    WHERE email = 'new_cl@gmail.ru';
UPDATE machine_usage
  SET in_use_from = in_use_from + INTERVAL'1 hour', in_use_to = in_use_to +
    ↪ INTERVAL'1 hour'
    WHERE machine_id = 3;
```

- С помощью оператора DELETE удалите запись, имеющую максимальное (минимальное) значение некоторой совокупной характеристики

Листинг 18: Текст запросов

```
DELETE FROM machine WHERE id = (SELECT machine_id
                  FROM (SELECT machine_id, count(*) AS usage_count
                        FROM machine_usage
                        GROUP BY machine_id
                        ORDER BY usage_count ASC LIMIT 1) AS
    ↪ least_used_machine_id);

DELETE FROM machine WHERE id = (SELECT machine_id
                  FROM (SELECT machine_id, MIN(usage_count)
                        FROM (SELECT machine_id, count(*) AS usage_count
                              FROM machine_usage
                              GROUP BY machine_id) AS usage_counted
                        GROUP BY machine_id ) AS least_used_id);
```

- С помощью оператора DELETE удалите записи в главной таблице, на которые не ссылается подчиненная таблица (используя вложенный запрос)

Листинг 19: Текст запросов

```
DELETE FROM game
  WHERE id NOT IN (SELECT DISTINCT game_id
            FROM game_price);

DELETE FROM machine
  WHERE id NOT IN (SELECT DISTINCT machine_ID
           FROM available_machine_tier);
```

Первое индивидуальное задание: Для каждого пользователя вывести, сколько денег он потратил на этом сервисе (на приобретение игр и на оплату подписок).

Для решение данного задания было решено сначала отдельно подсчитать количество денег, потраченное на игры и подписки, для каждого клиента, а затем сложить их. Полученный код представлен в следующем листинге.

Листинг 20: Индивидуальное задание 1

```
WITH money_spent_on_sub(email, money_spent) AS (
  SELECT email, SUM(((date_part('day', active_to::timestamp - active_from::
    ↪ timestamp) / 31) + 1)::integer * price) as money_spent
  FROM (SELECT client.email, subscription_plan.price, client_subscription_plan.
    ↪ active_from, client_subscription_plan.active_to
      FROM client_subscription_plan
      INNER JOIN client ON client_subscription_plan.client_id = client.id
        INNER JOIN subscription_plan ON client_subscription_plan.
    ↪ subscription_plan_id = subscription_plan.id) AS sub_plans
      GROUP BY email),
```

```
8  money_spent_on_games (email, money_spent) AS (
9    SELECT email, SUM(price) as money_spent
10   FROM (SELECT client.email, game_price.price
11         FROM owned_game
12         INNER JOIN client ON owned_game.client_id = client.id
13         INNER JOIN game_price ON owned_game.game_id = game_price.game_id AND ((
       ↪ owned_game.purchase_date BETWEEN game_price.price_set_date AND game_price.
       ↪ price_end_date)
14                                                 OR (owned_game.purchase_date >=
       ↪ game_price.price_set_date AND game_price.price_end_date is NULL))) AS
       ↪ game_prices
15         GROUP BY email)
16 SELECT COALESCE(money_spent_on_sub.email, money_spent_on_games.email) as
       ↪ client_email, COALESCE(money_spent_on_sub.money_spent, 0) + COALESCE(
       ↪ money_spent_on_games.money_spent, 0) as total_money_spent
17 FROM money_spent_on_sub
18 FULL JOIN money_spent_on_games ON money_spent_on_sub.email =
       ↪ money_spent_on_games.email
```

Результат выполнения запроса показан на рисунке 3.14.

	client_email character varying (30)	total_money_spent bigint
1	Mgt@v0rL5s.com	14270
2	cfPheVV7jtc9jmJr@8XZICa...	33800
3	DUSxtMde@OKSh5SGa4.com	5290
4	4@pkUKCmnt2GX.com	51700
5	pWdEHPZ@xHglbjT.com	11840
6	LUTFGDWZ@sD1iMT2iDJdy....	28310
7	KiFXXCZksaHRdfCpLWTDyo...	11200
8	zM@c2Qizi.com	11400
9	4aCOeAMJCj8Z@DG9T3A1...	35100

Рисунок 3.14. Результат выполнения запроса

Второе индивидуальное задание: Для каждого дня недели вывести среднюю загруженность машин за последний месяц.

Для решения данного задания было решено сначала посчитать время проведенное за компьютерами в каждый день месяца, а затем получить среднее значение по дням недели и поделить его на общее количество компьютеров.

Листинг 21: Индивидуальное задание 2

```
1  WITH usage_per_dow (day_of_month, day_of_week, second_sum) AS (
2    SELECT date_part('day', in_use_from) as day_of_month, date_part('dow',
       ↪ in_use_from) as day_of_week, SUM(EXTRACT(EPOCH FROM in_use_to) - EXTRACT(
       ↪ EPOCH from in_use_from)) from machine_usage
3    INNER JOIN machine ON machine_usage.machine_id = machine.id
4    WHERE (in_use_from >= current_date - interval '1 month' AND in_use_to <=
       ↪ current_date)
5    GROUP BY day_of_month, day_of_week
6  ),
7  machine_count (amount_of_machines) AS (
8    SELECT COUNT(id) FROM machine
9  )
10 SELECT day_of_week, ROUND(AVG((second_sum / 3600 / amount_of_machines))::numeric
       ↪ , 10) as hours_occupied
11 FROM usage_per_dow, machine_count
12 GROUP BY day_of_week
13 ORDER BY day_of_week
```

Результат выполнения запроса показан на рисунке 3.15.

	day_of_week double precision	hours_occupied numeric
1	0	0.0021669221
2	1	0.0031102866
3	2	0.0015489734
4	3	0.0019899172
5	4	0.0010836970
6	5	0.0020844993
7	6	0.0031644208

Рисунок 3.15. Результат выполнения запроса

Третье индивидуальное задание: Вывести игры, популярность которых росла на протяжении трех или более месяцев. Популярность игры считать по количеству часов, в которые в нее играли.

Для решения данного задания было решено сначала подсчитать время сыгранное в игры для каждого месяца, а затем, используя оконную функцию, создать выборку где в каждой строчке есть значения времени игры для трех последовательных месяцев, а затем из этой выборки выбрать подходящие игры.

Листинг 22: Индивидуальное задание 3

