МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информатика»

Лабораторная работа №7

**«Взаимодействие клиентского приложения с СУБД»**

по дисциплине

**Информационные технологии и программирование**

|  |  |
| --- | --- |
|  | Выполнил: студент гр. БЭИ2202  Бояркин Е. В.  Вариант №3  Проверил: зав. кафедры «Информатика» Халабия Р. Ф. |

Москва, 2024 г.

Содержание

[1 Постановка задачи 4](#_Toc184184472)

[2 Алгоритмы 5](#_Toc184184473)

[2.1 join\_strings 5](#_Toc184184474)

[2.2 get\_valid\_int 5](#_Toc184184475)

[2.3 get\_valid\_string 5](#_Toc184184476)

[2.4 get\_valid\_date 6](#_Toc184184477)

[2.5 Group::to\_json, Dormitory::to\_json, Room::to\_json, Student::to\_json, AccommodationInfo::to\_json, Field::to\_json 6](#_Toc184184478)

[2.6 Group::from\_json, Dormitory::from\_json, Room::from\_json, Student::from\_json, AccommodationInfo::from\_json, Field::from\_json 6](#_Toc184184479)

[2.7 DatabaseHandler::DatabaseHandler 6](#_Toc184184480)

[2.8 DatabaseHandler::~DatabaseHandler 7](#_Toc184184481)

[2.9 DatabaseHandler::save 7](#_Toc184184482)

[2.10 DatabaseHandler::create\_default\_tables 7](#_Toc184184483)

[2.11 DatabaseHandler::is\_group\_code\_unique, is\_dormitory\_code\_unique, is\_room\_code\_unique, is\_student\_code\_unique, is\_accommodation\_unique 7](#_Toc184184484)

[2.12 DatabaseHandler::group\_exists, dormitory\_exists, room\_exists, student\_exists 7](#_Toc184184485)

[2.13 DatabaseHandler::add\_group, add\_dormitory, add\_room, add\_student, add\_accommodation 8](#_Toc184184486)

[2.14 DatabaseHandler::delete\_group, delete\_dormitory, delete\_room, delete\_student, delete\_accommodation 8](#_Toc184184487)

[2.15 DatabaseHandler::edit\_group, edit\_dormitory, edit\_room, edit\_student, edit\_accommodation 8](#_Toc184184488)

[2.16 DatabaseHandler::display\_groups, display\_dormitories, display\_rooms, display\_students, display\_accommodations 8](#_Toc184184489)

[2.17 DatabaseHandler::export\_to\_postgresql 9](#_Toc184184490)

[2.18 DatabaseHandler::import\_from\_postgresql 9](#_Toc184184491)

[2.19 DatabaseHandler::menu\_groups, menu\_dormitories, menu\_rooms, menu\_students, menu\_accommodations 9](#_Toc184184492)

[2.20 DatabaseHandler::menu\_edit\_additional\_fields 9](#_Toc184184493)

[2.21 DatabaseHandler::main\_menu 10](#_Toc184184494)

[2.22 main 10](#_Toc184184495)

[3 Листинг программы 11](#_Toc184184496)

[4 Результаты работы программы 50](#_Toc184184497)

[5 Вывод 57](#_Toc184184498)

# 1 Постановка задачи

Изучить основные принципы взаимодействия приложений, разработанных в архитектуре «клиент-сервер», осуществляющее доступ к базе данных по технологии ODBC (или другой технологии взаимодействия с базами данных), изучить основные принципы работы клиентского приложения с API ODBC и с другими технологиями доступа к СУБД.

Задание:

Добавить в Ваше клиентское приложение, выполненное в прошлом задании, следующие пункты меню:

* специальное меню: с добавлением поля с указанием формата и типа поля, с удалением поля и редактирование поля с указанием формата, и типа поля. Необходимо выбирать все таблицы вашего варианта. Главное, чтобы таблица формируется как новая с помощью вашего меню, или если файл существует, структура считывается из файла.
* экспорт таблиц данных в СУБД MySQL, с сопоставлением всех возможных типов данных и форматов представления и их длины.
* импорт таблиц данных из СУБД MySQL с учетом всех изменений, которые могли произойти в СУБД MySQL.

# 2 Алгоритмы

## 2.1 join\_strings

Шаг 1. Инициализировать пустую строку для результата.

Шаг 2. Для каждого элемента в векторе строк:

Шаг 2.1. Добавить текущую строку к результату.

Шаг 2.2. Если текущий элемент не последний:

Шаг 2.2.1. Добавить разделитель к результату.

Шаг 3. Вернуть полученную строку.

## 2.2 get\_valid\_int

Шаг 1. Начать бесконечный цикл:

Шаг 1.1. Вывести подсказку для ввода целого числа.

Шаг 1.2. Прочитать ввод пользователя.

Шаг 1.3. Если ввод некорректен (не является числом):

Шаг 1.3.1. Вывести сообщение об ошибке.

Шаг 1.3.2. Очистить поток ввода и пропустить неверные данные.

Шаг 1.4. Иначе:

Шаг 1.4.1. Очистить оставшиеся данные в потоке ввода.

Шаг 1.4.2. Вернуть введенное целое число.

## 2.3 get\_valid\_string

Шаг 1. Начать бесконечный цикл:

Шаг 1.1. Вывести подсказку для ввода строки.

Шаг 1.2. Прочитать строку от пользователя.

Шаг 1.3. Если строка пуста:

Шаг 1.3.1. Вывести сообщение об ошибке.

Шаг 1.4. Иначе:

Шаг 1.4.1. Вернуть введенную строку.

## 2.4 get\_valid\_date

Шаг 1. Определить шаблон регулярного выражения для даты в формате "ДД.ММ.ГГГГ".

Шаг 2. Начать бесконечный цикл:

Шаг 2.1. Вывести подсказку для ввода даты.

Шаг 2.2. Прочитать дату от пользователя.

Шаг 2.3. Если дата не соответствует шаблону:

Шаг 2.3.1. Вывести сообщение об ошибке формата даты.

Шаг 2.4. Иначе:

Шаг 2.4.1. Извлечь день и месяц из даты.

Шаг 2.4.2. Если день или месяц вне допустимого диапазона:

Шаг 2.4.2.1. Вывести сообщение об ошибке даты.

Шаг 2.4.3. Иначе:

Шаг 2.4.3.1. Вернуть введенную дату.

## 2.5 Group::to\_json, Dormitory::to\_json, Room::to\_json, Student::to\_json, AccommodationInfo::to\_json, Field::to\_json

Шаг 1. Создать JSON-объект с полями структуры.

Шаг 2. Вернуть созданный JSON-объект.

## 2.6 Group::from\_json, Dormitory::from\_json, Room::from\_json, Student::from\_json, AccommodationInfo::from\_json, Field::from\_json

Шаг 1. Создать пустую структуру.

Шаг 2. Извлечь данные из JSON-объекта и заполнить соответствующие поля структуры.

Шаг 3. Вернуть заполненную структуру.

## 2.7 DatabaseHandler::DatabaseHandler

Шаг 1. Инициализировать имя файла и схему базы данных.

Шаг 2. Попытаться подключиться к PostgreSQL:

Шаг 2.1. Если подключение успешно:

Шаг 2.1.1. Вывести сообщение об успешном подключении.

Шаг 2.2. Если подключение не удалось:

Шаг 2.2.1. Вывести сообщение об ошибке подключения.

Шаг 3. Загрузить данные из файла или создать пустые таблицы по умолчанию, если файл отсутствует.

Шаг 4. Завершить инициализацию.

## 2.8 DatabaseHandler::~DatabaseHandler

Шаг 1. Если подключение установлено:

Шаг 1.1. Закрыть подключение и освободить ресурсы.

## 2.9 DatabaseHandler::save

Шаг 1. Создать JSON-объект из текущих данных.

Шаг 2. Сохранить JSON-объект в файл.

## 2.10 DatabaseHandler::create\_default\_tables

Шаг 1. Определить таблицы и поля по умолчанию.

Шаг 2. Создать таблицы в локальной базе и PostgreSQL, если они отсутствуют.

Шаг 3. Сохранить обновленную схему.

## 2.11 DatabaseHandler::is\_group\_code\_unique, is\_dormitory\_code\_unique, is\_room\_code\_unique, is\_student\_code\_unique, is\_accommodation\_unique

Шаг 1. Проверить, что код отсутствует среди существующих записей в соответствующей таблице.

Шаг 2. Вернуть результат проверки.

## 2.12 DatabaseHandler::group\_exists, dormitory\_exists, room\_exists, student\_exists

Шаг 1. Проверить, существует ли запись с заданным кодом в соответствующей таблице.

Шаг 2. Вернуть результат проверки.

## 2.13 DatabaseHandler::add\_group, add\_dormitory, add\_room, add\_student, add\_accommodation

Шаг 1. Проверить уникальность кода.

Шаг 2. Если код не уникален или данные некорректны:

Шаг 2.1. Вывести сообщение об ошибке.

Шаг 2.2. Отменить добавление.

Шаг 3. Добавить запись в локальные данные.

Шаг 4. Сохранить изменения.

## 2.14 DatabaseHandler::delete\_group, delete\_dormitory, delete\_room, delete\_student, delete\_accommodation

Шаг 1. Проверить, существует ли запись с заданным кодом.

