# Лабораторная работа №1

## Разработка серверной части прикладной программы

### 1. План выполнения работы

Данная лабораторная работа выполняется на основе лабораторной работы №6 первого семестра по дисциплине «Базы данных», то есть весь реализованный ранее функционал необходимо сохранить. Данная работа включает:

− разработку спецификаций (требований к ПО) серверной части;

− программирование серверной части.

Разработка спецификаций (технических требований) серверной части (backend) программы. Включает минимально:

− составление уточненной реляционной схемы, определение имен и типов данных для всех полей БД;

− приведение имен в БД и имен в реляционной схеме в полное соответствие и создание подробного описания для каждого имени (имена таблиц и полей в реляционной схеме должны быть идентичны именам в SQL операторах);

− выделение двух ролей — пользователь и суперпользователь и определение их прав;

− выделение основной таблицы, с отображения которой по умолчанию стартует приложение;

− выделение справочников (LookUp Table) в отдельную категорию таблиц, право на запись в которые имеет только суперпользователь;

− оформление спецификации в соответствии с ГОСТ.

Программирование серверной части программы. Включает:

− создание скриптов для создания пустой базы данных;

− первичное наполнение таблиц-справочников;

− создание генераторов и скриптов для наполнения основной и прочих больших таблиц (объем в несколько сот записей);

− создание и отладка необходимых триггеров, функций и операторов;

− создание скриптов для сохранения/восстановления БД;

− тестирование сервера с помощью psql;

− оформление отчета о лабораторной работе;

− использование библиотеки libpq.

### 2. Ход работы

Лабораторные работы выполняются на основе модели данных и реляционной схемы, которые были представлены в первом семестре

дисциплины «Базы данных». При необходимости реляционная схема может быть изменена, чтобы более соответствовать разрабатываемому приложению. При внесении изменений в реляционную схему необходимо аргументировать те или иные решения. В данном случае внесение изменений не требуется и реляционная схема остается такого же вида, как и в предыдущих работах. Детализация требований к ПО должна быть такова, чтобы их можно было передать программисту для реализации без комментариев. Далее приведен пример основных моментов выполнения работы.

### 2.1. Технические требования

Уточненная реляционная схема изображается в соответствии с UML и должна иметь вид, изображенный на рисунке 2.1:

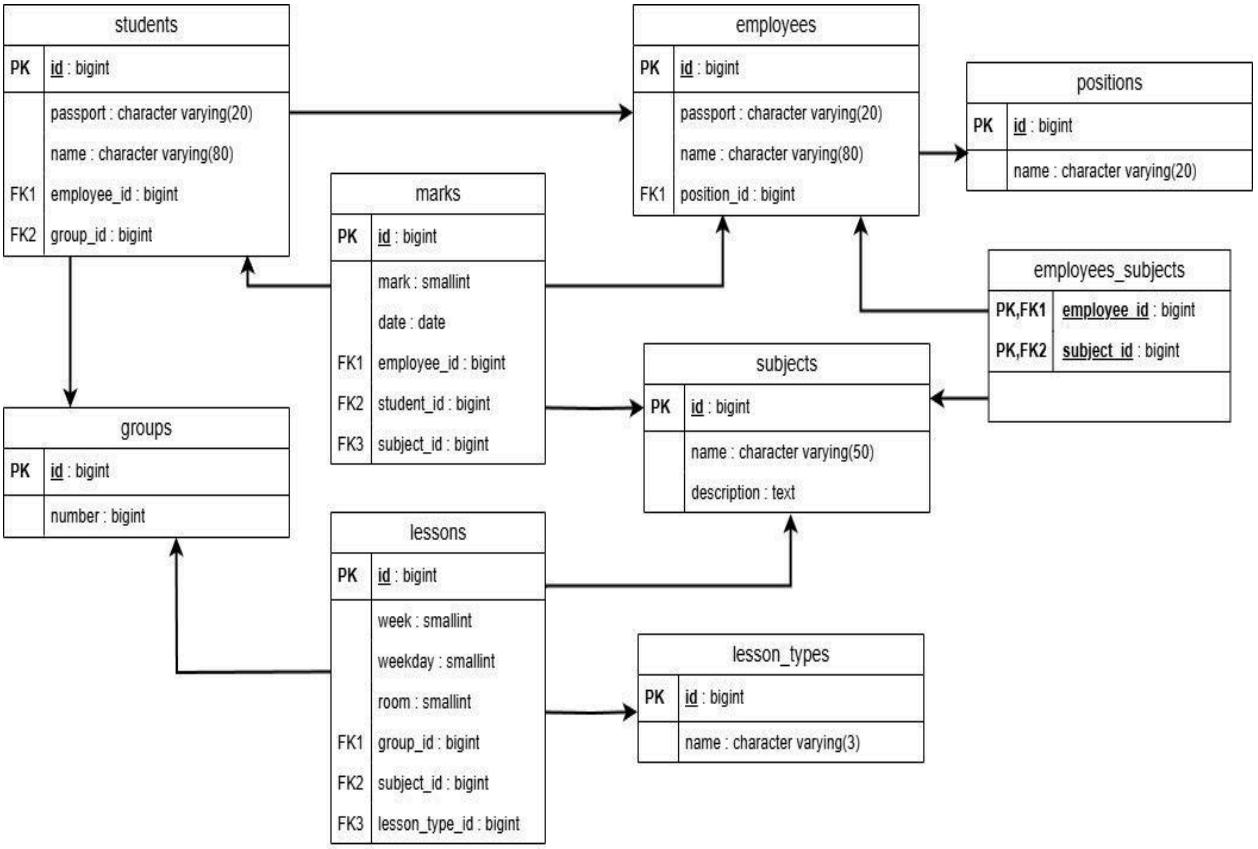


Рисунок 2.1 – Уточненная реляционная схема

Таблица `students` представляет студентов университета. Описание имен таблицы `students`:

- `id`: идентификатор студента. Первичный ключ;
- `passport`: номер паспорта студента;
- `name`: ФИО студента;
- `employee_id`: идентификатор сотрудника, у которого в подчинении

студент. Внешний ключ;

– group_id: идентификатор группы студента. Внешний ключ.

Таблица groups представляет учебные группы. Описание имен таблицы groups:

– id: идентификатор группы. Первичный ключ;

– number: номер группы.

Таблица marks представляет журнал оценок. Описание имен таблицы marks:

– id: идентификатор оценки. Первичный ключ;

– mark: оценка;

– date: дата выставления оценки;

– employee_id: идентификатор преподавателя, выставившего оценку. Внешний ключ;

– student_id: идентификатор студента, получившего оценку. Внешний ключ;

– subject_id: идентификатор предмета, по которому выставлена оценка. Внешний ключ.

Таблица lessons представляет расписание занятий. Описание имен таблицы lessons:

– id: идентификатор занятия. Первичный ключ;

– week: номер недели;

– weekday: день недели;

– room: номер аудитории;

– group_id: идентификатор группы, у которой назначено занятие. Внешний ключ;

– subject_id: идентификатор предмета, по которому будет занятие. Внешний ключ;

– lesson_type_id: идентификатор типа занятия. Внешний ключ.

Таблица employees представляет сотрудников университета. Описание имен таблицы employees:

– id: идентификатор сотрудника. Первичный ключ;

– passport: номер паспорта сотрудника;

– name: ФИО сотрудника;

– position_id: идентификатор должности сотрудника. Внешний ключ.

Таблица subjects представляет предметы, изучаемые в университете. Описание имен таблицы subjects:

– id: идентификатор предмета. Первичный ключ;

– name: название предмета;

– description: описание предмета.

Таблица lesson_types представляет типы занятий. Описание имен

таблицы lesson_types:

– id: идентификатор типа занятия. Первичный ключ;

– name: название типа занятия.

Таблица positions представляет должности в университете. Описание имен таблицы positions:

– id: идентификатор должности. Первичный ключ;

– name: название должности.

Таблица employees_subjects представляет предметы, которые может вести преподаватель. Описание имен таблицы employees_subjects:

– employee_id: идентификатор преподавателя. Первичный ключ;

– subject_id: идентификатор предмета. Первичный ключ.

В данной схеме в категорию справочных таблиц должны быть выделены: positions, groups, subjects и lesson_types, так как они содержат предварительно определенные неизменяемые данные, которые используются для поиска и сопоставления значений в других таблицах.

В качестве основной таблицы должна быть выделена таблица lessons, так как она содержит основные бизнес-события (занятия), а также данные в ней динамичны и постоянно изменяются.

Пользователь должен обладать правами просмотра, сохранения результатов запросов и редактирования всех таблиц, кроме справочных, а суперпользователь обладать теми же правами что и обычный пользователь, но с возможностью редактирования справочных таблиц и создания бэкапа базы данных. Для выполнения действий от имени суперпользователя приложение должно запрашивать пароль суперпользователя.