```
WITH game_played_time(title, minutes_played, month_played) AS (
  SELECT game.title, SUM((date_part('hour', machine_usage.in_use_to) * 60 +
    ↪ date_part('minute', machine_usage.in_use_to)
            - date_part('hour', machine_usage.in_use_from) * 60 - date_part('
    ↪ minute', machine_usage.in_use_from))) as minutes_played,
          date_part('month',machine_usage.in_use_to) + date_part('year',
    ↪ machine_usage.in_use_to) * 12 as month_from_beginning_played
  FROM owned_game
  RIGHT JOIN machine_usage ON owned_game.id = machine_usage.owned_game_id
  INNER JOIN game ON owned_game.game_id = game.id
  GROUP BY game.title, month_from_beginning_played
  ORDER BY game.title, month_from_beginning_played
),
cte2 AS (SELECT title, minutes_played, month_played, LAG(minutes_played, 1) OVER
    ↪  (PARTITION BY title ORDER BY month_played) as previous_month_playtime,
    LAG(minutes_played, 2) OVER (PARTITION BY title ORDER BY month_played) as
    ↪ pre_previous_month_playtime
FROM game_played_time)
SELECT DISTINCT title FROM cte2
WHERE previous_month_playtime > pre_previous_month_playtime AND minutes_played >
    ↪  previous_month_playtime
```

Результат выполнения запроса показан на рисунке 3.16.

	title character varying (30)
1	091gXEUpTFK01TcKcxo
2	0FDWCIcob5M7
3	0J6TbJOWRJxHMvdqc5A46...
4	0V8JXagfojBnst6b6hesU9
5	1ct6igoFfCEcLn5STFT0iOc...
6	21Azdd8it4x0VFhqokzW5y
7	25QlYHjpvjQSqAbgX
8	27OI
9	2rfVfWdzIa9MrfGVioTPFH6

Рисунок 3.16. Результат выполнения запроса

4. Лабораторная работа №4. Нагрузка базы данных и оптимизация запросов

4.1. Цель работы

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

4.2. Программа работа

- Написание параметризированных 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. Выполнение работы

Для тестирование были написаны 5 параметризированных запросов, которые пользователи могут использовать в реальной ситуации: получение цены на конкретную игру, получения списка игр, в которые сыграно более определенного числа минут, подсчет количества игр для конкретного клиента, подсчет количества пользователей, купивших конкретную игру и получение списка планов по подписке конкретного пользователя. Массив с данными запросами показан в листинге 23.

Листинг 23: Запросы для тестирования

```
1     private val queries = listOf(
2         "SELECT␣game.title,␣game_price.price\n" +
3                 "FROM␣game_price\n" +
4                 "INNER␣JOIN␣game␣ON␣game.id␣=␣game_price.id␣AND␣game_price.
    ↪ price_end_date␣IS␣NULL\n" +
5                 "WHERE␣game.title␣=␣",
6         "SELECT␣game.title\n" +
7                 "FROM␣owned_game\n" +
```

```
8               "RIGHT JOIN machine_usage ON owned_game.id = machine_usage.
    ↪ owned_game_id AND in_use_from > current_date - interval '1 month'\n" +
9               "INNER JOIN game ON owned_game.game_id = game.id\n" +
10              "GROUP BY game.title HAVING SUM((date_part('hour', machine_usage
    ↪ .in_use_to) * 60 + date_part('minute', machine_usage.in_use_to) -
    ↪ date_part('hour', machine_usage.in_use_from) * 60 - date_part('minute',
    ↪ machine_usage.in_use_from))) / 60 > ",
11        "SELECT COUNT(game.title), client_id\n" +
12              "FROM owned_game\n" +
13              "INNER JOIN game ON owned_game.game_id = game.id\n" +
14              "WHERE client_id = ",
15        "SELECT game.title, COUNT(client_id) as amount_sold\n" +
16              "FROM owned_game\n" +
17              "INNER JOIN game ON owned_game.game_id = game.id\n" +
18              "WHERE game.title = ",
19        "SELECT client.nickname, subscription_plan.name,
    ↪ client_subscription_plan.active_from, client_subscription_plan.active_to\n
    ↪ " +
20              "FROM client_subscription_plan\n" +
21              "INNER JOIN client ON client_id = client.id\n" +
22              "INNER JOIN subscription_plan ON subscription_plan_id =
    ↪ subscription_plan.id\n" +
23              "WHERE client.email = "
24    )
```

Для отправления запросов базе была создана функция(листинг 24), которая создает потоки с подключениями к базе данных, каждый из потоков выполняет определенное количество запросов, при этом начинают их выполнение они одновременно. Данная функция возвращает массив, состоящий из длительности выполнения всех запросов каждым потоком.

Листинг 24: Формирование потоков, отправляющих запросы

```
1   fun attackBd(conn: Connection, amountOfThreads: Int, amountOfQueries: Int):
    ↪ List<Deferred<Long>> {
2       val results = mutableListOf<Deferred<Long>>()
3
4       val statement = conn.createStatement()
5       var resultSet = statement.executeQuery("SELECT title FROM game")
6       while (resultSet.next()) {
7           gameTitles.add(resultSet.getString("title"))
8       }
9
10      resultSet = statement.executeQuery("SELECT id, email FROM client")
11      while (resultSet.next()) {
12          clientIds.add(resultSet.getString("id"))
13          clientEmails.add(resultSet.getString("email"))
14      }
15
16      if (isIndex) {
17          statement.executeUpdate("CREATE INDEX ${indexesNames[0]} ON game(
    ↪ title)")
18          statement.executeUpdate("CREATE INDEX ${indexesNames[1]} ON client(
    ↪ email)")
19          statement.executeUpdate("CREATE INDEX ${indexesNames[2]} ON
    ↪ owned_game(game_id, client_id)")
20          statement.executeUpdate("CREATE INDEX ${indexesNames[3]} ON
    ↪ client_subscription_plan(client_id, subscription_plan_id)")
21          statement.executeUpdate("CREATE INDEX ${indexesNames[4]} ON
    ↪ machine_usage(owned_game_id)")
22          isIndex = false
```