Шаг 2. Проверить наличие связанных данных, которые могут блокировать удаление.

Шаг 3. Удалить запись, если она существует и нет блокирующих данных.

Шаг 4. Сохранить изменения.

## 2.15 DatabaseHandler::edit\_group, edit\_dormitory, edit\_room, edit\_student, edit\_accommodation

Шаг 1. Найти запись с заданным кодом.

Шаг 2. Проверить уникальность нового кода и корректность данных.

Шаг 3. Обновить запись новыми значениями.

Шаг 4. Сохранить изменения.

## 2.16 DatabaseHandler::display\_groups, display\_dormitories, display\_rooms, display\_students, display\_accommodations

Шаг 1. Вывести заголовок таблицы.

Шаг 2. Для каждой записи в таблице:

Шаг 2.1. Сформировать строку с данными записи.

Шаг 2.2. Вывести строку.

## 2.17 DatabaseHandler::export\_to\_postgresql

Шаг 1. Подключиться к PostgreSQL.

Шаг 2. Очистить таблицы PostgreSQL.

Шаг 3. Синхронизировать схемы таблиц.

Шаг 4. Экспортировать данные.

Шаг 5. Закоммитить транзакцию.

## 2.18 DatabaseHandler::import\_from\_postgresql

Шаг 1. Подключиться к PostgreSQL.

Шаг 2. Импортировать схемы таблиц.

Шаг 3. Импортировать данные.

Шаг 4. Сохранить изменения в локальной базе.

## 2.19 DatabaseHandler::menu\_groups, menu\_dormitories, menu\_rooms, menu\_students, menu\_accommodations

Шаг 1. Отобразить меню управления.

Шаг 2. Выполнить действия в зависимости от выбора пользователя:

Добавить новую запись.

Удалить существующую запись.

Изменить запись.

Отобразить записи таблицы.

Шаг 3. Повторять цикл, пока не выбран выход.

## 2.20 DatabaseHandler::menu\_edit\_additional\_fields

Шаг 1. Получить название таблицы и ID записи.

Шаг 2. Отобразить текущие дополнительные поля.

Шаг 3. В зависимости от выбора пользователя:

Добавить новое дополнительное поле.

Изменить существующее поле.

Удалить дополнительное поле.

Шаг 4. Сохранить изменения.

## 2.21 DatabaseHandler::main\_menu

Шаг 1. Отобразить главное меню.

Шаг 2. В зависимости от выбора пользователя:

Выполнить действия для управления таблицами.

Выполнить экспорт или импорт данных.

Отобразить таблицу.

Изменить дополнительные поля.

Шаг 3. Повторять цикл, пока не выбран выход.

## 2.22 main

Шаг 1. Создать объект `DatabaseHandler`.

Шаг 2. Запустить главное меню приложения.

Шаг 3. Обработать возможные ошибки.

