Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатики, искусственного интеллекта и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка базы данных для обеспечения тренировок в фитнес-клубе

Студент ИУ9-62Б _____ **Терентьева А. С.**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ **Домрачева А. Б.**
(Подпись, дата) (И.О.Фамилия)

Москва, 2023 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Цель данной курсовой работы – разработать базу данных для фитнес-клуба, которая позволит вести учет тренировок пользователей и хранить статистику их показателей тела. База данных является важным инструментом для успешного функционирования фитнес-клуба по следующим причинам:

1. Отслеживание прогресса: база данных может хранить данные о весе, объеме талии, бедер и т.д., а также данные о тренировках, такие как количество повторений, нагрузка и т.д. Это позволяет посетителям отслеживать свой прогресс и улучшать свои результаты.
2. Удобство: база данных может быть доступна для посетителей через веб-интерфейс, что позволяет им удобно отслеживать свой прогресс и планировать свои тренировки.

В работе будут рассмотрены следующие задачи:

- Анализ требований к базе данных для фитнес-клуба;
- Проектирование структуры базы данных;
- Создание таблиц и связей между ними;
- Наполнение базы данных тестовыми данными;
- Создание запросов для извлечения информации из базы данных;
- Разработка интерфейса для работы с базой данных.

Для решения поставленных задач будет использоваться язык SQL и СУБД PostgreSQL. Также будет реализован сервер на языке JavaScript с помощью Node.js и фреймворка Express.js. Результатом работы станет функционирующая база данных для фитнес-клуба, которая будет обеспечивать учет тренировок пользователей и хранение статистики их показателей тела.

1. Теоретические сведения

База данных – это организованная структура данных, которая хранит информацию и предоставляет возможность эффективного доступа к ней. Базы данных используются в различных приложениях для хранения и управления большим объемом информации, такой как данные пользователей, транзакции, каталоги товаров, отчеты, статистика и т.д.

База данных обычно управляется системой управления базами данных (СУБД). Она используется для организации и обработки данных. Она также обеспечивает механизмы для контроля целостности данных, обеспечения безопасности и резервного копирования данных.

Данные в наиболее распространенных типах современных баз данных обычно хранятся в виде строк и столбцов, формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL).

По структуре и способу связей основные базы данных делятся на типы: иерархические, сетевые, реляционные, нереляционные (NoSQL), документно-ориентированные, колоночные и графовые.

1.1. Релияционные базы данных

Реляционная база данных – это тип базы данных, который использует реляционную модель данных, предложенную Эдгаром Коддом в 1970 году. Ее примерами являются Oracle, MySQL, PostgreSQL, Microsoft SQL Server и другие.

Реляционные базы данных представляют собой базы данных, которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. В реляционной модели данные представлены в виде таблиц, состоящих из строк и столбцов. Каждая таблица представляет отдельную

сущность (например, пользователи, заказы, товары и т.д.), а каждая строка таблицы представляет конкретный экземпляр этой сущности, называемый записью. Каждый столбец в таблице представляет отдельное свойство данных (такое как имя, адрес или дата рождения), которое может быть использовано для фильтрации и сортировки данных. Отношения между таблицами устанавливаются с помощью ключевых полей (первичных и внешних ключей), которые связывают записи в разных таблицах.

Реляционная модель базы данных также определяет отношения между таблицами, которые могут быть использованы для связывания данных из разных таблиц. Эти отношения могут быть один-к-одному, один-ко-многим или многие-ко-многим. Они обеспечивают возможность связывания данных из разных таблиц и извлечения информации из нескольких таблиц с помощью запросов на языке SQL.

Типы связей между таблицами:

– «один-к-одному» [1:1]. Каждая запись в одной таблице связана с одной и только одной записью в другой таблице. Для создания связи один-к-одному необходимо добавить в одну из таблиц уникальный внешний ключ, который ссылается на первичный ключ другой таблицы;

– «один-ко-многим» [1:N]. Каждая запись в одной таблице может быть связана с несколькими записями в другой таблице. В таких связях сущность на стороне 1 называют родительской, а на стороне N – дочерней. Для создания связи один-ко-многим необходимо добавить в дочернюю таблицу внешний ключ, который ссылается на первичный ключ родительской таблицы;

– «многие-ко-многим» [N:M]. Этот тип связи означает, что каждая запись в одной таблице может быть связана с несколькими записями в другой таблице, и наоборот. Для создания связи многие-ко-многим необходимо создать третью таблицу, которая будет содержать два внешних ключа, один из которых ссылается на первичный ключ одной таблицы, а другой - на первичный ключ другой таблицы;

1.2. Язык SQL и СУБД PostgreSQL

SQL (Structured Query Language) – это язык программирования, используемый для управления данными в реляционных базах данных. SQL позволяет создавать, изменять и удалять таблицы и записи, а также выполнять запросы на выборку данных из таблиц.

PostgreSQL [1] – это объектно-реляционная СУБД (система управления базами данных), которая использует язык SQL для управления данными. PostgreSQL предоставляет множество возможностей для хранения и обработки данных, включая поддержку многопоточности, транзакций, хранимых процедур и т.д. PostgreSQL имеет множество преимуществ, таких как высокая производительность, надежность и стабильность, безопасность данных и поддержка масштабирования. Он также имеет открытый исходный код, что позволяет пользователям изменять код и разрабатывать свои собственные решения на основе этой СУБД.