```
23          }
24
25          val connections = mutableListOf<Connection>()
26          for (i in 1..amountOfThreads) {
27              val username = "postgres"
28              val password = "r177"
29              lateinit var conn2: Connection
30              val connectionProps = Properties()
31              connectionProps["user"] = username
32              connectionProps["password"] = password
33              try {
34                  conn2 = DriverManager.getConnection(
35                      "jdbc:" + "postgresql" + "://" +
36                              "127.0.0.1:5432" + "/" +
37                              "streaming_service",
38                      connectionProps
39                  )
40              } catch (ex: SQLException) {
41                  ex.printStackTrace()
42              } catch (ex: Exception) {
43                  ex.printStackTrace()
44              }
45              connections.add(conn2)
46          }
47          for (i in 1..amountOfThreads) {
48              results.add(async(coroutineContext) {
49                  oneThreadAttack(amountOfQueries, connections[i - 1])
50              })
51          }
52
53          return results
54      }
55
56      private fun oneThreadAttack(amountOfQueries: Int, conn: Connection): Long {
57          val statement = conn.createStatement()
58          if (isPrepare) {
59              for (i in 0 until queries.size) {
60                  when (i) {
61                      0, 4 -> statement.executeUpdate("PREPARE␣" + queryPlanNames[
    ↪ i] + "(text)␣AS\n" + queries[i] + "$1")
62                      1 -> statement.executeUpdate("PREPARE␣" + queryPlanNames[i]
    ↪ + "(int)␣AS\n" + queries[i] + "$1")
63                      2 -> statement.executeUpdate("PREPARE␣" + queryPlanNames[i]
    ↪ + "(int)␣AS\n" + queries[i] + "$1\nGROUP␣BY␣client_id")
64                      else -> statement.executeUpdate("PREPARE␣" + queryPlanNames[
    ↪ i] + "(text)␣AS\n" + queries[i] + "$1\nGROUP␣BY␣game.title")
65                  }
66              }
67          }
68          val timesList = mutableListOf<Long>()
69          for (j in 1..amountOfQueries) {
70              val query: String = if (!isPrepare) {
71                  when (j % queries.size) {
72                      0 -> queries[j % queries.size] + "'" + gameTitles[(0 until
    ↪ gameTitles.size).random()] + "'"
73                      1 -> queries[j % queries.size] + (0..100).random().toString
    ↪ ()
74                      2 -> queries[j % queries.size] + clientIds[(0 until
    ↪ clientIds.size).random()] + "\nGROUP␣BY␣client_id"
75                      3 -> queries[j % queries.size] + "'" + gameTitles[(0 until
```

```
    ↪ gameTitles.size).random()] + "'\nGROUP␣BY␣game.title"
76                  else -> queries[j % queries.size] + "'" + clientEmails[(0
    ↪ until clientEmails.size).random()] + "'"
77              }
78          } else {
79              "EXECUTE␣" + when (j % queries.size) {
80                  0, 3 -> queryPlanNames[j % queries.size] + "('" + gameTitles
    ↪ [(0 until gameTitles.size).random()] + "')"
81                  1 -> queryPlanNames[j % queries.size] + "(" + (0..100).
    ↪ random().toString() + ")"
82                  else -> queryPlanNames[j % queries.size] + "(" + clientIds
    ↪ [(0 until clientIds.size).random()] + ")"
83              }
84          }
85          val start = currentTimeMillis()
86          statement.executeQuery(query)
87          timesList.add(currentTimeMillis() - start)
88      }
89      conn.close()
90      return timesList.average().toLong()
91  }
92 }
```

Вызов представленной выше функции происходит в функции main(). Она вызывается с различными параметрами количества потоков и запросов, а также с разной настройкой оптимизации. Значения, полученные в результате вызова функции attackBd() в реальном времени отображаются на графике, а также записываются в файл. Код данной части представлен в листинге 25.

Листинг 25: Код для тестирования базы данных

```
1 ЭкспоненциальныйростКоличествопотоковштДлительностьвыполнениязапросамсКвадратичныйростКоличествопоток
2       launch {
3           val statement = conn.createStatement()
4           for (indexName in attacker.indexesNames) {
5               statement.executeUpdate("DROP␣INDEX␣IF␣EXISTS␣$indexName")
6           }
7           //Графики без оптимизации
8           //Экспоненциальный рост
9           var amountOfThreads = 1
10          var dynamicY = Array(xThreadsExp.size) { _ -> 0.0 }
11
12          while (amountOfThreads <= xThreadsExp.size) {
13              val listOfTimes =
14                  attacker.attackBd(conn, exp(amountOfThreads - 1.0).toInt(),
    ↪ amountOfQueries).awaitAll()
15              dynamicY[amountOfThreads - 1] =
16                  BigDecimal(listOfTimes.average()).setScale(2, RoundingMode.
    ↪ HALF_EVEN).toDouble()
17              val expMap = mutableMapOf<Int, Double>()
18              for (i in 0 until xThreadsExp.size) {
19                  expMap[xThreadsExp[i]] = dynamicY[i]
20              }
21              traceAmountOfThreadsExp.y = xThreadsExp.map { i -> expMap[i] }
22              amountOfThreads++
23          }
24          //Квадратичный рост
25          amountOfThreads = 1
26          dynamicY = Array(xThreadsSquare.size) { _ -> 0.0 }
27
28          while (amountOfThreads <= xThreadsSquare.size) {
```