# 3 Листинг программы

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <optional>  #include <format>  #include <regex>  #include <limits>  #include <vector>  #include <map>  #include "json.hpp"  #include <pqxx/pqxx>  using json = nlohmann::json;  using namespace std;  string join\_strings(const vector<string>& vec, const string& delimiter)  {  string result;  for (size\_t i = 0; i < vec.size(); ++i)  {  result += vec[i];  if (i != vec.size() - 1)  result += delimiter;  }  return result;  }  namespace  {  int get\_valid\_int(const string& prompt)  {  int value;  while (true)  {  cout << prompt;  cin >> value;  if (cin.fail())  {  cout << "Неверный ввод. Пожалуйста, введите целое число.\n";  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  }  else  {  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  return value;  }  }  }  string get\_valid\_string(const string& prompt)  {  string value;  while (true)  {  cout << prompt;  getline(cin, value);  if (value.empty())  {  cout << "Ввод не может быть пустым. Пожалуйста, попробуйте снова.\n";  }  else  {  return value;  }  }  }  string get\_valid\_date(const string& prompt)  {  string date;  const regex date\_pattern(R"(^\d{2}\.\d{2}\.\d{4}$)");  while (true)  {  cout << prompt;  getline(cin, date);  if (!regex\_match(date, date\_pattern))  {  cout << "Неверный формат даты. Пожалуйста, используйте формат ДД.ММ.ГГГГ.\n";  }  else  {  const int day = stoi(date.substr(0, 2));  const int month = stoi(date.substr(3, 2));  if (day < 1 || day > 31 || month < 1 || month > 12)  {  cout << "Неверная дата. Пожалуйста, введите корректную дату.\n";  }  else  {  return date;  }  }  }  }  }  struct Group  {  int code;  string name;  string faculty;  [[nodiscard]] json to\_json() const  {  return json{{"code", code}, {"name", name}, {"faculty", faculty}};  }  static Group from\_json(const json& j)  {  Group g;  g.code = j.at("code").get<int>();  g.name = j.at("name").get<string>();  g.faculty = j.at("faculty").get<string>();  return g;  }  };  struct Dormitory  {  int code;  string name;  string address;  [[nodiscard]] json to\_json() const  {  return json{{"code", code}, {"name", name}, {"address", address}};  }  static Dormitory from\_json(const json& j)  {  Dormitory d;  d.code = j.at("code").get<int>();  d.name = j.at("name").get<string>();  d.address = j.at("address").get<string>();  return d;  }  };  struct Room  {  int code;  string number;  int dormitory\_code;  [[nodiscard]] json to\_json() const  {  return json{{"code", code}, {"number", number}, {"dormitory\_code", dormitory\_code}};  }  static Room from\_json(const json& j)  {  Room r;  r.code = j.at("code").get<int>();  r.number = j.at("number").get<string>();  r.dormitory\_code = j.at("dormitory\_code").get<int>();  return r;  }  };  struct Student  {  int code;  string full\_name;  string birth\_date;  string gender;  int group\_code;  string passport\_series;  string passport\_number;  string issued\_by;  string issued\_date;  [[nodiscard]] json to\_json() const  {  return json{  {"code", code},  {"full\_name", full\_name},  {"birth\_date", birth\_date},  {"gender", gender},  {"group\_code", group\_code},  {"passport\_series", passport\_series},  {"passport\_number", passport\_number},  {"issued\_by", issued\_by},  {"issued\_date", issued\_date}  };  }  static Student from\_json(const json& j)  {  Student s;  s.code = j.at("code").get<int>();  s.full\_name = j.at("full\_name").get<string>();  s.birth\_date = j.at("birth\_date").get<string>();  s.gender = j.at("gender").get<string>();  s.group\_code = j.at("group\_code").get<int>();  s.passport\_series = j.at("passport\_series").get<string>();  s.passport\_number = j.at("passport\_number").get<string>();  s.issued\_by = j.at("issued\_by").get<string>();  s.issued\_date = j.at("issued\_date").get<string>();  return s;  }  };  struct AccommodationInfo  {  int room\_code;  int student\_code;  string move\_in\_date;  string move\_out\_date;  [[nodiscard]] json to\_json() const  {  return json{  {"room\_code", room\_code},  {"student\_code", student\_code},  {"move\_in\_date", move\_in\_date},  {"move\_out\_date", move\_out\_date}  };  }  static AccommodationInfo from\_json(const json& j)  {  AccommodationInfo a;  a.room\_code = j.at("room\_code").get<int>();  a.student\_code = j.at("student\_code").get<int>();  a.move\_in\_date = j.at("move\_in\_date").get<string>();  a.move\_out\_date = j.at("move\_out\_date").get<string>();  return a;  }  };  struct Field  {  string name;  string type;  string format;  [[nodiscard]] json to\_json() const  {  return json{{"name", name}, {"type", type}, {"format", format}};  }  static Field from\_json(const json& j)  {  Field f;  f.name = j.at("name").get<string>();  f.type = j.at("type").get<string>();  f.format = j.at("format").get<string>();  return f;  }  };  class DatabaseHandler  {  string filename;  json data;  map<string, vector<Field>> schema;  pqxx::connection\* conn;  map<string, vector<string>> default\_fields = {  {"groups", {"code", "name", "faculty"}},  {"dormitories", {"code", "name", "address"}},  {"rooms", {"code", "number", "dormitory\_code"}},  {  "students", {  "code", "full\_name", "birth\_date", "gender", "group\_code",  "passport\_series", "passport\_number", "issued\_by", "issued\_date"  }  },  {"accommodations", {"room\_code", "student\_code", "move\_in\_date", "move\_out\_date"}}  };  map<string, string> primary\_keys = {  {"groups", "code"},  {"dormitories", "code"},  {"rooms", "code"},  {"students", "code"},  {"accommodations", "student\_code"}  };  public:  DatabaseHandler(string fname) : filename(std::move(fname)), conn(nullptr)  {  const string conn\_str = "dbname=itap7 user=itap7 password=12345 hostaddr=127.0.0.1 port=5432";  try  {  conn = new pqxx::connection(conn\_str);  if (conn->is\_open())  {  cout << "Подключение к PostgreSQL успешно.\n";  }  else  {  cout << "Не удалось открыть подключение к PostgreSQL.\n";  delete conn;  conn = nullptr;  }  }  catch (const exception& e)  {  cout << "Ошибка подключения к PostgreSQL: " << e.what() << '\n';  conn = nullptr;  }  ifstream file(filename);  if (file.good())  {  file >> data;  if (data.contains("schema"))  {  for (auto& [table, fields] : data["schema"].items())  {  for (auto& field : fields)  {  schema[table].emplace\_back(Field::from\_json(field));  }  }  }  else  {  data["schema"] = json::object();  }  // Initialize additional\_data if not present  if (!data.contains("additional\_data"))  {  data["additional\_data"] = json::object();  }  }  else  {  data = json{  {"groups", json::array()},  {"dormitories", json::array()},  {"rooms", json::array()},  {"students", json::array()},  {"accommodations", json::array()},  {"schema", json::object()},  {"additional\_data", json::object()}  };  create\_default\_tables();  save();  }  }  ~DatabaseHandler()  {  if (conn && conn->is\_open())  {  conn->close();  delete conn;  }  }  void save() const  {  json j;  j["groups"] = data["groups"];  j["dormitories"] = data["dormitories"];  j["rooms"] = data["rooms"];  j["students"] = data["students"];  j["accommodations"] = data["accommodations"];  j["schema"] = data["schema"];  j["additional\_data"] = data["additional\_data"];  ofstream file(filename);  file << std::format("{0:4}\n", j.dump(4));  }  void create\_default\_tables()  {  // Define default tables and their fields  map<string, vector<Field>> default\_schema = {  {  "groups", {  {"code", "INT PRIMARY KEY", ""},  {"name", "VARCHAR(100)", ""},  {"faculty", "VARCHAR(100)", ""}  }  },  {  "dormitories", {  {"code", "INT PRIMARY KEY", ""},  {"name", "VARCHAR(100)", ""},  {"address", "VARCHAR(200)", ""}  }  },  {  "rooms", {  {"code", "INT PRIMARY KEY", ""},  {"number", "VARCHAR(10)", ""},  {"dormitory\_code", "INT", ""}  }  },  {  "students", {  {"code", "INT PRIMARY KEY", ""},  {"full\_name", "VARCHAR(150)", ""},  {"birth\_date", "DATE", ""},  {"gender", "VARCHAR(10)", ""},  {"group\_code", "INT", ""},  {"passport\_series", "VARCHAR(10)", ""},  {"passport\_number", "VARCHAR(10)", ""},  {"issued\_by", "VARCHAR(100)", ""},  {"issued\_date", "DATE", ""}  }  },  {  "accommodations", {  {"room\_code", "INT", ""},  {"student\_code", "INT PRIMARY KEY", ""},  {"move\_in\_date", "DATE", ""},  {"move\_out\_date", "DATE", ""}  }  }  };  for (const auto& [table, fields] : default\_schema)  {  if (schema.contains(table))  {  continue;  }  schema[table] = fields;  data["schema"][table] = json::array();  for (const auto& field : fields)  {  data["schema"][table].push\_back(field.to\_json());  }  string create\_stmt = "CREATE TABLE IF NOT EXISTS " + table + " (";  vector<string> column\_defs;  for (const auto& field : fields)  {  column\_defs.push\_back(field.name + " " + field.type);  }  create\_stmt += join\_strings(column\_defs, ", ") + ");";  try  {  pqxx::work txn(\*conn);  txn.exec(create\_stmt);  txn.commit();  cout << "Таблица '" << table << "' успешно создана или уже существует.\n";  }  catch (const pqxx::sql\_error& e)  {  cout << "SQL ошибка при создании таблицы '" << table << "': " << e.what() << '\n';  cout << "Запрос: " << e.query() << '\n';  }  catch (const exception& e)  {  cout << "Ошибка при создании таблицы '" << table << "': " << e.what() << '\n';  }  }  save();  cout << "Схема базы данных обновлена и сохранена в 'database.json'.\n";  }  [[nodiscard]] bool is\_group\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["groups"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool group\_exists(const int code) const  {  return std::ranges::any\_of(data["groups"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_group(const Group& group)  {  if (!is\_group\_code\_unique(group.code))  {  cout << "Группа с таким кодом уже существует. Добавление отменено.\n";  return;  }  data["groups"].push\_back(group.to\_json());  save();  }  void delete\_group(int code)  {  if (!group\_exists(code))  {  std::cout << "Группа с кодом " << code << " не найдена.\n";  return;  }  bool has\_students = std::any\_of(data["students"].begin(), data["students"].end(),  [code](const json& j) { return j["group\_code"] == code; });  if (has\_students)  {  std::cout << "Невозможно удалить группу, так как к ней привязаны студенты.\n";  return;  }  auto& groups = data["groups"];  groups.erase(  std::remove\_if(groups.begin(), groups.end(),  [code](const json& j) { return j["code"] == code; }),  groups.end());  save();  std::cout << "Группа удалена.