Основные типы данных в PostgreSQL:

– Целочисленные значения:

`integer`;

`serial` – специальный тип данных, который автоматически генерирует уникальные значения при вставке новых записей в таблицу. Основан на типе данных `integer` со встроенной функцией `default` и `sequence` и ограничением `not null`. Значение данного типа образуется путем автоинкремента значения предыдущей строки. Как правило, используется для определения идентификатора строки;

– Символьные строки:

`text`;

`varchar(n)` или `character varying(n)` – принимает параметр `n` - максимальную длину строки в символах;

– Дата и время:

`date` – дата в формате `yyyy-mm-dd`;

`timestamp` - дата и время в формате `yyyy-mm-dd hh:mi:ss`;

`timestamp with time zone` - это тип данных для хранения даты и времени в формате `yyyy-mm-dd hh:mi:ss tz`. Он содержит дату и время с точностью до микросекунд и учитывает часовой пояс;

– Булевый тип: `boolean`;

– Бинарные данные: `bytea`;

– Массивы: `array`;

– Данные в формате JSON: `json`;

Основные действия над таблицами в PostgreSQL:

1. Создание новой таблицы. Пример синтаксиса:

```
CREATE TABLE table_name (column1 datatype1, ...);
```

2. Изменение структуры таблицы, такое как добавление, изменение или удаление столбцов. Пример синтаксиса:

```
ALTER TABLE table_name
ADD COLUMN column_name datatype;
```

3. Удаление таблицы. Пример синтаксиса:

```
DROP TABLE table_name;
```

4. Вставка новых строк в таблицу. Пример синтаксиса:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, column2, ...);
```

5. Обновление существующих строк в таблице. Пример синтаксиса:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

6. Удаление строк из таблицы. Пример синтаксиса:

```
DELETE FROM table_name WHERE condition;
```

7. Извлечение данных из таблицы. Пример синтаксиса:

```
SELECT column1, column2, ... table_name WHERE condition;
SELECT * FROM table_name; – получение всех столбцов таблицы
```

Данные операции могут выполняться на уровне отдельных записей в таблице или на уровне всей таблицы.

Ограничения (constraints) в базах данных используются для определения правил, которым должны соответствовать данные в таблице. Ограничения могут использоваться для обеспечения целостности данных, защиты от ошибок ввода и других важных задач.

Основные типы ограничений в PostgreSQL:

1. `PRIMARY KEY`: определение первичного ключа таблицы – уникального идентификатора, который однозначно идентифицирует каждую запись в таблице.
2. `FOREIGN KEY`: используется для установления связи между двумя таблицами. Ограничение `FOREIGN KEY` ссылается на столбец первичного ключа в другой таблице и гарантирует, что значение в этом столбце существует в таблице-родителе.
3. `NOT NULL`: указание, что значение в столбце не может быть `NULL`.
4. `CHECK`: проверка, что значение в столбце соответствует определенному условию.
5. `UNIQUE`: указание, что значение в столбце должно быть уникальным. Это означает, что две записи в таблице не могут иметь одинаковые значения в этом столбце.

Пример синтаксиса каждого из ограничений:

```
CREATE TABLE table_name (
     column1 datatype1 PRIMARY KEY,
     column2 datatype2 NOT NULL,
     column3 datatype3 CHECK (column3 > 0),
     column 4 datatype4 UNIQUE,
     ....
     FOREIGN KEY (column3) REFERENCES parent_table (parent_column)
 );
```

Триггеры – это специальные хранимые процедуры в базах данных, которые автоматически запускаются при определенных событиях, таких как, например, вставка, обновление или удаление записей в таблицах. Триггеры могут выполнять определенные действия, например, проверять целостность данных, генерировать отчеты действий, обновлять связанные таблицы и т.д.

Существуют несколько типов триггеров:

- `BEFORE` – запускается до выполнения операции
- `AFTER` – запускается после выполнения операции
- `INSTEAD OF` – запускается вместо выполнения операции

Пример синтаксиса создания триггера:

```
CREATE TRIGGER trigger_name
AFTER INSERT OR UPDATE OR DELETE ON table_name
FOR EACH ROW
EXECUTE PROCEDURE function_name();
```

Функции – это хранимые процедуры в PostgreSQL, которые могут принимать аргументы и возвращать результаты. Создание функций в PostgreSQL происходит с помощью языка SQL или PL/pgSQL. PL/pgSQL - это язык программирования, который был разработан специально для работы с PostgreSQL.

Пример создания функции на языке PL/pgSQL:

```
CREATE FUNCTION function_name(arg1 type1, ..)
RETURNS ret_type_1 AS $$
BEGIN
RETURN (SELECT ret_column1 FROM table_name WHERE column1 = arg1);
END;
$$ LANGUAGE plpgsql;
```

В PostgreSQL также существует множество встроенных функций, которые могут быть использованы для работы с данными, такие как функции для работы с датами и временем, строками, математические функции и т.д.

Индексы в базах данных – это структуры данных, которые ускоряют поиск и сортировку записей в таблицах. Они создаются на одном или нескольких

столбцах таблицы и позволяют быстро находить записи, удовлетворяющие определенному условию. Индексы на нескольких столбцах называются составными индексами. Индексы в базах данных могут быть уникальными или неуникальными. Уникальный индекс ограничивает значения в столбце таблицы, чтобы они не повторялись, тогда как неуникальный индекс позволяет повторяющимся значениям. Пример создания уникального индекса, связывающего два:

```
CREATE UNIQUE INDEX index_name ON table_name (column_1, column_2);
```

Оператор `RAISE` используется для генерации ошибок или исключений во время выполнения SQL-запросов. Его синтаксис:

```
RAISE exception_type 'message';
```

где `exception_type` - тип исключения (например, `EXCEPTION`, `WARNING`, `NOTICE`), а `message` - текст сообщения об ошибке.

2. Проектирование базы данных

2.1. Анализ предметной области

В качестве предметной области было выбрано приложения для поддержания тренировок и ведения статистики их результатов. Необходимо разработать базу данных, которая будет соответствовать следующим требованиям:

1. Обеспечение хранения данных о пользователях. База данных должна предоставлять следующие возможности пользователю для управления своим аккаунтом:
   - Регистрация (создание аккаунта)
   - Редактирование персональной информации
   - Удаление аккаунта
2. Обеспечение хранения статистики. База данных должна предоставлять следующие возможности пользователям для ведения статистики показателей тела и результатов тренировок:
   - Создание нового параметра, ведения статистики показателей которого хочет начать пользователь
   - Получение параметров, созданных пользователем
   - Получение, добавление, редактирование и удаление установленных показателей. Каждому показателю соответствует параметр, а параметр, в свою очередь, может содержать несколько показателей
   - Учет даты и времени установления показателей
3. Обеспечение хранения комплексов упражнений. База данных должна предоставлять следующие возможности пользователям для просмотра комплекса упражнений:
   - Создание новой тренировки
   - Получения всех тренировок, хранящихся в базе данных
   - Получение создателя тренировки

– Получение и изменение списка добавленных тренировок в профиль пользователя

2.2. Модель «сущность-связь»

Модель "сущность-связь" (Entity-Relationship Model, ER-модель) – это методология для описания структуры данных в базе данных на основе сущностей и их взаимосвязей. ER-модель представляет собой графическую диаграмму, которая позволяет визуализировать и описать сущности (entities), атрибуты (attributes) и связи (relationships) между сущностями. В ER-модели сущность представляет реальный объект или понятие, о котором хранится информация. Связи в ER-модели определяют отношения между сущностями. Они показывают, как одна сущность связана с другой.

На основе описанной предметной области была создана модель «сущность-связь», включающая четыре сущности:

1. USER – сущность, являющаяся абстракцией пользователя.

Идентификатор:

– user_id;

Атрибуты:

– name – отображаемое имя пользователя;
– user_login – уникальное имя пользователя;
– password – хэш пароля пользователя;
– email – электронная почта пользователя;
– gender – пол пользователя;
– birthday – день рождения пользователя;

2. PARAMETER – сущность, являющаяся абстракцией параметра.

Идентификаторы:

– user_id – идентификатор пользователя, создавшего параметр;
– parameter_id;

Атрибуты:

– name – название параметра;

– updated_at – время последнего добавления показателей тела;

3. BODYDATA – сущность, являющаяся абстракцией показателя теля.

Идентификаторы:

– parameter_id – идентификатор соответствующего параметра;

– user_id – идентификатор пользователя, создавшего параметр;

– id;

Атрибуты:

– value – значение показателя тела;

– date – время установления значения показателя;

4. WORKOUT – сущность, являющаяся абстракцией тренировки.

Идентификатор:

– id;

Атрибуты:

– name – название тренировки;

– info – описание тренировки;

Диаграмма данной модели приведена на рис. 1.

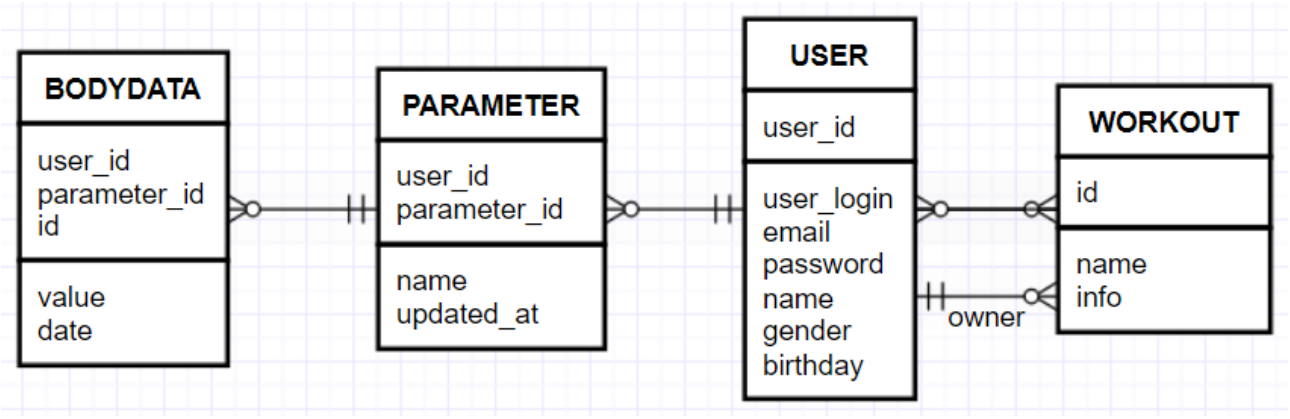

рис. 1 Модель «сущность-связь»

У каждого параметра обязательно должен быть автор, который его создал, а сам пользователь может создавать несколько параметров (или ни одного),

между сущностями USER и PARAMETER установлена связь «один-ко-многим» (с минимальными кардинальными числами 1 и 0 соответственно).