```
29                  val listOfTimes = attacker.attackBd(conn, amountOfThreads *
    ↪ amountOfThreads, amountOfQueries).awaitAll()
30                  dynamicY[amountOfThreads - 1] =
31                      BigDecimal(listOfTimes.average()).setScale(2, RoundingMode.
    ↪ HALF_EVEN).toDouble()
32                  traceAmountOfThreadsSquare.y = xThreadsSquare.map { i ->
    ↪ dynamicY[sqrt(i.toDouble()).toInt() - 1] }
33                  amountOfThreads++
34              }
35
36              //Линейный рост
37              amountOfThreads = 1
38              dynamicY = Array(20) { _ -> 0.0 }
39
40              while (amountOfThreads <= 20) {
41                  val listOfTimes = attacker.attackBd(conn, amountOfThreads * 2,
    ↪ amountOfQueries).awaitAll()
42                  dynamicY[amountOfThreads - 1] =
43                      BigDecimal(listOfTimes.average()).setScale(2, RoundingMode.
    ↪ HALF_EVEN).toDouble()
44                  traceAmountOfThreads.y = xThreads.map { i -> dynamicY[i / 2 - 1]
    ↪  }
45                  amountOfThreads++
46              }
47              //Запись значений в файл
48              var resultsStr = StringBuilder("")
49              for (dynamicYelement in dynamicY) {
50                  resultsStr.append("$dynamicYelement\n")
51
52              }
53              File("linear.txt").writeText(resultsStr.toString())
54
55              //Оптимизация CREATE INDEX
56              attacker.isIndex = true
57
58              val normalValues = dynamicY
59              amountOfThreads = 1
60              dynamicY = Array(20) { _ -> 0.0 }
61
62              while (amountOfThreads <= 20) {
63                  val listOfTimes = attacker.attackBd(conn, amountOfThreads * 2,
    ↪ amountOfQueries).awaitAll()
64                  dynamicY[amountOfThreads - 1] =
65                      BigDecimal(listOfTimes.average()).setScale(2, RoundingMode.
    ↪ HALF_EVEN).toDouble()
66                  traceAmountOfThreadsIndex.y = xThreads.map { i -> dynamicY[i / 2
    ↪  - 1] }
67                  traceAmountOfThreadsIndexDiff.y = xThreads.map { i ->
    ↪ normalValues[i / 2 - 1] - dynamicY[i / 2 - 1] }
68                  amountOfThreads++
69              }
70              //Запись значений в файл
71              resultsStr = StringBuilder("")
72              for (dynamicYelement in dynamicY) {
73                  resultsStr.append("$dynamicYelement\n")
74
75              }
76              File("index.txt").writeText(resultsStr.toString())
77
78              for (indexName in attacker.indexesNames) {
```

```
79                  statement.executeUpdate("DROP INDEX IF EXISTS $indexName")
80              }
81
82              //Оптимизация PREPARE
83              attacker.isPrepare = true
84
85              amountOfThreads = 1
86              dynamicY = Array(20) { _ -> 0.0 }
87
88              while (amountOfThreads <= 20) {
89                  val listOfTimes = attacker.attackBd(conn, amountOfThreads * 2,
    ↪ amountOfQueries).awaitAll()
90                  dynamicY[amountOfThreads - 1] =
91                      BigDecimal(listOfTimes.average()).setScale(2, RoundingMode.
    ↪ HALF_EVEN).toDouble()
92                  traceAmountOfThreadsPrepared.y = xThreads.map { i -> dynamicY[i
    ↪ / 2 - 1] }
93                  traceAmountOfThreadsPreparedDiff.y = xThreads.map { i ->
    ↪ normalValues[i / 2 - 1] - dynamicY[i / 2 - 1] }
94                  amountOfThreads++
95              }
96              //Запись значений в файл
97              resultsStr = StringBuilder("")
98              for (dynamicYelement in dynamicY) {
99                  resultsStr.append("$dynamicYelement\n")
100
101             }
102             File("prepare.txt").writeText(resultsStr.toString())
103
104             statement.executeUpdate("DEALLOCATE ALL")
105             attacker.isPrepare = false
106
107             //Оптимизация и INDEX и PREPARE
108             attacker.isPrepare = true
109             attacker.isIndex = true
110
111             amountOfThreads = 1
112             dynamicY = Array(20) { _ -> 0.0 }
113
114             while (amountOfThreads <= 20) {
115                 val listOfTimes = attacker.attackBd(conn, amountOfThreads * 2,
    ↪ amountOfQueries).awaitAll()
116                 dynamicY[amountOfThreads - 1] =
117                     BigDecimal(listOfTimes.average()).setScale(2, RoundingMode.
    ↪ HALF_EVEN).toDouble()
118                 traceAmountOfThreadsBoth.y = xThreads.map { i -> dynamicY[i / 2
    ↪ - 1] }
119                 traceAmountOfThreadsBothDiff.y = xThreads.map { i ->
    ↪ normalValues[i / 2 - 1] - dynamicY[i / 2 - 1] }
120                 amountOfThreads++
121             }
122             //Запись значений в файл
123             resultsStr = StringBuilder("")
124             for (dynamicYelement in dynamicY) {
125                 resultsStr.append("$dynamicYelement\n")
126
127             }
128             File("both.txt").writeText(resultsStr.toString())
129
130             statement.executeUpdate("DEALLOCATE ALL")
```