\n";  }  void edit\_group(const int code, const Group& new\_group)  {  auto& groups = data["groups"];  for (auto& j : groups)  {  if (j["code"] == code)  {  if (new\_group.code != code && !is\_group\_code\_unique(new\_group.code))  {  cout << "Группа с новым кодом уже существует. Изменение отменено.\n";  return;  }  j = new\_group.to\_json();  save();  return;  }  }  cout << "Группа с кодом " << code << " не найдена.\n";  }  [[nodiscard]] bool is\_dormitory\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["dormitories"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool dormitory\_exists(const int code) const  {  return std::ranges::any\_of(data["dormitories"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_dormitory(const Dormitory& dorm)  {  if (!is\_dormitory\_code\_unique(dorm.code))  {  cout << "Общежитие с таким кодом уже существует. Добавление отменено.\n";  return;  }  data["dormitories"].push\_back(dorm.to\_json());  save();  }  void delete\_dormitory(int code)  {  if (!dormitory\_exists(code))  {  std::cout << "Общежитие с кодом " << code << " не найдено.\n";  return;  }  bool has\_rooms = std::any\_of(data["rooms"].begin(), data["rooms"].end(),  [code](const json& j) { return j["dormitory\_code"] == code; });  if (has\_rooms)  {  std::cout << "Невозможно удалить общежитие, так как к нему привязаны комнаты.\n";  return;  }  auto& dorms = data["dormitories"];  dorms.erase(  std::remove\_if(dorms.begin(), dorms.end(),  [code](const json& j) { return j["code"] == code; }),  dorms.end());  save();  std::cout << "Общежитие удалено.\n";  }  void edit\_dormitory(const int code, const Dormitory& new\_dorm)  {  auto& dorms = data["dormitories"];  for (auto& j : dorms)  {  if (j["code"] == code)  {  if (new\_dorm.code != code && !is\_dormitory\_code\_unique(new\_dorm.code))  {  cout << "Общежитие с новым кодом уже существует. Изменение отменено.\n";  return;  }  j = new\_dorm.to\_json();  save();  return;  }  }  cout << "Общежитие с кодом " << code << " не найдено.\n";  }  [[nodiscard]] bool is\_room\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["rooms"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool room\_exists(const int code) const  {  return std::ranges::any\_of(data["rooms"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_room(const Room& room)  {  if (!is\_room\_code\_unique(room.code))  {  cout << "Комната с таким кодом уже существует. Добавление отменено.\n";  return;  }  if (!dormitory\_exists(room.dormitory\_code))  {  cout << "Общежитие с кодом " << room.dormitory\_code << " не существует. Добавление комнаты отменено.\n";  return;  }  data["rooms"].push\_back(room.to\_json());  save();  }  void delete\_room(int code)  {  if (!room\_exists(code))  {  std::cout << "Комната с кодом " << code << " не найдена.\n";  return;  }  bool has\_accommodations = std::any\_of(data["accommodations"].begin(), data["accommodations"].end(),  [code](const json& j) { return j["room\_code"] == code; });  if (has\_accommodations)  {  std::cout << "Невозможно удалить комнату, так как к ней привязаны записи заселения.\n";  return;  }  auto& rooms = data["rooms"];  rooms.erase(  std::remove\_if(rooms.begin(), rooms.end(),  [code](const json& j) { return j["code"] == code; }),  rooms.end());  save();  std::cout << "Комната удалена.\n";  }  void edit\_room(const int code, const Room& new\_room)  {  auto& rooms = data["rooms"];  for (auto& j : rooms)  {  if (j["code"] == code)  {  if (new\_room.code != code && !is\_room\_code\_unique(new\_room.code))  {  cout << "Комната с новым кодом уже существует. Изменение отменено.\n";  return;  }  if (!dormitory\_exists(new\_room.dormitory\_code))  {  cout << "Общежитие с кодом " << new\_room.dormitory\_code << " не существует. Изменение комнаты отменено.\n";  return;  }  j = new\_room.to\_json();  save();  return;  }  }  cout << "Комната с кодом " << code << " не найдена.\n";  }  [[nodiscard]] bool is\_student\_code\_unique(const int code) const  {  return std::ranges::all\_of(data["students"], [code](auto j)  {  return j["code"].template get<int>() != code;  });  }  [[nodiscard]] bool student\_exists(const int code) const  {  return std::ranges::any\_of(data["students"], [code](auto j)  {  return j["code"].template get<int>() == code;  });  }  void add\_student(const Student& student)  {  if (!is\_student\_code\_unique(student.code))  {  cout << "Студент с таким кодом уже существует. Добавление отменено.\n";  return;  }  if (!group\_exists(student.group\_code))  {  cout << "Группа с кодом " << student.group\_code << " не существует. Добавление студента отменено.\n";  return;  }  data["students"].push\_back(student.to\_json());  save();  }  void delete\_student(int code)  {  if (!student\_exists(code))  {  std::cout << "Студент с кодом " << code << " не найден.\n";  return;  }  bool has\_accommodation = std::any\_of(data["accommodations"].begin(), data["accommodations"].end(),  [code](const json& j) { return j["student\_code"] == code; });  if (has\_accommodation)  {  std::cout << "Невозможно удалить студента, так как к нему привязана информация о заселении.\n";  return;  }  auto& students = data["students"];  students.erase(  std::remove\_if(students.begin(), students.end(),  [code](const json& j) { return j["code"] == code; }),  students.end());  save();  std::cout << "Студент удален.\n";  }  void edit\_student(const int code, const Student& new\_student)  {  auto& students = data["students"];  for (auto& j : students)  {  if (j["code"] == code)  {  if (new\_student.code != code && !is\_student\_code\_unique(new\_student.code))  {  cout << "Студент с новым кодом уже существует. Изменение отменено.\n";  return;  }  if (!group\_exists(new\_student.group\_code))  {  cout << "Группа с кодом " << new\_student.group\_code << " не существует. Изменение студента отменено.\n";  return;  }  j = new\_student.to\_json();  save();  return;  }  }  cout << "Студент с кодом " << code << " не найден.\n";  }  [[nodiscard]] bool is\_accommodation\_unique(const int student\_code) const  {  return std::ranges::all\_of(data["accommodations"], [student\_code](auto j)  {  return j["student\_code"].template get<int>() != student\_code;  });  }  void add\_accommodation(const AccommodationInfo& acc)  {  if (!room\_exists(acc.room\_code))  {  cout << "Комната с кодом " << acc.room\_code << " не существует. Добавление информации о заселении отменено.\n";  return;  }  if (!student\_exists(acc.student\_code))  {  cout << "Студент с кодом " << acc.student\_code << " не существует. Добавление информации о заселении отменено.\n";  return;  }  if (!is\_accommodation\_unique(acc.student\_code))  {  cout << "Информация о заселении для этого студента уже существует. Добавление отменено.\n";  return;  }  data["accommodations"].push\_back(acc.to\_json());  save();  }  void delete\_accommodation(int student\_code)  {  auto& accs = data["accommodations"];  auto it = std::find\_if(accs.begin(), accs.end(),  [student\_code](const json& j) { return j["student\_code"] == student\_code; });  if (it == accs.end())  {  std::cout << "Информация о заселении для студента с кодом " << student\_code << " не найдена.\n";  return;  }  accs.erase(it);  save();  std::cout << "Информация о заселении удалена.\n";  }  void edit\_accommodation(const int student\_code, const AccommodationInfo& new\_acc)  {  auto& accs = data["accommodations"];  for (auto& j : accs)  {  if (j["student\_code"] == student\_code)  {  if (!room\_exists(new\_acc.room\_code))  {  cout << "Комната с кодом " << new\_acc.room\_code << " не существует. Изменение информации о заселении отменено.\n";  return;  }  if (!student\_exists(new\_acc.student\_code))  {  cout << "Студент с кодом " << new\_acc.student\_code << " не существует. Изменение информации о заселении отменено.\n";  return;  }  j = new\_acc.to\_json();  save();  return;  }  }  cout << "Информация о заселении для студента с кодом " << student\_code << " не найдена.\n";  }  void display\_groups() const  {  const string header = std::format("{:<10} {:<30} {:<30}\n", "Код", "Название группы", "Факультет");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["groups"])  {  string row = std::format("{:<10} {:<30} {:<30}\n",  j["code"].get<int>(),  j["name"].get<string>(),  j["faculty"].get<string>());  cout << row;  }  }  void display\_dormitories() const  {  const string header = std::format("{:<10} {:<30} {:<50}\n", "Код", "Название", "Адрес");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["dormitories"])  {  string row = std::format("{:<10} {:<30} {:<50}\n",  j["code"].get<int>(),  j["name"].get<string>(),  j["address"].get<string>());  cout << row;  }  }  void display\_rooms() const  {  const string header = std::format("{:<10} {:<20} {:<20}\n", "Код", "Номер комнаты", "Код общежития");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["rooms"])  {  string row = std::format("{:<10} {:<20} {:<20}\n",  j["code"].get<int>(),  j["number"].get<string>(),  j["dormitory\_code"].get<int>());  cout << row;  }  }  void display\_students() const  {  const string header = std::format("{:<10} {:<30} {:<15} {:<10} {:<10} {:<10} {:<15} {:<30} {:<15}\n",  "Код", "ФИО", "Дата рождения", "Пол", "Код группы",  "Серия", "Номер паспорта", "Кем выдан", "Дата выдачи");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["students"])  {  string row = std::format("{:<10} {:<30} {:<15} {:<10} {:<10} {:<10} {:<15} {:<30} {:<15}\n",  j["code"].get<int>(),  j["full\_name"].get<string>(),  j["birth\_date"].get<string>(),  j["gender"].get<string>(),  j["group\_code"].get<int>(),  j["passport\_series"].get<string>(),  j["passport\_number"].get<string>(),  j["issued\_by"].get<string>(),  j["issued\_date"].get<string>());  cout << row;  }  }  void display\_accommodations() const  {  const string header = std::format("{:<15} {:<15} {:<15} {:<15}\n", "Код комнаты", "Код студента", "Дата заселения",  "Дата выселения");  cout << header;  cout << string(header.size() - 1, '-') << '\n';  for (const auto& j : data["accommodations"])  {  string row = std::format("{:<15} {:<15} {:<15} {:<15}\n",  j["room\_code"].get<int>(),  j["student\_code"].get<int>(),  j["move\_in\_date"].get<string>(),  j["move\_out\_date"].get<string>());  cout << row;  }  }  void add\_field(const string& table)  {  Field f;  f.name = get\_valid\_string("Введите имя поля: ");  f.type = get\_valid\_string("Введите тип поля: ");  f.format = get\_valid\_string("Введите формат поля: ");  schema[table].emplace\_back(f);  if (!data["data"].contains(table))  {  data["data"][table] = json::array();  }  save();  cout << "Поле добавлено.