Серверная часть прикладной программы должна быть реализована в виде HTTP-сервера. Тела ответов сервера, так же, как и тела запросов должны быть представлены в формате JSON.

Для взаимодействия с ресурсами (таблицами) должны использоваться стандартные HTTP-методы:

1) GET – получение данных о ресурсе;

2) POST – создание нового ресурса;

3) PUT – обновление существующего ресурса;

4) DELETE – удаление ресурса.

Каждый ресурс должен быть доступен по уникальному URL:

– /api/employees: таблица employees;

– /api/students: таблица students;

– /api/groups: таблица groups;

– /api/marks: таблица marks;

– /api/lessons: таблица lessons;

– /api/subjects: таблица subjects;

– /api/lesson-types: таблица lesson_types;

– /api/positions: таблица `positions`;

– /api/employees-subjects: таблица `employees_subjects`.

Серверная часть прикладной программы должна предоставлять следующие операции для работы с базой данных:

− просмотр таблиц;

− фильтрация содержимого таблиц;

− добавление записей в таблицы;

− обновление записей в таблицах;

− удаление записей из таблиц;

− выполнение специальных запросов;

− создание бэкапов базы данных;

− сохранение результатов запросов в файл.

### 2.2. Программирование серверной части

Работа с базой данных в серверной части должна осуществляться с использованием библиотеки libpq. Как было сказано ранее, данная лабораторная работа выполняется на основе уже имеющегося приложения для работы с базой данных. В данном случае, приложение было написано на языке программирования, который предоставляет мощные инструменты для встраивания и вызова кода на C. Таким образом, уже существующее приложение можно адаптировать под отдельную серверную часть, в которой будет осуществляться взаимодействие с базой данных с помощью библиотеки libpq через механизм другого языка. Если язык программирования, на котором ранее была выполнена работа №6 не поддерживает механизмы работы с C, то необходимо предпринять соответствующие меры, например реализовать серверную часть с помощью инструментов, поддерживающих работу с libpq.

Реализация серверной части в виде HTTP-сервера обеспечит возможность взаимодействия с ней, например из браузера, расширяя возможности реализации пользовательского интерфейса прикладной программы.

Так как функционал серверной части сохраняет функционал приложения из лабораторной работы №6, за исключением того, что для получения данных теперь используются HTTP-запросы, в качестве примера будет показан запрос на получение всех данных из таблицы `lessons`. Для этого пользователю необходимо выполнить GET-запрос следующего формата: `http://{адрес хоста}:{порт}/api/lessons`. Результат запроса в браузере изображен на рисунке 2.2:

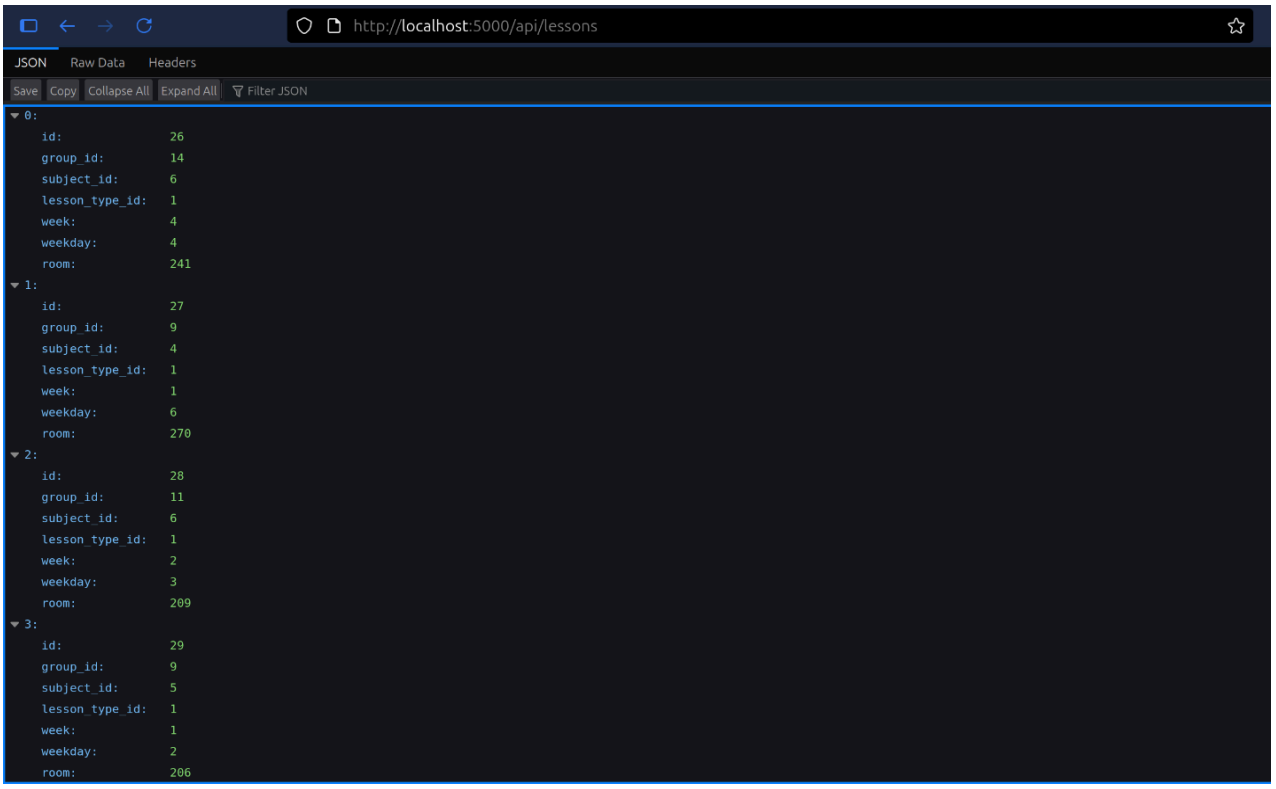


Рисунок 2.2 – Результат запроса в браузере

Далее, в качестве примера одного из скриптов, для создания таблиц в базе данных используется следующий скрипт:

```
CREATE TABLE IF NOT EXISTS public.students
(
    id BIGSERIAL NOT NULL,
    name character varying(80),
    passport character varying(20),
    employee_id bigint,
    group_id bigint,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.groups
(
    id BIGSERIAL NOT NULL,
    number bigint,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.employees
(
    id BIGSERIAL NOT NULL,
    name character varying(80),
    passport character varying(20),
    position_id bigint,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.subjects
```

```
(
    id BIGSERIAL NOT NULL,
    name character varying(50),
    description text,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.marks
(
    id BIGSERIAL NOT NULL,
    employee_id bigint,
    student_id bigint,
    subject_id bigint,
    mark smallint,
    date date,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.lesson_types
(
    id BIGSERIAL NOT NULL,
    name character varying(3),
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.lessons
(
    id BIGSERIAL NOT NULL,
    group_id bigint,
    subject_id bigint,
    lesson_type_id bigint,
    week smallint,
    weekday smallint,
    room smallint,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS public.employees_subjects
(
    employee_id bigint,
    subject_id bigint,
    PRIMARY KEY (employee_id, subject_id)
);

CREATE TABLE IF NOT EXISTS public.positions
(
    id BIGSERIAL NOT NULL,
    name character varying(20),
    PRIMARY KEY (id)
);

ALTER TABLE IF EXISTS public.students
    ADD CONSTRAINT fk_students_employees FOREIGN KEY (employee_id)
    REFERENCES public.employees (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.students
    ADD CONSTRAINT fk_students_groups FOREIGN KEY (group_id)
    REFERENCES public.groups (id) MATCH SIMPLE
```

```
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.employees
    ADD CONSTRAINT position_id_fk FOREIGN KEY (position_id)
    REFERENCES public.positions (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.marks
    ADD CONSTRAINT employee_id_fk FOREIGN KEY (employee_id)
    REFERENCES public.employees (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.marks
    ADD CONSTRAINT student_id_fk FOREIGN KEY (student_id)
    REFERENCES public.students (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.marks
    ADD CONSTRAINT subject_id_fk FOREIGN KEY (subject_id)
    REFERENCES public.subjects (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.lessons
    ADD CONSTRAINT group_id_fk FOREIGN KEY (group_id)
    REFERENCES public.groups (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.lessons
    ADD CONSTRAINT subject_id_fk FOREIGN KEY (subject_id)
    REFERENCES public.subjects (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.lessons
    ADD CONSTRAINT lesson_type_id_fk FOREIGN KEY (lesson_type_id)
    REFERENCES public.lesson_types (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.employees_subjects
```

```
    ADD CONSTRAINT employee_id_fk FOREIGN KEY (employee_id)
    REFERENCES public.employees (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;


ALTER TABLE IF EXISTS public.employees_subjects
    ADD CONSTRAINT subject_id_fk FOREIGN KEY (subject_id)
    REFERENCES public.subjects (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
```