```
131             attacker.isPrepare = false
132
133             for (indexName in attacker.indexesNames) {
134                 statement.executeUpdate("DROP INDEX IF EXISTS $indexName")
135             }
136         }
```

Тестирование проводилось для четырех ситуаций:

- Оптимизация отсутствует;
- Используется оптимизация Prepare(строки 59-66 листинга 24);
- Используется оптимизация Index(строки 17-21 листинга 24);
- Используются обе вышеуказанные оптимизации.

Тестирование реакции базы данных на нагрузку до и после оптимизации было протестировано для ситуации, когда таблицы в базе имеют приблизительно 5000 записей, а также ситуации, когда таблицы в базе имеют более 60000 записей. На следующих рисунках показаны результаты тестирования для первой ситуации. На первых трех графиках показана зависимость средней длительности одного запроса от количества потоков, одновременно подающих запросы. Количество потоков меняется с разной скоростью, а значит и скорость изменения нагрузки меняется. На следующих трех графиках слева показана зависимость длительности выполнения одного запроса от количества потоков, а справа разница между обычным временем и временем выполнения запроса с конкретной оптимизацией.

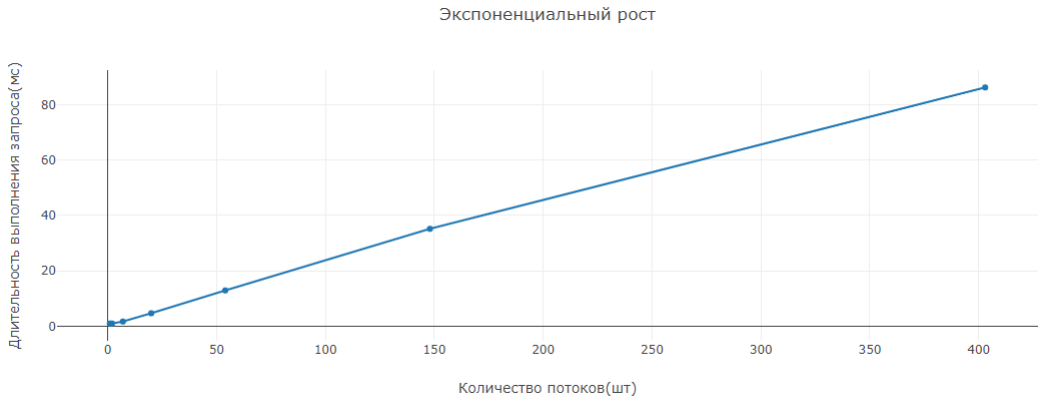

Рисунок 4.1. Возрастание нагрузки экспоненциально

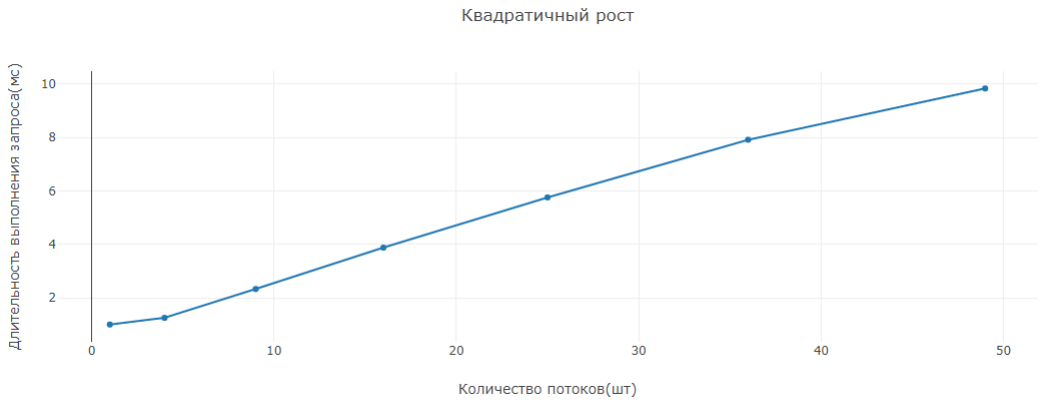

Рисунок 4.2. Возрастание нагрузки квадратично

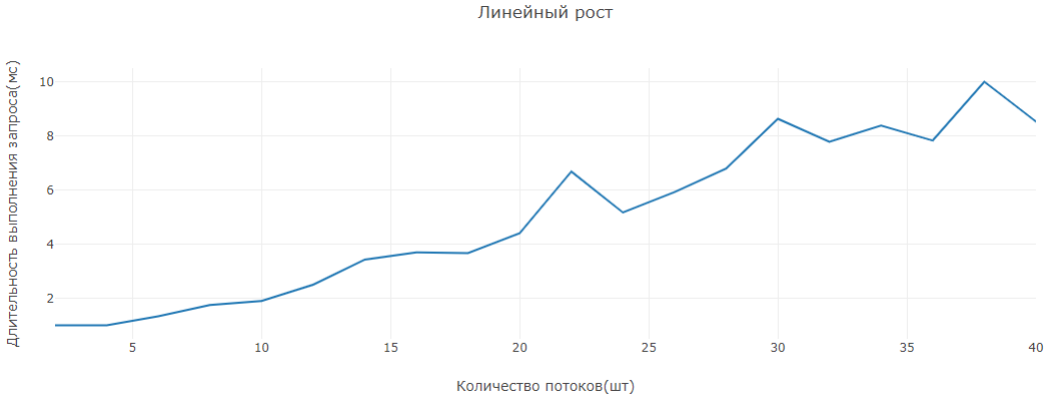

Рисунок 4.3. Возрастание нагрузки линейно

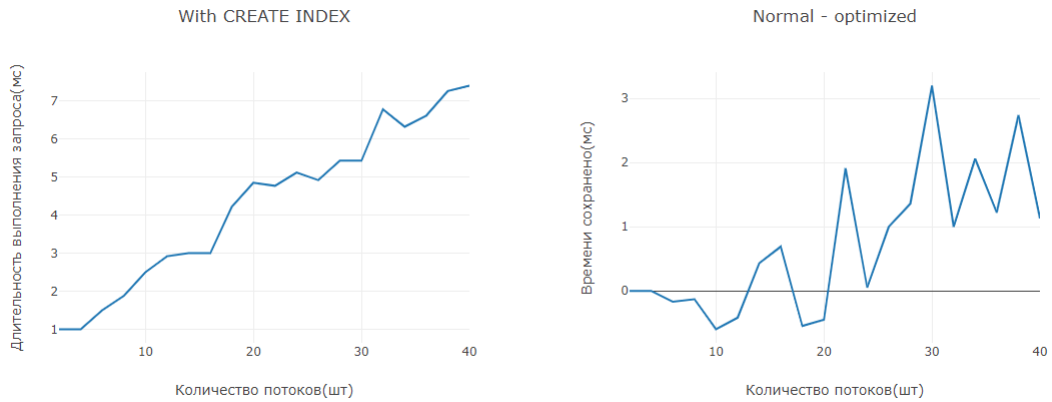

Рисунок 4.4. График после оптимизации с использованием Index

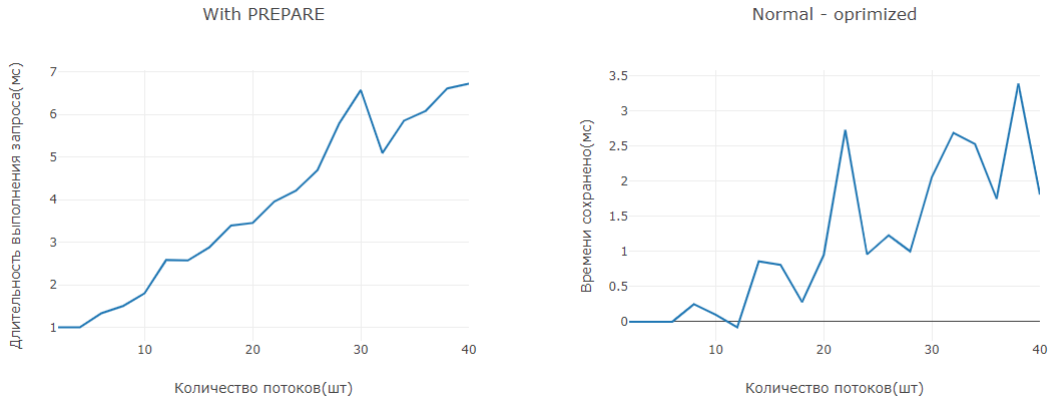

Рисунок 4.5. График после оптимизации с использованием Prepare

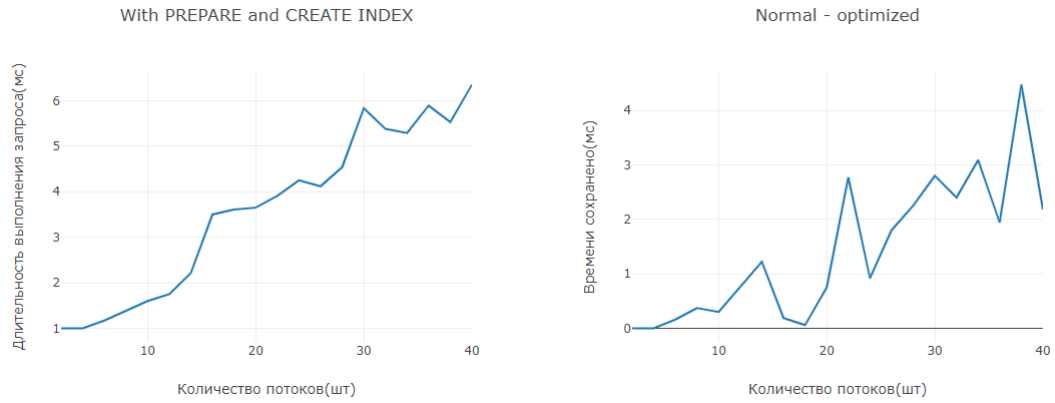

Рисунок 4.6. График после оптимизации с использованием Prepare и Index

Как видно, оба способа оптимизации улучшили длительность обработки запроса, пусть и не на много. Наиболее эффективным оказалось использование обоих способов оптимизации.

На рисунке 4.7 показаны графики длительности выполнения запроса, построенные на одном графике.

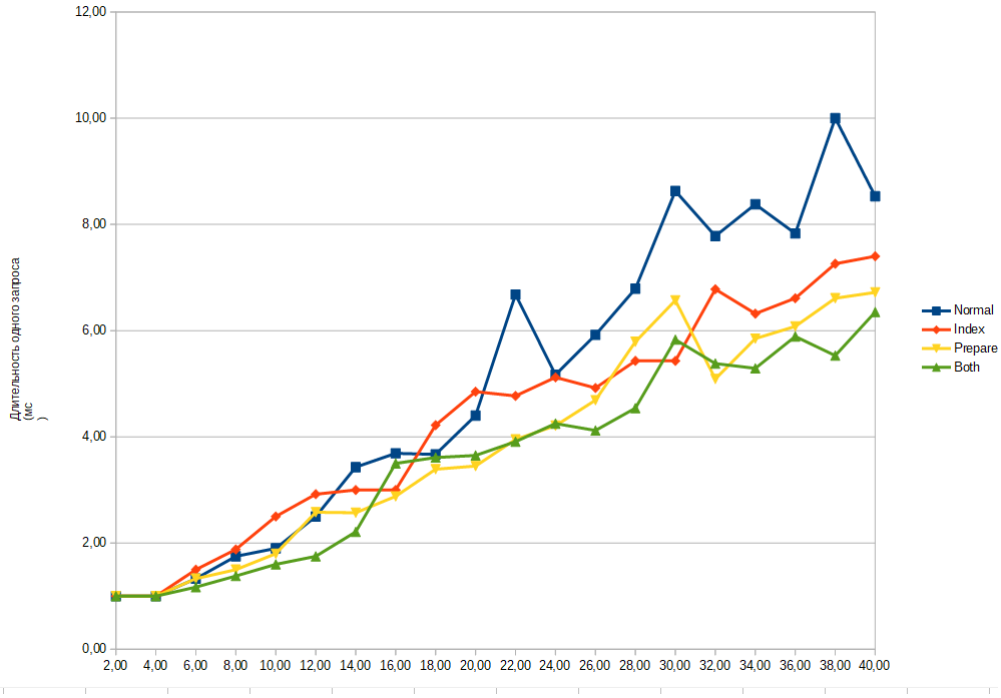

Рисунок 4.7. Все методы на одном графике