\n";  }  void delete\_field(const string& table)  {  if (!schema.contains(table))  {  cout << "Таблица " << table << " не существует в схеме.\n";  return;  }  string field\_name = get\_valid\_string("Введите имя поля для удаления: ");  auto& fields = schema[table];  auto it = remove\_if(fields.begin(), fields.end(), [&](const Field& f) { return f.name == field\_name; });  if (it != fields.end())  {  fields.erase(it, fields.end());  for (auto& row : data["data"][table])  {  row.erase(field\_name);  }  save();  cout << "Поле удалено.\n";  }  else  {  cout << "Поле не найдено.\n";  }  }  void edit\_field(const string& table)  {  if (!schema.contains(table))  {  cout << "Таблица " << table << " не существует в схеме.\n";  return;  }  string field\_name = get\_valid\_string("Введите имя поля для изменения: ");  auto& fields = schema[table];  for (auto& field : fields)  {  if (field.name == field\_name)  {  field.type = get\_valid\_string("Введите новый тип поля: ");  field.format = get\_valid\_string("Введите новый формат поля: ");  save();  cout << "Поле изменено.\n";  return;  }  }  cout << "Поле не найдено.\n";  }  void special\_menu()  {  int choice;  do  {  cout << "\n--- Специальное меню ---\n";  cout << "1. Добавить поле\n";  cout << "2. Удалить поле\n";  cout << "3. Изменить поле\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  string table = get\_valid\_string("Введите название таблицы: ");  add\_field(table);  }  else if (choice == 2)  {  string table = get\_valid\_string("Введите название таблицы: ");  delete\_field(table);  }  else if (choice == 3)  {  string table = get\_valid\_string("Введите название таблицы: ");  edit\_field(table);  }  }  while (choice != 0);  }  void export\_to\_postgresql()  {  if (!conn || !conn->is\_open())  {  cout << "Подключение к PostgreSQL не установлено.\n";  return;  }  try  {  // Ensure all default tables are created before exporting data  create\_default\_tables();  // Define default tables to process  static const vector<string> default\_tables = {"groups", "dormitories", "rooms", "students", "accommodations"};  // Start a single transaction for schema synchronization and data export  pqxx::work txn\_export(\*conn);  for (const auto& table : default\_tables)  {  string truncate\_query = "TRUNCATE TABLE " + table + " RESTART IDENTITY CASCADE;";  try  {  txn\_export.exec(truncate\_query);  cout << "Table '" << table << "' has been truncated.\n";  }  catch (const pqxx::sql\_error& e)  {  cout << "SQL error while truncating table '" << table << "': " << e.what() << '\n';  cout << "Query: " << e.query() << '\n';  // Decide whether to continue or abort based on your requirements  // For now, we'll continue with other tables  }  catch (const exception& e)  {  cout << "Error while truncating table '" << table << "': " << e.what() << '\n';  }  }  for (const auto& table : default\_tables)  {  if (!schema.contains(table))  {  cout << "Схема для таблицы '" << table << "' не найдена. Пропуск.\n";  continue;  }  auto& fields = schema[table];  // Retrieve current columns from PostgreSQL table  string get\_columns\_query = "SELECT column\_name FROM information\_schema.columns WHERE table\_name = '" + table + "';";  pqxx::result res\_columns = txn\_export.exec(get\_columns\_query);  vector<string> existing\_columns;  for (const auto& row : res\_columns)  {  existing\_columns.emplace\_back(row[0].as<string>());  }  // Identify additional fields not present in PostgreSQL table  vector<Field> additional\_data;  for (const auto& field : fields)  {  if (find(existing\_columns.begin(), existing\_columns.end(), field.name) == existing\_columns.end())  {  additional\_data.emplace\_back(field);  }  }  // Add missing columns to PostgreSQL table  for (const auto& field : additional\_data)  {  string alter\_stmt = "ALTER TABLE " + table + " ADD COLUMN " + field.name + " " + field.type + ";";  try  {  txn\_export.exec(alter\_stmt);  cout << "Добавлено новое поле '" << field.name << "' в таблицу '" << table << "'.\n";  }  catch (const pqxx::sql\_error& e)  {  cout << "SQL ошибка при добавлении поля '" << field.name << "' в таблицу '" << table << "': " << e.what() << '\n';  cout << "Запрос: " << e.query() << '\n';  // Continue with other fields/tables  }  catch (const exception& e)  {  cout << "Ошибка при добавлении поля '" << field.name << "' в таблицу '" << table << "': " << e.what() << '\n';  }  }  fields = schema[table];  // Now, prepare the INSERT query dynamically  // List of columns  vector<string> column\_names;  // List of placeholders ($1, $2, ...)  vector<string> placeholders;  for (size\_t i = 0; i < fields.size(); ++i)  {  column\_names.emplace\_back(fields[i].name);  placeholders.emplace\_back("$" + to\_string(i + 1));  }  string insert\_query = "INSERT INTO " + table + " (" + join\_strings(column\_names, ", ") + ") VALUES (" +  join\_strings(placeholders, ", ") + ") ON CONFLICT DO NOTHING;";  // Iterate over each record in the table  for (const auto& record : data[table])  {  pqxx::params params;  bool skip\_record = false;  for (const auto& field : fields)  {  if (default\_fields.at(table).end() == std::find(default\_fields.at(table).begin(), default\_fields.at(table).end(),  field.name))  {  // Is additional  if (data["additional\_data"][table].contains(to\_string(record["code"])) && data["additional\_data"][table][  to\_string(record["code"])].contains(field.name) && !data["additional\_data"][table][  to\_string(record["code"])][field.name].is\_null())  {  if (field.type.find("INT") != string::npos)  {  params.append(  to\_string(data["additional\_data"][table][to\_string(record["code"])][field.name].get<int>()));  }  else if (field.type.find("DATE") != string::npos)  {  // Assuming date is stored as "DD.MM.YYYY" in JSON, convert to "YYYY-MM-DD" for PostgreSQL  string date = data["additional\_data"][table][to\_string(record["code"])][field.name].get<string>();  if (date.size() == 10 && date[2] == '.' && date[5] == '.')  {  string formatted\_date = date.substr(6, 4) + "-" + date.substr(3, 2) + "-" + date.substr(0, 2);  params.append(formatted\_date);  }  else  {  // Handle invalid date formats as NULL or skip  params.append(""); // Alternatively, you can decide how to handle this  }  }  else  {  // For VARCHAR and other string types  string value = data["additional\_data"][table][to\_string(record["code"])][field.name].get<string>();  // Escape single quotes by doubling them  size\_t pos = 0;  while ((pos = value.find("'", pos)) != string::npos)  {  value.insert(pos, "'");  pos += 2; // Move past the escaped quote  }  params.append(value);  }  }  else  {  params.append(""); // Represent NULL as empty string, PostgreSQL will interpret appropriately  }  }  else  {  if (record.contains(field.name) && !record[field.name].is\_null())  {  if (field.type.find("INT") != string::npos)  {  params.append(to\_string(record[field.name].get<int>()));  }  else if (field.type.find("DATE") != string::npos)  {  // Assuming date is stored as "DD.MM.YYYY" in JSON, convert to "YYYY-MM-DD" for PostgreSQL  string date = record[field.name].get<string>();  if (date.size() == 10 && date[2] == '.' && date[5] == '.')  {  string formatted\_date = date.substr(6, 4) + "-" + date.substr(3, 2) + "-" + date.substr(0, 2);  params.append(formatted\_date);  }  else  {  // Handle invalid date formats as NULL or skip  params.append(""); // Alternatively, you can decide how to handle this  }  }  else  {  // For VARCHAR and other string types  string value = record[field.name].get<string>();  // Escape single quotes by doubling them  size\_t pos = 0;  while ((pos = value.find("'", pos)) != string::npos)  {  value.insert(pos, "'");  pos += 2; // Move past the escaped quote  }  params.append(value);  }  }  else  {  params.append(""); // Represent NULL as empty string, PostgreSQL will interpret appropriately  }  }  }  // Execute the INSERT query with parameters  try  {  txn\_export.exec(insert\_query, params);  }  catch (const pqxx::sql\_error& e)  {  cout << "SQL ошибка при вставке записи в таблицу '" << table << "': " << e.what() << '\n';  cout << "Запрос: " << e.query() << '\n';  // Continue with other records  }  catch (const exception& e)  {  cout << "Ошибка при вставке записи в таблицу '" << table << "': " << e.what() << '\n';  }  }  }  // Commit the export transaction  txn\_export.commit();  // Now handle additional\_data export  cout << "Экспорт данных в PostgreSQL выполнен успешно.\n";  }  catch (const exception& e)  {  cout << "Ошибка PostgreSQL: " << e.what() << '\n';  }  }  void import\_from\_postgresql()  {  if (!conn || !conn->is\_open())  {  std::cout << "PostgreSQL connection is not established.\n";  return;  }  try  {  // Define default tables to process  static const std::vector<std::string> default\_tables = {"groups", "dormitories", "rooms", "students", "accommodations"};  // Initialize additional\_data if it doesn't exist  if (!data.contains("additional\_data"))  {  data["additional\_data"] = json::object();  }  // Begin transaction for importing schema and data  pqxx::work txn\_import(\*conn);  for (const auto& table : default\_tables)  {  // Fetch current columns from PostgreSQL table  std::string get\_columns\_query = "SELECT column\_name, data\_type FROM information\_schema.columns WHERE table\_name = '" + table  + "';";  pqxx::result res\_columns = txn\_import.exec(get\_columns\_query);  data["schema"][table] = json::array();  schema[table].clear(); // Clear existing fields in schema[table]  for (const auto& row : res\_columns)  {  std::string column\_name = row["column\_name"].as<std::string>();  std::string data\_type = row["data\_type"].as<std::string>();  // Map PostgreSQL data types to JSON types  std::string json\_type;  if (data\_type == "integer" || data\_type == "bigint" || data\_type == "smallint")  json\_type = "INT";  else if (data\_type.find("character varying") != std::string::npos || data\_type.find("varchar") != std::string::npos)  json\_type = "VARCHAR(255)";  else if (data\_type == "date")  json\_type = "DATE";  else  json\_type = "VARCHAR(255)"; // Default type  // Determine if the column is a primary key  bool is\_primary = false;  auto pk\_it = primary\_keys.find(table);  if (pk\_it != primary\_keys.end() && pk\_it->second == column\_name)  {  is\_primary = true;  }  // Create Field object  Field field;  field.name = column\_name;  field.type = json\_type;  //if (is\_primary)  //field.constraints = "PRIMARY KEY"; // If you have a constraints field  // Add field to schema[table]  schema[table].push\_back(field);  // Add field to JSON schema  data["schema"][table].push\_back(field.to\_json());  }  std::cout << "Schema for table '" << table << "' has been updated.