Показателей тела одного параметра, может быть неограниченное количество (в том числе и ни одного). Каждый показатель соответствует одному определенному параметру. Между сущностями PARAMETER и BODYDATA установлена связь «один-ко-многим» (с минимальными кардинальными числами 1 и 0 соответственно).

Так как у комплекса упражнений должен быть единственный создатель и при этом пользователь может быть создателем одновременно нескольких комплексов, связь USER – WORKOUT (owner) имеет тип «один-ко-многим» (с минимальными кардинальными числами 1 и 0 соответственно).

Рассмотрим связь USER – WORKOUT. Пользователь может добавить несколько тренировок в профиль, с другой стороны, несколько пользователей могут добавить одну тренировку в свои профили. Поэтому связь USER – WORKOUT имеет тип «многие-ко-многим» (с минимальными кардинальными числами 0 и 0 соответственно).

2.3. Преобразование модели «сущность-связь» в реляционную модель

Из модели «сущность-связь» согласно процедуре преобразования была получена реляционная модель, включающая 5 таблиц (атрибуты двух моделей совпадают):

1. USER – сущность, являющаяся абстракцией пользователя.

Идентификатор:

– user_id – идентификатор пользователя;

Атрибуты:

– user_login – альтернативный идентификатор пользователя;

– email – уникальный адрес электронный почты;

2. PARAMETER – сущность, являющаяся абстракцией параметра.

Идентификаторы:

– parameter_id;

Атрибуты:

– user_id – идентификатор пользователя, создавшего параметр;

3. BODYDATA – сущность, являющаяся абстракцией показателя теля.

Идентификаторы:

– id;

Атрибуты:

– user_id – идентификатор пользователя, создавшего параметр;

– parameter_id – идентификатор соответствующего параметра;

4. WORKOUT – сущность, являющаяся абстракцией тренировки.

Идентификатор:

– id;

Атрибуты:

– owner_id – идентификатор пользователя, создавшего тренировку;

Диаграмма данной модели приведена на рис. 2.

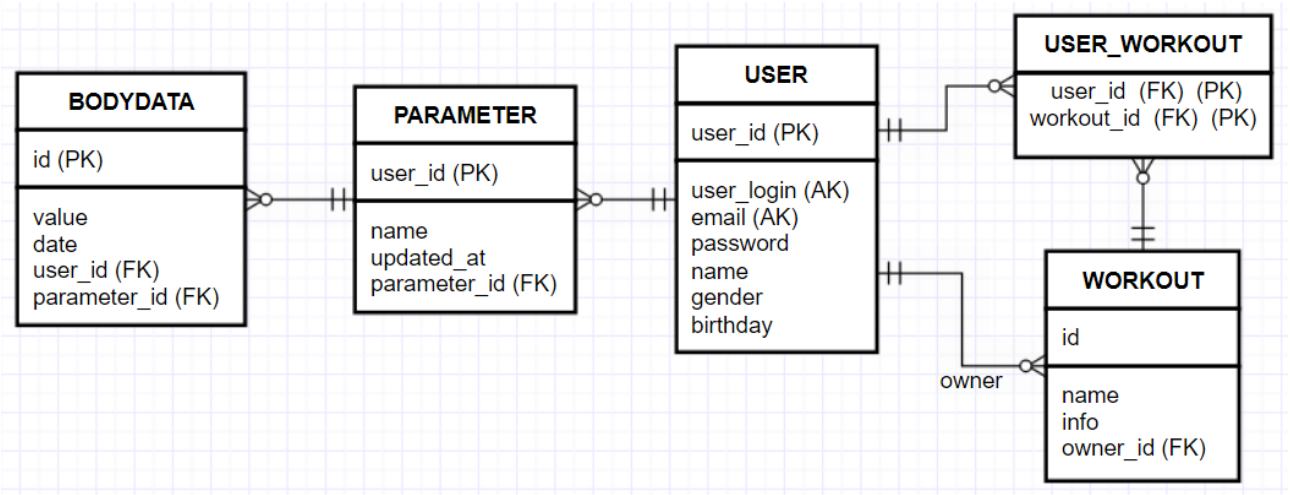

рис. 2 Реляционная модель

3. Реализация

В данной части содержится описание реализации приложения.

Клиент и веб-интерфейс проекта разработан с использованием языка разметки HTML, языка каскадных таблиц стилей CSS и языка программирования JavaScript. Сервер проекта разработан на языке JavaScript с помощью Node.js и фреймворка Express.js. При разработке базы данных использовался язык SQL и СУБД PostgreSQL. Сервер базы данных размещается в качестве контейнера в Docker. Репозиторий проекта находится на странице [2].

Для запуска сервера необходимо выполнить следующие команды:

1. `npm install` – установка зависимостей
2. `node server.js`

Для того, чтобы обратиться к приложению необходимо открыть `http://localhost:3000/`.

3.1. Инструменты веб-разработки

HTML [3] – Hyper Text Markup Language – Язык разметки гипертекста. это код, который используется для структурирования и отображения веб-страницы и её контента. технологии разработки сайтов страница, которую понимает веб-браузер.

Кроме HTML-кода в браузер загружаются и обрабатываются CSS - каскадные таблицы стилей, и программы (обычно называемые скриптами) на таких языках, как Java, JavaScript, TypeScript.

CSS – это инструмент, позволяющий придать единый вид различным типовым элементам сайта, в частности - легко настраивать и изменять облик всех элементов страницы.

Node.js [4] – это среда выполнения JavaScript, построенная на движке V8. Она позволяет запускать JavaScript на стороне сервера, что позволяет

разрабатывать серверные приложения и API. Node.js основан на событийно-ориентированной и неблокирующей модели ввода-вывода, что делает его эффективным и масштабируемым для обработки большого количества запросов.

Express.js [5] – это гибкий фреймворк для разработки веб-приложений на Node.js. Он предоставляет интерфейс для создания маршрутов и обработки HTTP-запросов и отправки ответов на них клиенту.

Создание сервера с помощью Express.js:

```
const express = require('express');
const app = express();
const port = 3000;

app.listen(port, () => {
    console.log("listen server " + port);
});
```

Маршрутизация запросов:

```
app.METHOD('/path, function);
```

где `METHOD` – метод HTTP запроса (например, get, put или post),

`/path` – путь запроса,

`function` – функция-обработчик запроса.

Промежуточное программное обеспечение (middleware) в Express.js представляет собой функции, которые выполняются последовательно при обработке запросов. Они выполняются перед тем, как запрос достигнет конечного обработчика (route handler).

Метод `use()` позволяет указать промежуточное программное обеспечение, которое будет применяться ко всем запросам, независимо от метода и пути запроса. Он может быть использован для выполнения различных задач, таких как обработка ошибок, авторизация пользователя, обработка сессий и многое другое.

Пример:

```
app.use((req, res, next) => {
    ...
```

```
    next(); // Передача управления следующему промежуточному программному
обеспечению
 });
```

EJS (Embedded JavaScript) [6] – это шаблонизатор для JavaScript и Node.js. Он позволяет создавать динамические HTML-страницы, вставлять переменные, выполнять условные операторы и циклы, а также подключать другие шаблоны. EJS позволяет разделить представление и логику приложения, что делает код более читаемым и поддерживаемым. В Express.js сгенерированные HTML-страниц отправляются клиенту с помощью функции `response.render`.

Подключение ejs-шаблонов и папки, со статическими файлами:

```
app.use(express.static('static'));
app.set('view engine', 'ejs');
app.set('views', './templates');
```

Node.js, Express.js и EJS предоставляют разработчикам инструменты для создания серверных приложений и веб-сайтов с динамическим содержимым. Они позволяют эффективно работать с запросами, маршрутизацией и шаблонизацией.

Для получения доступа к базе данных на сервере была использована библиотека – pg (node-postgres) [7] – клиент PostgreSQL для Node.js.

Пример ее использования:

```
const { Pool } = require('pg');
const pool = new Pool({
    user: 'postgres',
    host: 'localhost',
    database: 'postgres',
    password: 'postgrespw',
    port: 49154,
});
pool.query("select * from users");
```

Для получения и изменения информации текущего сеанса пользователя была подключена библиотека express-session [8].

Пример ее использования:

```
const session = require('express-session');
```

```
app.use(
    session({
        secret: 'you secret key',
        resave: false,
        saveUninitialized: false,
    })
);
app.get('/increment', (req, res) => {
    req.session.counter = req.session.counter || 0;
    req.session.counter += 1;
});
```

Для генерации диаграмм была использована библиотека chart.js [9].

Docker — это инструмент для создания, развертывания и запуска приложений в контейнерах. Для данной работы было принято решение использовать его для развертывания сервера базы данных PostgreSQL.

3.2. Создание таблиц в базе данных

Запросы создания таблиц в базе данных представлены в приложении А.

Выделим несколько особенностей, возникших при создании таблиц и связей между ними:

1) При добавлении в таблицу `users` записей с пустым полем user_login, по умолчанию поле должно становить равным `'user'+user_id`.

Это осуществляется путем создания триггерной функции на событие `insert` с типом `before`:

```
create or replace function users_insert_function()
  returns trigger as
$$
begin
      if new.user_login = '' then
            new.user_login = 'user' || cast(new.user_id as varchar(250));
      end if;
      return new;
end;
$$
language 'plpgsql'

create or replace trigger users_insert_trigger
before insert on users
for each row
      execute function users_insert_function()
```

Доступ к вставляемым записям происходит при помощи ключевого слова new. Конкатенация строк – при помощи оператора || или функции concat(str1, str2). Преобразование числа в строку – с использованием функции cast с указанием результирующего типа.

2) Необходимо, чтобы названия параметров были уникальными в рамках одного пользователя. Один из способов обеспечения этого свойства – создание уникального индекса:

```
create unique index user_parameter_index on body_data (user_id_ref, parameter_name);
```

3) Поле updated_at таблицы body_data должно обновляться при добавлении новых дочерних записей в таблицу parameter_data. Создадим триггерную функцию, которая будет срабатывать при добавлении и изменении записей в дочернюю таблицу:

```
create or replace function update_statistic()
returns trigger as $$
begin
  update body_data
  set updated_at = now()
  where body_data.id = new.body_data_ref;
  return new;
end;
$$ language plpgsql;

create or replace trigger update_statistic
after insert or update on parameter_data
for each row
execute function update_statistic();
```

4) В процессе работы над проектом, могла возникнуть потребность внесения изменений в структуру таблиц. Например, в данном примере устанавливаются действия при удалении и изменении родительской таблицы:

```
alter table body_data drop constraint body_data_user_id_ref_fkey;
alter table body_data add constraint body_data_user_id_ref_fkey
foreign key (user_id_ref) references users(user_login) on delete cascade on update cas-
cade;
```

Чтобы внести изменения над атрибутом с ограничением foreign key, необходимо сначала удалить старое ограничение, а затем создать заново с указанными действиями на события родительской таблицы – users.

Другой пример:

```
alter table body_data
alter column updated_at type timestamp with time zone,
alter column updated_at set default now();
```

до применения изменений:

```
updated_at timestamp
```

Здесь не было установлено значение по умолчанию и было необходимо изменить тип данных на тип, учитывающий часовой пояс.

Полученная схема базы данных показана на рис. 3.

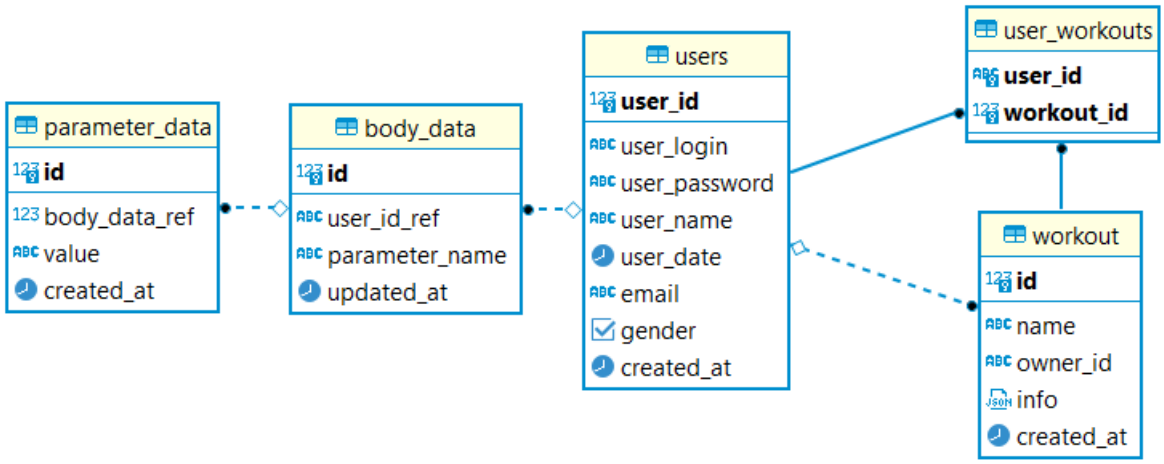

рис. 3 Схема базы данных

3.3. Файловая структура приложения

Проект содержит следующую файловую структуру:

/db – содержит код подключения к PostgreSQL и предоставляет интерфейс для доступа к БД, а также содержит функции-обработчики запросов на сервер

/static – содержит статические файлы, такие как скрипты, css – стили и изображения, передаются на браузер вместе с html страницей

/templates – содержит ejs-шаблоны страниц сайта

package.json – содержит информацию о зависимостях

server.js – содержит создание и настройку сервера, а также маршрутизацию запросов

3.4. Функциональные возможности приложения

Интерфейс приложения можно разделить на 4 части:

1) Авторизация

Данный раздел содержит главную страницу, страницу регистрации и страницу аутентификации (рис. 4 a-c).

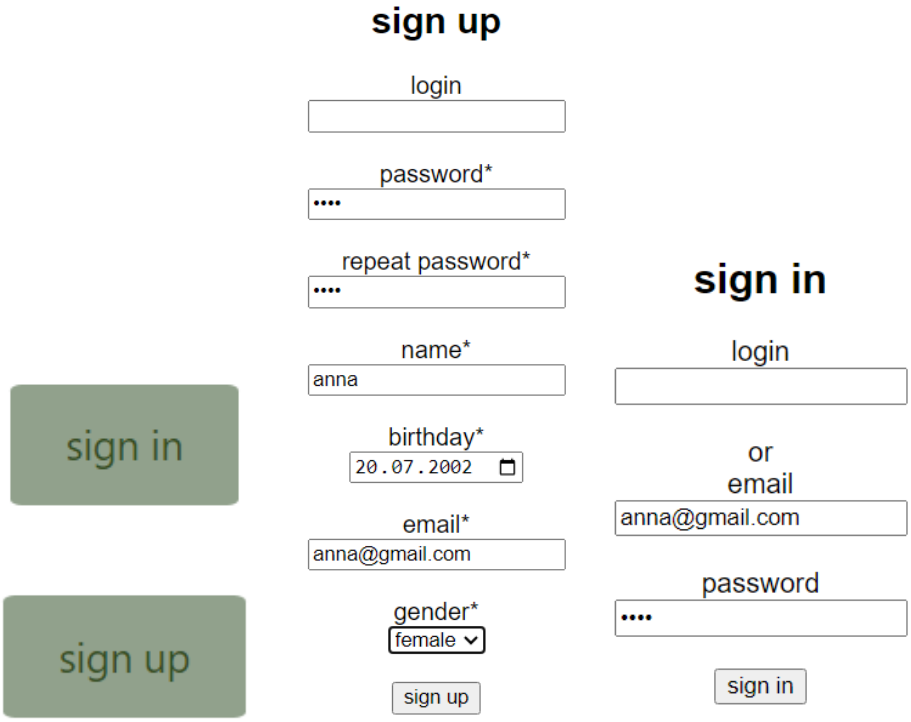

рис. 4 a – главная страница, b – форма регистрации, c – форма авторизации

Пользователь может добавить аккаунт, если значения полей `login` и `email` еще не существуют в таблице `users`. В противном случае – сервер выдаст ошибку. Пользователь может войти в аккаунт, если введенный login или email существует в базе данных и при этом введенный пароль совпадает с сохраненным.

2) Профиль

Данный раздел содержит страницу профиля и страницу редактирования данных пользователя (рис. 5 a-c).

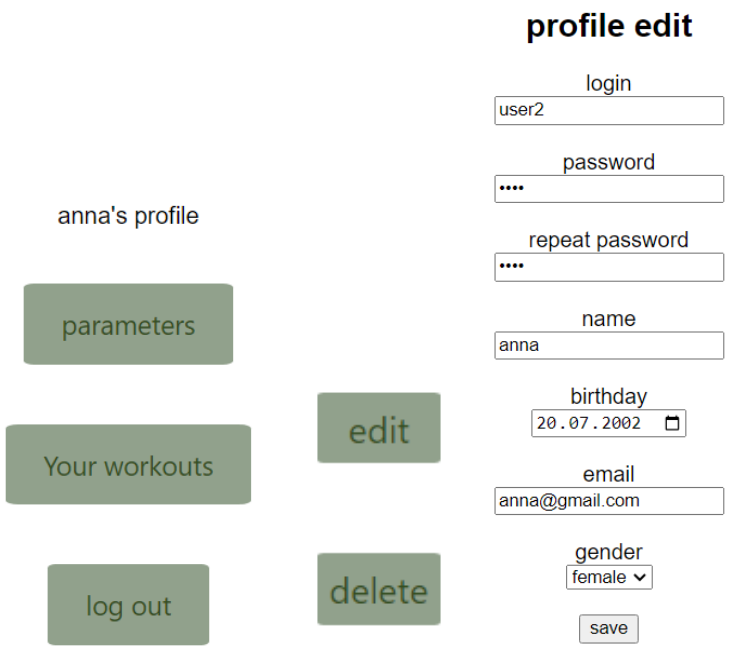

рис. 5 a – страница профиля, b – кнопки удаления и перехода на страницу редактирования профиля, c – страница редактирования профиля

На главной странице профиля пользователь может перейти в другие разделы, покинуть аккаунт, а также перейти на страницу с формой редактирования данных профиля и удалить свой профиль из базы данных. При попытке изменить `login` или `email` на существующий сервер выдаст ошибку.

3) Раздел тренировок

Данная часть сайта содержит список добавленных в профиль пользователя тренировок и страницу всех тренировок, содержащихся в базе данных, но не добавленных пользователем в свой профиль (рис. 6 a-c).

рис. 6 a – страница с добавленными тренировками, b – окно с описанием тренировки, c – страница всех тренировок, не добавленных пользователем

На странице с добавленными тренировками пользователь может открыть окно с подробным описанием комплекса упражнений, также он может исключить из своего списка любую из тренировок. На странице общих тренировок пользователь может выбрать из имеющихся и добавить в свой профиль. Если пользователь является создателем тренировки, то он может ее удалить из базы данных. Дополнительно список всех тренировок можно сортировать по: названию, владельцу или дате создания, по убыванию (desc) или