На рисунках далее представлен результат тестирования для ситуации, когда в базе данных находится большое количество данных.

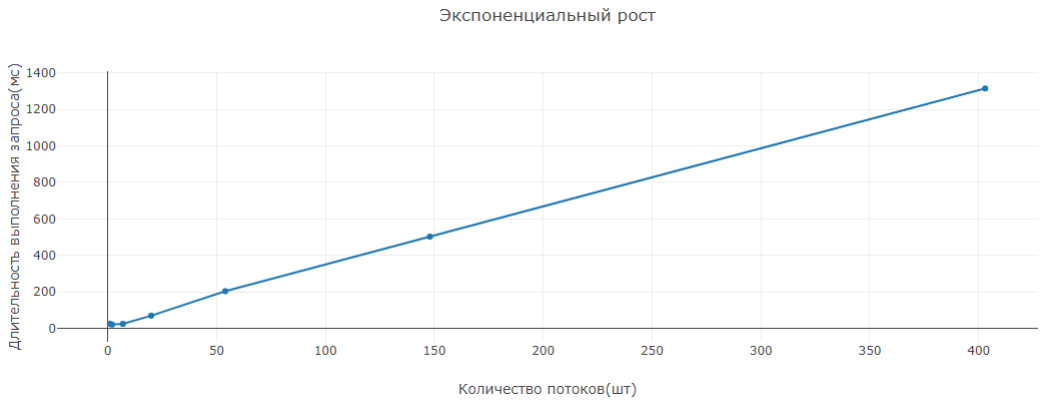

Рисунок 4.8. Возрастание нагрузки экспоненциально

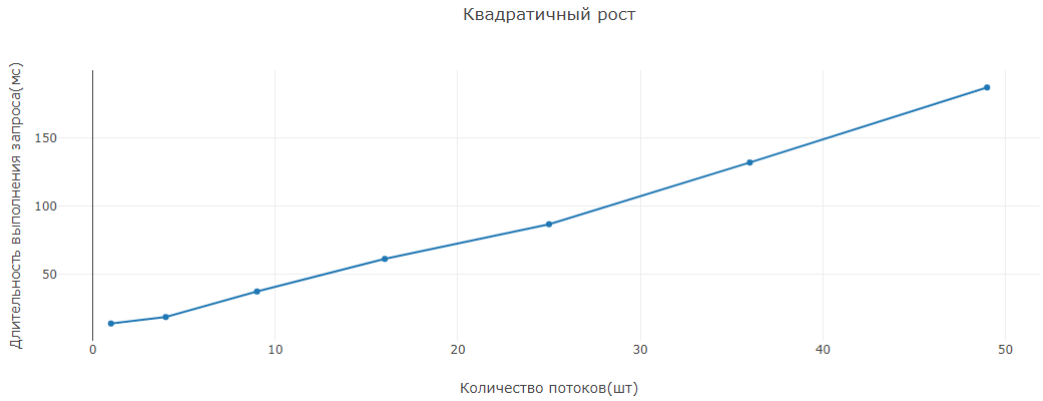

Рисунок 4.9. Возрастание нагрузки квадратично

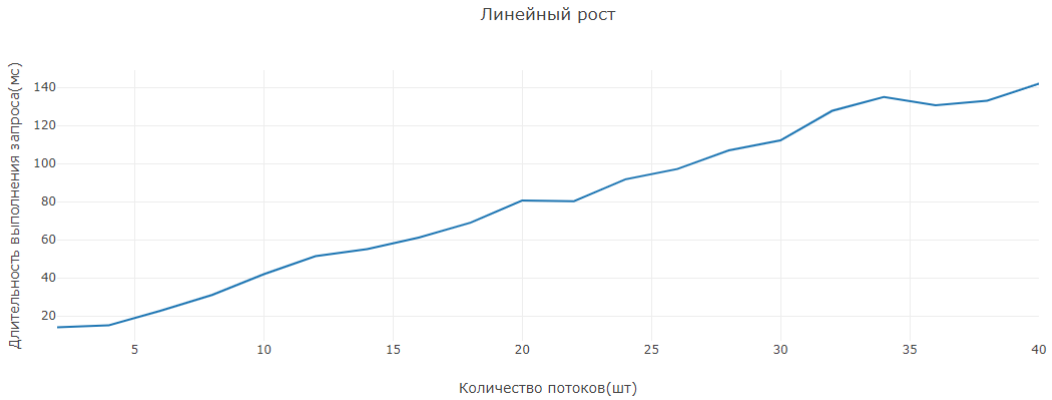

Рисунок 4.10. Возрастание нагрузки линейно

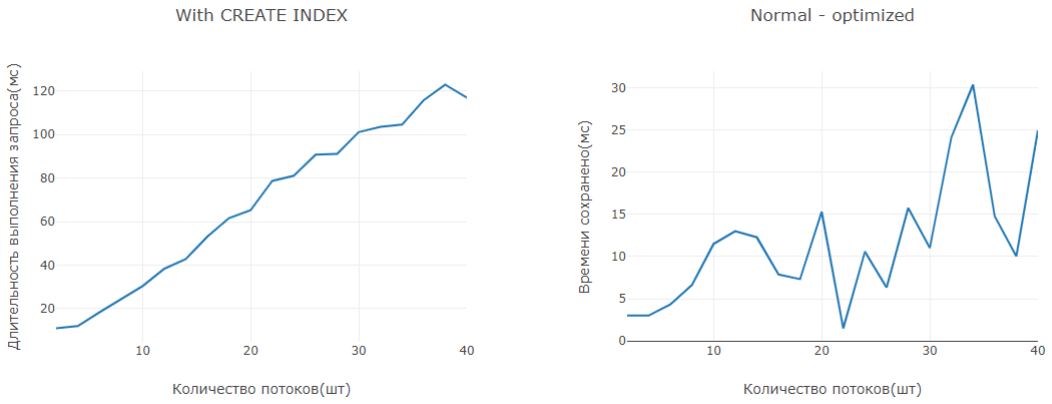

Рисунок 4.11. График после оптимизации с использованием Index

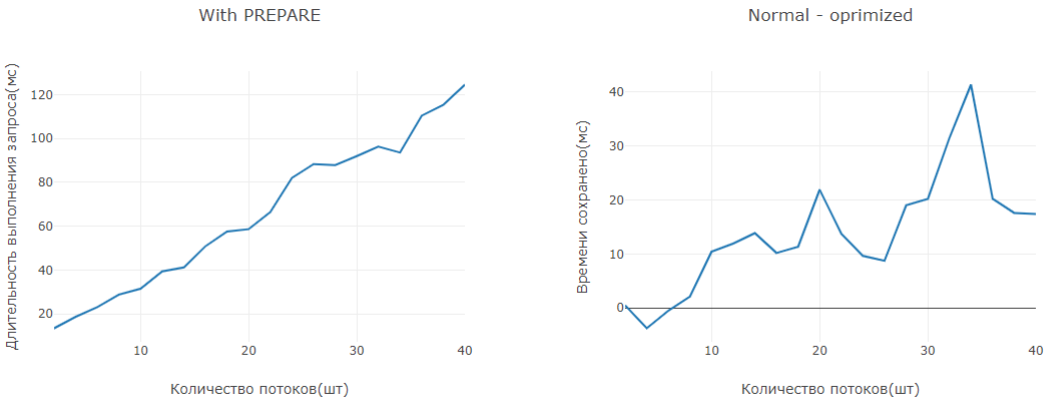

Рисунок 4.12. График после оптимизации с использованием Prepare

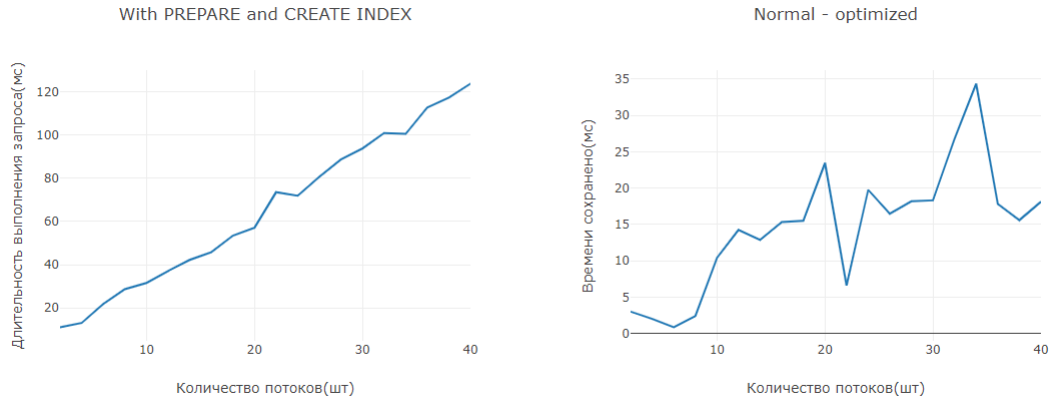

Рисунок 4.13. График после оптимизации с использованием Prepare и Index

В данной ситуации оба метода оптимизации также помогли, причем сильнее, чем до этого. В этот раз незначительная разница в длительности обработки запроса при использовании обоих методов и только Prepare не позволяет сказать, какой из этих способов является более эффективным.

На рисунке 4.14 показаны графики длительности выполнения запроса, построенные на одном графике.