\n";  auto fields = schema[table];  // Fetch data from the table  std::string select\_query = "SELECT \* FROM " + table + ";";  pqxx::result res\_data = txn\_import.exec(select\_query);  // Clear existing data for the table in JSON  data[table] = json::array();  // Get primary key for the table  std::string primary\_key = "";  auto pk\_it = primary\_keys.find(table);  if (pk\_it != primary\_keys.end())  {  primary\_key = pk\_it->second;  }  else  {  std::cout << "Primary key for table '" << table << "' is not defined. Skipping additional data handling.\n";  }  for (const auto& row : res\_data)  {  json record;  std::string record\_id\_str = "";  json additional\_record = json::object();  for (const auto& field : fields)  {  std::string column\_name = field.name;  std::string data\_type = field.type;  // Determine if the field is additional  if (default\_fields.at(table).end() == std::find(default\_fields.at(table).begin(), default\_fields.at(table).end(),  column\_name))  {  // It's an additional field  if (!row[column\_name].is\_null())  {  try  {  if (data\_type.find("INT") != std::string::npos)  {  int value = row[column\_name].as<int>();  additional\_record[column\_name] = value;  }  else if (data\_type.find("DATE") != std::string::npos)  {  std::string date = row[column\_name].as<std::string>();  if (date.size() == 10 && date[4] == '-' && date[7] == '-')  {  std::string formatted\_date = date.substr(8, 2) + "." + date.substr(5, 2) + "." + date.substr(0,  4);  additional\_record[column\_name] = formatted\_date;  }  else  {  additional\_record[column\_name] = date;  }  }  else  {  std::string value = row[column\_name].as<std::string>();  additional\_record[column\_name] = value;  }  std::cout << "Processing additional field '" << column\_name << "' with type '" << data\_type <<  "' and value '"  << (row[column\_name].is\_null() ? "NULL" : row[column\_name].as<std::string>()) << "'.\n";  }  catch (const std::exception& e)  {  std::cout << "Error processing additional field '" << column\_name << "': " << e.what() << '\n';  additional\_record[column\_name] = nullptr;  }  }  }  else  {  // Access the column by name to avoid index mismatches  if (row[column\_name].is\_null())  {  record[column\_name] = nullptr;  }  else  {  try  {  if (data\_type.find("INT") != std::string::npos)  {  int value = row[column\_name].as<int>();  record[column\_name] = value;  if (column\_name == primary\_key)  record\_id\_str = std::to\_string(value);  }  else if (data\_type.find("DATE") != std::string::npos)  {  std::string date = row[column\_name].as<std::string>();  // Convert from "YYYY-MM-DD" to "DD.MM.YYYY"  if (date.size() == 10 && date[4] == '-' && date[7] == '-')  {  std::string formatted\_date = date.substr(8, 2) + "." + date.substr(5, 2) + "." + date.substr(0,  4);  record[column\_name] = formatted\_date;  }  else  {  record[column\_name] = date; // Keep as is if format is unexpected  }  }  else  {  std::string value = row[column\_name].as<std::string>();  record[column\_name] = value;  }  // Log processing of the field  std::cout << "Processing field '" << column\_name << "' with type '" << data\_type << "' and value '"  << (row[column\_name].is\_null() ? "NULL" : row[column\_name].as<std::string>()) << "'.\n";  }  catch (const std::exception& e)  {  std::cout << "Error processing field '" << column\_name << "': " << e.what() << '\n';  record[column\_name] = nullptr; // Assign null on error  }  }  }  }  data[table].push\_back(record);  std::cout << "Added record to table '" << table << "' with ID '" << record\_id\_str << "'.\n";  // Save additional fields  if (!additional\_record.empty() && !record\_id\_str.empty())  {  if (!data["additional\_data"].contains(table))  {  data["additional\_data"][table] = json::object();  }  data["additional\_data"][table][record\_id\_str] = additional\_record;  std::cout << "Added additional data for record with " << primary\_key << " = " << record\_id\_str << " in table '" <<  table << "'.\n";  }  }  }  // Commit the transaction  txn\_import.commit();  std::cout << "Data import from PostgreSQL completed successfully.\n";  // Save the updated JSON to file  save();  std::cout << "Data saved to JSON file.\n";  }  catch (const exception& e)  {  cout << "Ошибка при импорте данных из PostgreSQL: " << e.what() << '\n';  }  }  void display\_table(const string& table) const  {  if (!data["data"].contains(table))  {  cout << "Таблица не существует.\n";  return;  }  if (!schema.contains(table))  {  cout << "Схема таблицы не определена.\n";  return;  }  const auto& fields = schema.at(table);  string header;  for (auto& field : fields)  {  header += std::format("{:<20} ", field.name);  }  cout << header << '\n';  cout << string(header.size(), '-') << '\n';  for (auto& row : data["data"][table])  {  string line;  for (auto& field : fields)  {  if (row.contains(field.name))  {  if (row[field.name].is\_null())  line += std::format("{:<20} ", "NULL");  else  line += std::format("{:<20} ", row[field.name].get<string>());  }  else  {  line += std::format("{:<20} ", "NULL");  }  }  cout << line << '\n';  }  }  void menu\_groups()  {  int choice;  do  {  cout << "\n--- Управление группами ---\n";  cout << "1. Добавить группу\n";  cout << "2. Удалить группу\n";  cout << "3. Изменить группу\n";  cout << "4. Показать все группы\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Group g;  while (true)  {  g.code = get\_valid\_int("Введите код группы: ");  if (is\_group\_code\_unique(g.code))  break;  cout << "Группа с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  g.name = get\_valid\_string("Введите название группы: ");  g.faculty = get\_valid\_string("Введите название факультета: ");  add\_group(g);  cout << "Группа добавлена.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код группы для удаления: ");  delete\_group(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код группы для изменения: ");  if (!group\_exists(code))  {  cout << "Группа с таким кодом не существует.\n";  continue;  }  Group g;  while (true)  {  g.code = get\_valid\_int("Введите новый код группы: ");  if (g.code == code || is\_group\_code\_unique(g.code))  break;  cout << "Группа с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  g.name = get\_valid\_string("Введите новое название группы: ");  g.faculty = get\_valid\_string("Введите новое название факультета: ");  edit\_group(code, g);  cout << "Группа изменена.\n";  }  else if (choice == 4)  {  display\_groups();  }  }  while (choice != 0);  }  void menu\_dormitories()  {  int choice;  do  {  cout << "\n--- Управление общежитиями ---\n";  cout << "1. Добавить общежитие\n";  cout << "2. Удалить общежитие\n";  cout << "3. Изменить общежитие\n";  cout << "4. Показать все общежития\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Dormitory d;  while (true)  {  d.code = get\_valid\_int("Введите код общежития: ");  if (is\_dormitory\_code\_unique(d.code))  break;  cout << "Общежитие с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  d.name = get\_valid\_string("Введите название общежития: ");  d.address = get\_valid\_string("Введите адрес общежития: ");  add\_dormitory(d);  cout << "Общежитие добавлено.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код общежития для удаления: ");  delete\_dormitory(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код общежития для изменения: ");  if (!dormitory\_exists(code))  {  cout << "Общежитие с таким кодом не существует.\n";  continue;  }  Dormitory d;  while (true)  {  d.code = get\_valid\_int("Введите новый код общежития: ");  if (d.code == code || is\_dormitory\_code\_unique(d.code))  break;  cout << "Общежитие с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  d.name = get\_valid\_string("Введите новое название общежития: ");  d.address = get\_valid\_string("Введите новый адрес общежития: ");  edit\_dormitory(code, d);  cout << "Общежитие изменено.\n";  }  else if (choice == 4)  {  display\_dormitories();  }  }  while (choice != 0);  }  void menu\_rooms()  {  int choice;  do  {  cout << "\n--- Управление комнатами ---\n";  cout << "1. Добавить комнату\n";  cout << "2. Удалить комнату\n";  cout << "3. Изменить комнату\n";  cout << "4. Показать все комнаты\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Room r;  while (true)  {  r.code = get\_valid\_int("Введите код комнаты: ");  if (is\_room\_code\_unique(r.code))  break;  cout << "Комната с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  r.number = get\_valid\_string("Введите номер комнаты: ");  while (true)  {  r.dormitory\_code = get\_valid\_int("Введите код общежития: ");  if (dormitory\_exists(r.dormitory\_code))  break;  cout << "Общежитие с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  add\_room(r);  cout << "Комната добавлена.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код комнаты для удаления: ");  delete\_room(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код комнаты для изменения: ");  if (!room\_exists(code))  {  cout << "Комната с таким кодом не существует.\n";  continue;  }  Room r;  while (true)  {  r.code = get\_valid\_int("Введите новый код комнаты: ");  if (r.code == code || is\_room\_code\_unique(r.code))  break;  cout << "Комната с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  r.number = get\_valid\_string("Введите новый номер комнаты: ");  while (true)  {  r.dormitory\_code = get\_valid\_int("Введите новый код общежития: ");  if (dormitory\_exists(r.dormitory\_code))  break;  cout << "Общежития с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  edit\_room(code, r);  cout << "Комната изменена.