по возрастанию (asc). Также здесь присутствует форма добавления нового комплекса упражнений.

4) Раздел показателей тела

Данный раздел содержит список всех созданных пользователем параметров и список значений каждого из параметра (рис. 7 a-c).

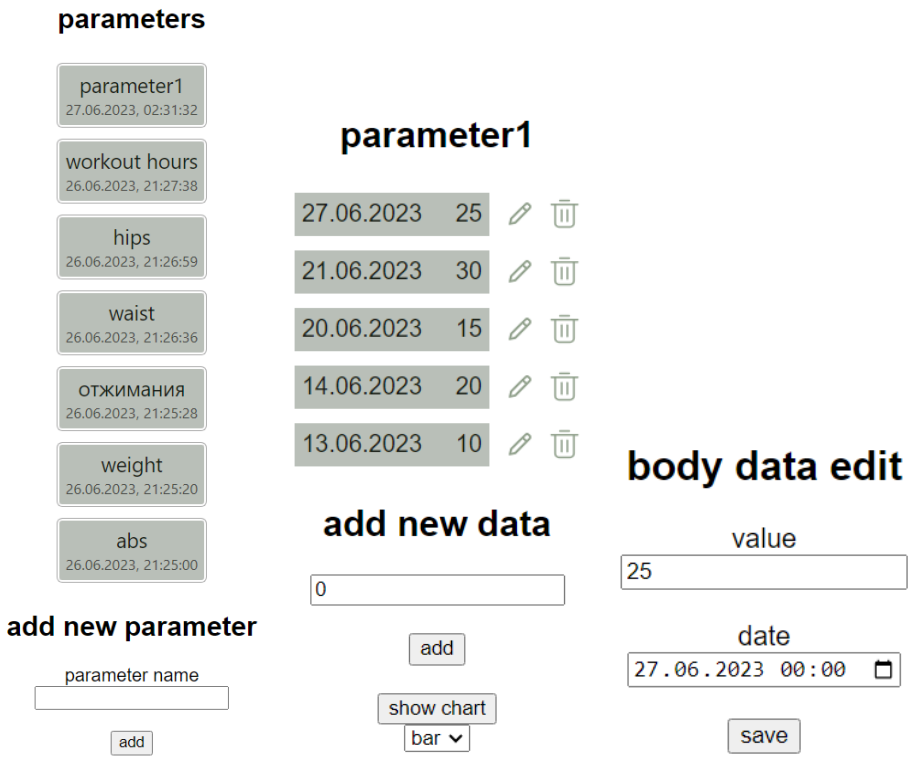

рис. 7 a – страница с параметрами, b – страница со значениями параметра, c – страница редактирования значения и даты показателя тела

На странице со списком параметров пользователь может добавить новый параметр, если его название не совпадает ни с одним из созданных (в рамках одного пользователя). При нажатии на параметр откроется страница со значениями этого параметра в разные промежутки времени. Здесь можно добавить новый показатель, редактировать и удалять записи.

Также можно получить диаграмму – статистику значений параметра тела на временной школе:

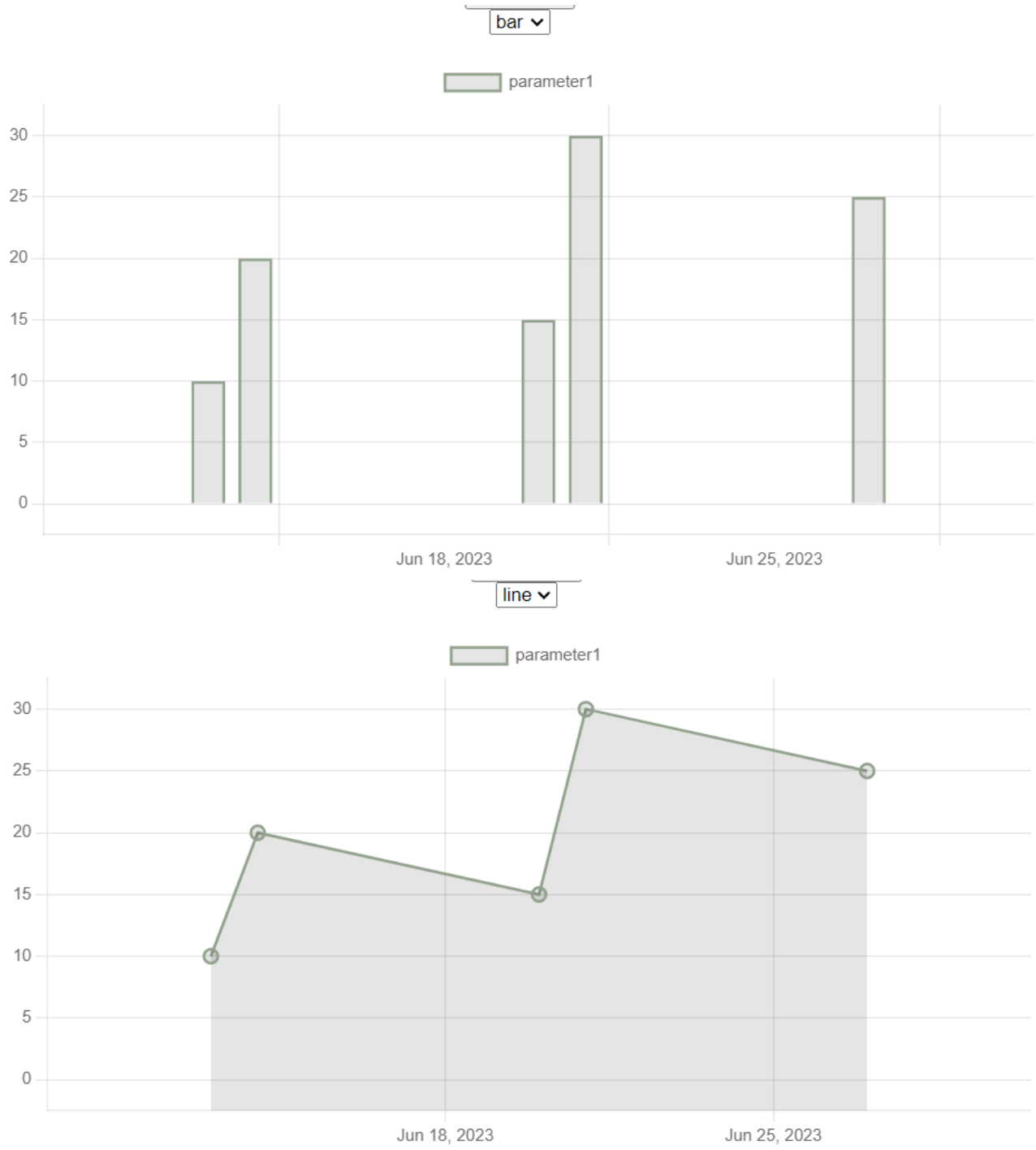

рис. 8 Примеры разных типов диаграмм статистики параметра

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были приобретены практические навыки разработки реляционных баз данных в СУБД PostgreSQL. В рамках выполнения задания была спроектирована реляционная база данных, удовлетворяющая требованиям, которые были сформулированы в процессе исследования предметной области приложения. Были сформированы запросы для создания, изменения и удаления таблиц и запросы для управления данными.

На основе изученной информации было реализовано веб-приложение, которое предоставляет удобный интерфейс для просмотра и изменения данных на сервере. С помощью него пользователи также могут получить наглядную диаграмму статистики по каждому из параметров показателей тела, что дополнительно расширяет функциональные возможности приложения.

Все поставленные задачи курсовой работы были выполнены. Помимо работы с базами данных, были приобретены навыки отправки и обработки HTTP-запросов и создания клиент-серверных приложений на языке программирования JavaScript.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация PostgreSQL – https://www.postgresql.org/docs/
2. Репозиторий проекта – https://github.com/mathhyyn/bd_coursework
3. Документация веб-технологий для разработчиков – https://developer.mozilla.ORG/ru/docs/Web
4. Документация Node.js – https://nodejs.org/ru/docs
5. Документация Express.js – https://expressjs.com/
6. Документация шаблонизатора ejs – https://ejs.co/#docs
7. Документация библиотеки для работы с PostgreSQL в Node.js – https://node-postgres.com/
8. Документация библиотеки express-session – https://www.npmjs.com/package/express-session
9. Документация генератора графиков и диаграмм – https://www.chartjs.org/docs/latest/

ПРИЛОЖЕНИЕ