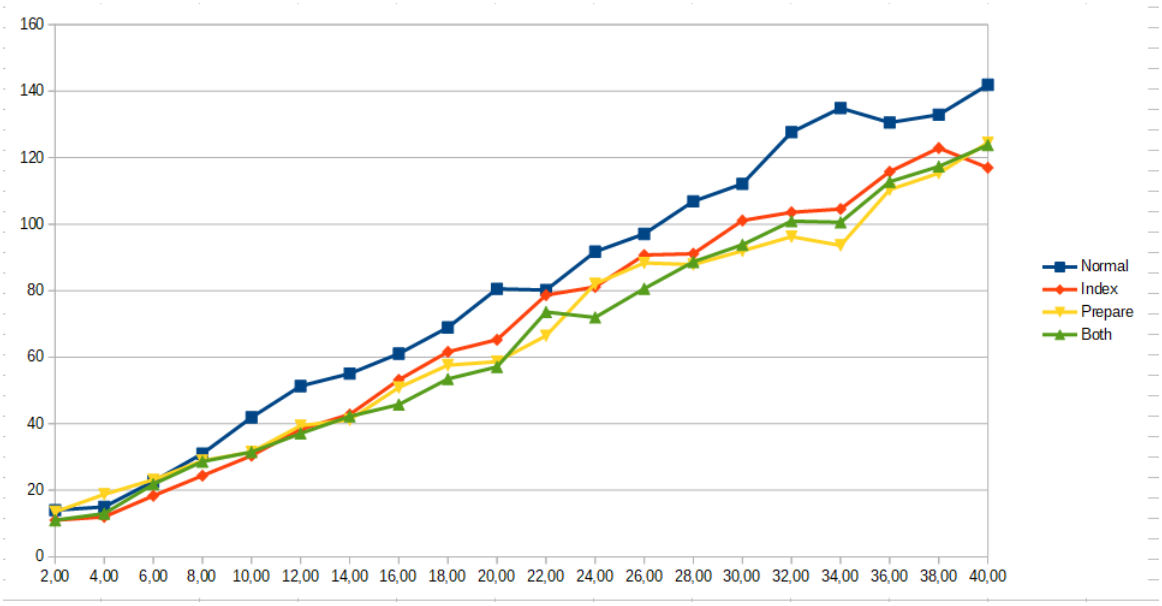

Рисунок 4.14. Все методы на одном графике

При тестировании была произведена попытка "положить"базу данных. Для этого количество клиентов, одновременно посылающих запросы повышалось вплоть до 500 в последней попытке. Также для того, чтобы упростить перегрузку базы, память, выделенная ей, например shared_buffers, work_mem и temp_buffers, была значительно уменьшена. В результате используемый компьютер был на длительное время приведен в неработоспособное состояние, однако точно сказать, что база данных "упала"нельзя.

5. Вывод

В ходе данной работы были получены навыки по проектированию базы данных, а также написанию скриптов для создания и изменения бд. Далее был разработан и оптимизирован генератор данных, подключающийся к базе данных через драйвер, после чего были написаны различные запросы к базе данных и проведена их оптимизация. При проведении оптимизации выяснилось, что для данной ситуации наиболее эффективным методом оптимизации является Prepare, хоть и разница с другим методом является небольшой.