\n";  }  else if (choice == 4)  {  display\_rooms();  }  }  while (choice != 0);  }  void menu\_students()  {  int choice;  do  {  cout << "\n--- Управление студентами ---\n";  cout << "1. Добавить студента\n";  cout << "2. Удалить студента\n";  cout << "3. Изменить студента\n";  cout << "4. Показать всех студентов\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  Student s;  while (true)  {  s.code = get\_valid\_int("Введите код студента: ");  if (is\_student\_code\_unique(s.code))  break;  cout << "Студент с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  s.full\_name = get\_valid\_string("Введите ФИО студента: ");  s.birth\_date = get\_valid\_date("Введите дату рождения (ДД.ММ.ГГГГ): ");  s.gender = get\_valid\_string("Введите пол: ");  while (true)  {  s.group\_code = get\_valid\_int("Введите код группы: ");  if (group\_exists(s.group\_code))  break;  cout << "Группа с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  s.passport\_series = get\_valid\_string("Введите серию паспорта: ");  s.passport\_number = get\_valid\_string("Введите номер паспорта: ");  s.issued\_by = get\_valid\_string("Кем выдан паспорт: ");  s.issued\_date = get\_valid\_date("Дата выдачи паспорта (ДД.ММ.ГГГГ): ");  add\_student(s);  cout << "Студент добавлен.\n";  }  else if (choice == 2)  {  const int code = get\_valid\_int("Введите код студента для удаления: ");  delete\_student(code);  }  else if (choice == 3)  {  const int code = get\_valid\_int("Введите код студента для изменения: ");  if (!student\_exists(code))  {  cout << "Студент с таким кодом не существует.\n";  continue;  }  Student s;  while (true)  {  s.code = get\_valid\_int("Введите новый код студента: ");  if (s.code == code || is\_student\_code\_unique(s.code))  break;  cout << "Студент с таким кодом уже существует. Пожалуйста, введите другой код.\n";  }  s.full\_name = get\_valid\_string("Введите новое ФИО студента: ");  s.birth\_date = get\_valid\_date("Введите новую дату рождения (ДД.ММ.ГГГГ): ");  s.gender = get\_valid\_string("Введите новый пол: ");  while (true)  {  s.group\_code = get\_valid\_int("Введите новый код группы: ");  if (group\_exists(s.group\_code))  break;  cout << "Группа с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  s.passport\_series = get\_valid\_string("Введите новую серию паспорта: ");  s.passport\_number = get\_valid\_string("Введите новый номер паспорта: ");  s.issued\_by = get\_valid\_string("Кем выдан паспорт: ");  s.issued\_date = get\_valid\_date("Новая дата выдачи паспорта (ДД.ММ.ГГГГ): ");  edit\_student(code, s);  cout << "Студент изменен.\n";  }  else if (choice == 4)  {  display\_students();  }  }  while (choice != 0);  }  void menu\_accommodations()  {  int choice;  do  {  cout << "\n--- Управление заселением ---\n";  cout << "1. Добавить информацию о заселении\n";  cout << "2. Удалить информацию о заселении\n";  cout << "3. Изменить информацию о заселении\n";  cout << "4. Показать всю информацию о заселении\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  if (choice == 1)  {  AccommodationInfo a;  while (true)  {  a.room\_code = get\_valid\_int("Введите код комнаты: ");  if (room\_exists(a.room\_code))  break;  cout << "Комната с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  while (true)  {  a.student\_code = get\_valid\_int("Введите код студента: ");  if (student\_exists(a.student\_code))  {  if (is\_accommodation\_unique(a.student\_code))  {  break;  }  cout << "Информация о заселении для этого студента уже существует.\n";  return;  }  cout << "Студент с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  a.move\_in\_date = get\_valid\_date("Введите дату заселения (ДД.ММ.ГГГГ): ");  a.move\_out\_date = get\_valid\_date("Введите дату выселения (ДД.ММ.ГГГГ): ");  add\_accommodation(a);  cout << "Информация о заселении добавлена.\n";  }  else if (choice == 2)  {  const int student\_code = get\_valid\_int("Введите код студента для удаления информации о заселении: ");  delete\_accommodation(student\_code);  }  else if (choice == 3)  {  const int student\_code = get\_valid\_int("Введите код студента для изменения информации о заселении: ");  if (!is\_accommodation\_unique(student\_code) && student\_exists(student\_code))  {  AccommodationInfo a;  while (true)  {  a.room\_code = get\_valid\_int("Введите новый код комнаты: ");  if (room\_exists(a.room\_code))  break;  cout << "Комната с таким кодом не существует. Пожалуйста, введите другой код.\n";  }  a.student\_code = student\_code;  a.move\_in\_date = get\_valid\_date("Введите новую дату заселения (ДД.ММ.ГГГГ): ");  a.move\_out\_date = get\_valid\_date("Введите новую дату выселения (ДД.ММ.ГГГГ): ");  edit\_accommodation(student\_code, a);  cout << "Информация о заселении изменена.\n";  }  else  {  cout << "Информация о заселении для этого студента не существует.\n";  }  }  else if (choice == 4)  {  display\_accommodations();  }  }  while (choice != 0);  }  void add\_field\_menu()  {  string table = get\_valid\_string("Введите название таблицы: ");  add\_field(table);  }  void delete\_field\_menu()  {  string table = get\_valid\_string("Введите название таблицы: ");  delete\_field(table);  }  void edit\_field\_menu()  {  string table = get\_valid\_string("Введите название таблицы: ");  edit\_field(table);  }  void special\_menu\_functions()  {  int choice;  do  {  cout << "\n--- Специальное меню ---\n";  cout << "1. Добавить поле\n";  cout << "2. Удалить поле\n";  cout << "3. Изменить поле\n";  cout << "0. Назад\n";  cout << "Выберите действие: ";  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  switch (choice)  {  case 1:  add\_field\_menu();  break;  case 2:  delete\_field\_menu();  break;  case 3:  edit\_field\_menu();  break;  case 0:  break;  default:  cout << "Неверный выбор. Попробуйте снова.\n";  }  }  while (choice != 0);  }  void export\_to\_postgresql\_func()  {  export\_to\_postgresql();  }  void import\_from\_postgresql\_func()  {  import\_from\_postgresql();  }  void menu\_edit\_additional\_fields()  {  string table = get\_valid\_string("Введите название таблицы: ");  // Check if the table exists  if (!data.contains(table))  {  cout << "Таблица '" << table << "' не существует.\n";  return;  }  // Get the primary key for the table  if (!primary\_keys.contains(table))  {  cout << "Для таблицы '" << table << "' не определён первичный ключ.\n";  return;  }  string primary\_key = primary\_keys[table];  int code = get\_valid\_int("Введите " + primary\_key + " записи: ");  // Check if the record exists  auto& records = data[table];  auto it = std::find\_if(records.begin(), records.end(),  [&](const json& j) { return j[primary\_key] == code; });  if (it == records.end())  {  cout << "Запись с " << primary\_key << " = " << code << " не найдена в таблице '" << table << "'.\n";  return;  }  // Access the additional fields for this record  json& additional\_fields = data["additional\_data"][table][std::to\_string(code)];  if (additional\_fields.is\_null() || additional\_fields.empty())  {  cout << "Дополнительных полей для этой записи нет.\n";  }  else  {  cout << "Текущие дополнительные поля:\n";  for (auto it = additional\_fields.begin(); it != additional\_fields.end(); ++it)  {  cout << it.key() << " : " << it.value() << '\n';  }  }  // Provide options to the user  cout << "Выберите действие:\n";  cout << "1. Добавить новое дополнительное поле\n";  cout << "2. Изменить существующие дополнительные поля\n";  cout << "3. Удалить дополнительное поле\n";  cout << "0. Отмена\n";  int choice = get\_valid\_int("Введите ваш выбор: ");  if (choice == 1)  {  // Add new additional field  string field\_name = get\_valid\_string("Введите имя нового дополнительного поля: ");  string field\_value = get\_valid\_string("Введите значение поля: ");  additional\_fields[field\_name] = field\_value;  cout << "Дополнительное поле добавлено.\n";  }  else if (choice == 2)  {  // Edit existing additional fields  if (additional\_fields.empty())  {  cout << "Нет дополнительных полей для редактирования.\n";  return;  }  for (auto it = additional\_fields.begin(); it != additional\_fields.end(); ++it)  {  string field\_name = it.key();  string old\_value = it.value();  string new\_value = get\_valid\_string(  "Введите новое значение для '" + field\_name + "' (текущее значение: '" + old\_value + "'): ");  additional\_fields[field\_name] = new\_value;  }  cout << "Дополнительные поля обновлены.\n";  }  else if (choice == 3)  {  // Delete an additional field  if (additional\_fields.empty())  {  cout << "Нет дополнительных полей для удаления.\n";  return;  }  string field\_name = get\_valid\_string("Введите имя дополнительного поля для удаления: ");  if (additional\_fields.contains(field\_name))  {  additional\_fields.erase(field\_name);  cout << "Дополнительное поле '" << field\_name << "' удалено.\n";  }  else  {  cout << "Поле '" << field\_name << "' не найдено среди дополнительных полей.\n";  }  }  else  {  cout << "Операция отменена.\n";  return;  }  // Save changes  save();  cout << "Изменения сохранены.\n";  }  void display\_table\_func()  {  string table = get\_valid\_string("Введите название таблицы для отображения: ");  display\_table(table);  }  void main\_menu()  {  int choice;  do  {  string menu =  std::format("\n=== Учёт студентов в общежитиях ===\n"  "1. Управление группами\n"  "2. Управление общежитиями\n"  "3. Управление комнатами\n"  "4. Управление студентами\n"  "5. Управление заселением\n"  "6. Специальное меню\n"  "7. Экспорт данных в PostgreSQL\n"  "8. Импорт данных из PostgreSQL\n"  "9. Показать таблицу\n"  "10. Редактировать дополнительные поля\n"  "0. Выход\n"  "Выберите действие: ");  cout << menu;  cin >> choice;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  switch (choice)  {  case 1:  menu\_groups();  break;  case 2:  menu\_dormitories();  break;  case 3:  menu\_rooms();  break;  case 4:  menu\_students();  break;  case 5:  menu\_accommodations();  break;  case 6:  special\_menu\_functions();  break;  case 7:  export\_to\_postgresql\_func();  break;  case 8:  import\_from\_postgresql\_func();  break;  case 9:  display\_table\_func();  break;  case 10:  menu\_edit\_additional\_fields();  break;  case 0:  cout << "Выход из программы.\n";  break;  default:  cout << "Неверный выбор. Попробуйте снова.\n";  }  }  while (choice != 0);  }  };  int main()  {  try  {  DatabaseHandler db("database.json");  db.main\_menu();  }  catch (const std::format\_error& e)  {  cerr << "Ошибка форматирования: " << e.what() << '\n';  }  catch (const json::exception& e)  {  cerr << "Ошибка работы с JSON: " << e.what() << '\n';  }  catch (const exception& e)  {  cerr << "Произошла ошибка: " << e.what() << '\n';  }  return 0;  } |