```
create table users (
  user_id serial primary key,
  user_login varchar(255) unique,
  user_password varchar(255) not null,
  user_name varchar(255) not null,
  user_date date not null,
  email varchar(255) unique not null,
  gender bool default false,
  created_at timestamp with time zone default now()
)

create table parameter_data (
  id serial primary key,
  body_data_ref integer references body_data(id),
  value  varchar(255),
  created_at timestamp with time zone default now()
);

create table body_data (
      id serial primary key,
      user_id_ref varchar(255) references users(user_login),
      parameter_name varchar(255),
      updated_at timestamp
);

alter table body_data
alter column updated_at type timestamp with time zone,
alter column updated_at set default now();

alter table body_data drop constraint body_data_user_id_ref_fkey;
alter table body_data add constraint body_data_user_id_ref_fkey
foreign key (user_id_ref) references users(user_login)
                                on delete cascade on update cascade;

alter table parameter_data drop constraint parameter_data_body_data_ref_fkey;
alter table parameter_data add constraint parameter_data_body_data_ref_fkey
foreign key (body_data_ref) references body_data(id)
                                on delete cascade on update cascade;


create table workout (
  id serial primary key,
  name varchar(255),
  owner_id  varchar(255) references users(user_login)
                                                on update cascade on delete set null,
  info json default '{"description":""}',
  created_at timestamp with time zone default current_timestamp
)

create table user_workouts (
  user_id varchar(255) references users(user_login)
                                        on update cascade on delete cascade,
  workout_id integer references workout(id) on update cascade on delete cascade,
  primary key (user_id, workout_id)
);
```

Листинг А – создание таблиц в базе данных