# 4 Результаты работы программы

Взаимодействие с программой представлено ниже:

|  |
| --- |
| Подключение к PostgreSQL успешно.  === Учёт студентов в общежитиях ===  1. Управление группами  2. Управление общежитиями  3. Управление комнатами  4. Управление студентами  5. Управление заселением  6. Специальное меню  7. Экспорт данных в PostgreSQL  8. Импорт данных из PostgreSQL  9. Показать таблицу  10. Редактировать дополнительные поля  0. Выход  Выберите действие: 10  Введите название таблицы: groups  Введите code записи: 12  Текущие дополнительные поля:  test : "123"  Выберите действие:  1. Добавить новое дополнительное поле  2. Изменить существующие дополнительные поля  3. Удалить дополнительное поле  0. Отмена  Введите ваш выбор: 2  Введите новое значение для 'test' (текущее значение: '123'): uuu  Дополнительные поля обновлены.  Изменения сохранены.  === Учёт студентов в общежитиях ===  1. Управление группами  2. Управление общежитиями  3. Управление комнатами  4. Управление студентами  5. Управление заселением  6. Специальное меню  7. Экспорт данных в PostgreSQL  8. Импорт данных из PostgreSQL  9. Показать таблицу  10. Редактировать дополнительные поля  0. Выход  Выберите действие: 7  Схема базы данных обновлена и сохранена в 'database.json'.  Table 'groups' has been truncated.  Table 'dormitories' has been truncated.  Table 'rooms' has been truncated.  Table 'students' has been truncated.  Table 'accommodations' has been truncated.  Экспорт данных в PostgreSQL выполнен успешно.  === Учёт студентов в общежитиях ===  1. Управление группами  2. Управление общежитиями  3. Управление комнатами  4. Управление студентами  5. Управление заселением  6. Специальное меню  7. Экспорт данных в PostgreSQL  8. Импорт данных из PostgreSQL  9. Показать таблицу  10. Редактировать дополнительные поля  0. Выход  Выберите действие: 6  --- Специальное меню ---  1. Добавить поле  2. Удалить поле  3. Изменить поле  0. Назад  Выберите действие: 1  Введите название таблицы: groups  Введите имя поля: lol  Введите тип поля: int  Введите формат поля:  Ввод не может быть пустым. Пожалуйста, попробуйте снова.  Введите формат поля: 1  Поле добавлено.  --- Специальное меню ---  1. Добавить поле  2. Удалить поле  3. Изменить поле  0. Назад  Выберите действие: 3  Введите название таблицы: groups  Введите имя поля для изменения: lol  Введите новый тип поля: int  Введите новый формат поля: 1  Поле изменено.  --- Специальное меню ---  1. Добавить поле  2. Удалить поле  3. Изменить поле  0. Назад  Выберите действие: 0  === Учёт студентов в общежитиях ===  1. Управление группами  2. Управление общежитиями  3. Управление комнатами  4. Управление студентами  5. Управление заселением  6. Специальное меню  7. Экспорт данных в PostgreSQL  8. Импорт данных из PostgreSQL  9. Показать таблицу  10. Редактировать дополнительные поля  0. Выход  Выберите действие: 10  Введите название таблицы: groups  Введите code записи: 144  Текущие дополнительные поля:  test : "test"  Выберите действие:  1. Добавить новое дополнительное поле  2. Изменить существующие дополнительные поля  3. Удалить дополнительное поле  0. Отмена  Введите ваш выбор: 1  Введите имя нового дополнительного поля: lol  Введите значение поля: 42  Дополнительное поле добавлено.  Изменения сохранены.  === Учёт студентов в общежитиях ===  1. Управление группами  2. Управление общежитиями  3. Управление комнатами  4. Управление студентами  5. Управление заселением  6. Специальное меню  7. Экспорт данных в PostgreSQL  8. Импорт данных из PostgreSQL  9. Показать таблицу  10. Редактировать дополнительные поля  0. Выход  Выберите действие: 10  Введите название таблицы: groups  Введите code записи: 12  Текущие дополнительные поля:  test : "uuu"  Выберите действие:  1. Добавить новое дополнительное поле  2. Изменить существующие дополнительные поля  3. Удалить дополнительное поле  0. Отмена  Введите ваш выбор: 1  Введите имя нового дополнительного поля: lol  Введите значение поля: 19  Дополнительное поле добавлено.  Изменения сохранены.  === Учёт студентов в общежитиях ===  1. Управление группами  2. Управление общежитиями  3. Управление комнатами  4. Управление студентами  5. Управление заселением  6. Специальное меню  7. Экспорт данных в PostgreSQL  8. Импорт данных из PostgreSQL  9. Показать таблицу  10. Редактировать дополнительные поля  0. Выход  Выберите действие: 7  Схема базы данных обновлена и сохранена в 'database.json'.  Table 'groups' has been truncated.  Table 'dormitories' has been truncated.  Table 'rooms' has been truncated.  Table 'students' has been truncated.  Table 'accommodations' has been truncated.  Добавлено новое поле 'lol' в таблицу 'groups'.  Экспорт данных в PostgreSQL выполнен успешно. |

В результате изменяемая таблица в PostgreSQL выглядит так, как показано на рисунке 2.

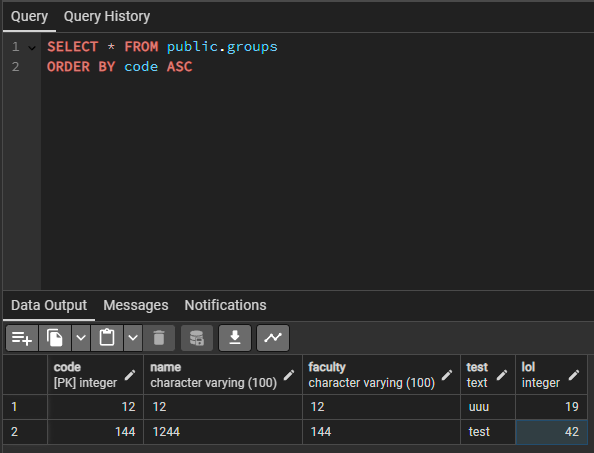


Рисунок 2 — Данные в PostgreSQL

JSON-файл же выглядит так, как показано ниже:

|  |
| --- |
| {  "accommodations": [],  "additional\_data": {  "groups": {  "12": {  "lol": "19",  "test": "uuu"  },  "144": {  "lol": "42",  "test": "test"  }  }  },  "dormitories": [],  "groups": [  {  "code": 12,  "faculty": "12",  "name": "12"  },  {  "code": 144,  "faculty": "144",  "name": "1244"  }  ],  "rooms": [],  "schema": {  "accommodations": [  {  "format": "",  "name": "room\_code",  "type": "INT"  },  {  "format": "",  "name": "student\_code",  "type": "INT"  },  {  "format": "",  "name": "move\_in\_date",  "type": "DATE"  },  {  "format": "",  "name": "move\_out\_date",  "type": "DATE"  }  ],  "dormitories": [  {  "format": "",  "name": "code",  "type": "INT"  },  {  "format": "",  "name": "name",  "type": "VARCHAR(255)"  },  {  "format": "",  "name": "address",  "type": "VARCHAR(255)"  }  ],  "groups": [  {  "format": "",  "name": "code",  "type": "INT"  },  {  "format": "",  "name": "name",  "type": "VARCHAR(255)"  },  {  "format": "",  "name": "faculty",  "type": "VARCHAR(255)"  },  {  "format": "",  "name": "test",  "type": "VARCHAR(255)"  }  ],  "rooms": [  {  "format": "",  "name": "code",  "type": "INT"  },  {  "format": "",  "name": "dormitory\_code",  "type": "INT"  },  {  "format": "",  "name": "number",  "type": "VARCHAR(255)"  }  ],  "students": [  {  "format": "",  "name": "issued\_date",  "type": "DATE"  },  {  "format": "",  "name": "birth\_date",  "type": "DATE"  },  {  "format": "",  "name": "group\_code",  "type": "INT"  },  {  "format": "",  "name": "code",  "type": "INT"  },  {  "format": "",  "name": "issued\_by",  "type": "VARCHAR(255)"  },  {  "format": "",  "name": "passport\_series",  "type": "VARCHAR(255)"  },  {  "format": "",  "name": "full\_name",  "type": "VARCHAR(255)"  },  {  "format": "",  "name": "passport\_number",  "type": "VARCHAR(255)"  },  {  "format": "",  "name": "gender",  "type": "VARCHAR(255)"  }  ]  },  "students": [  {  "birth\_date": "12.03.1322",  "code": 12,  "full\_name": "ФИО",  "gender": "М",  "group\_code": 12,  "issued\_by": "ГУ МВД Москвы",  "issued\_date": "12.04.1319",  "passport\_number": "242424",  "passport\_series": "1444"  }  ]  } |

# 5 Вывод

В процессе работы были изучены основные принципы взаимодействия приложений с СУБД.