Введение

Клиент-серверное приложение представляет собой программное обеспечение, архитектура которого предполагает разделение на две основные части: клиентскую и серверную. Каждая из этих частей выполняет свои специфические функции, которые в совокупности обеспечивают эффективное и надежное взаимодействие пользователя с приложением.

Клиентская часть (или просто клиент) является интерфейсом, с которым взаимодействует конечный пользователь. Основная задача клиентской части заключается в предоставлении удобных и интуитивно понятных инструментов для ввода, вывода и обработки данных. В рамках данного курсового проекта клиентская часть включает в себя таблицы и функциональные возможности для работы с ними, такие как поиск, фильтрация и сортировка данных. Эти инструменты позволяют пользователю оперативно и эффективно управлять данными, полученными от сервера.

Серверная часть (или сервер) отвечает за обработку запросов, поступающих от клиентов, и управление базой данных. В данном проекте сервер реализован на основе системы управления базами данных PostgreSQL (psql). Сервер принимает запросы от клиентов, выполняет операции с базой данных, такие как выборка, обновление, вставка и удаление данных, и возвращает результаты обратно клиентам. Важной задачей сервера является обеспечение целостности и безопасности данных, а также управление конкурентным доступом к ресурсам базы данных.

Для взаимодействия между клиентом и сервером разработан собственный протокол передачи информации. Этот протокол определяет формат и последовательность обмена данными, обеспечивая корректное и эффективное взаимодействие между компонентами приложения. Протокол включает в себя механизмы аутентификации и авторизации пользователей, обработки ошибок и управления сессиями, что способствует повышению уровня безопасности и надежности системы в целом.

Работа с базой данных PostgreSQL является ключевым элементом серверной части. PostgreSQL — это мощная, открытая и широко используемая система управления базами данных, которая поддерживает множество современных функций, таких как транзакции, сложные запросы и расширенные возможности индексирования. В данном проекте PostgreSQL используется для хранения и управления данными, а также для выполнения всех операций, связанных с их обработкой. Сервер обеспечивает взаимодействие с базой данных через SQL-запросы, предоставляя клиентам необходимую информацию и выполняя их запросы.

1. Постановка задачи

Целью данного курсового проекта является разработка клиент-серверного приложения на языке Java, обеспечивающего взаимодействие пользователей с базой данных. Приложение должно предоставлять функциональные возможности для управления данными на клиентской стороне и выполнять основные операции с базой данных на серверной стороне.

Требования к клиентской части:

Клиентская часть приложения должна предоставлять пользователям удобный и интуитивно понятный интерфейс для работы с таблицами базы данных. Основные функциональные возможности клиентской части включают:

- Добавление данных: возможность добавлять новые записи в таблицу базы данных.
- Удаление данных: возможность удалять существующие записи из таблицы.
- Редактирование данных: возможность редактировать данные в существующих записях таблицы.
- Поиск данных: возможность осуществлять поиск по определённым критериям в таблице.
- Фильтрация данных: возможность фильтрации данных в таблице по различным параметрам.

Требования к серверной части:

Серверная часть приложения должна обеспечивать взаимодействие с базой данных PostgreSQL и выполнять следующие задачи:

- Обработка запросов от клиентов: сервер должен принимать запросы от клиентов, выполнять соответствующие операции с базой данных и возвращать результаты.
- Управление базой данных: сервер должен обеспечивать добавление, удаление, редактирование, поиск и фильтрацию данных в базе данных.
- Уведомление клиентов: сервер должен уведомлять всех подключённых клиентов о любых изменениях, произошедших в таблице базы данных (например, при добавлении, удалении или редактировании записей).

Протокол взаимодействия:

Для обеспечения взаимодействия между клиентом и сервером необходимо разработать собственный протокол передачи данных. Этот протокол должен включать:

- Формат запросов и ответов: чётко определённый формат для отправки запросов от клиента на сервер и получения ответов от сервера.
- Механизмы аутентификации и авторизации: обеспечение безопасности и контроль доступа пользователей.
- Обработка ошибок: механизмы обработки и передачи сообщений об ошибках.
- Управление сессиями: поддержка активных сессий пользователей и управление их состоянием.

Ожидаемые результаты:

В результате выполнения данного курсового проекта должно быть создано клиент-серверное приложение, соответствующее указанным требованиям. Приложение должно быть реализовано на языке Java и обеспечивать:

- Надёжное и эффективное взаимодействие с базой данных PostgreSQL.
- Удобный и функциональный интерфейс для пользователей на клиентской стороне.
- Своевременное уведомление всех клиентов о любых изменениях в данных.
- Высокий уровень безопасности и защиты данных.

2. Описание алгоритма работы

Разработанная программа основывается на вышеописанном протоколе, состоящая из клиента и сервера.

Сначала запускаем сервер и создаем сокет. Устанавливаем локальную точку для прослушивания подключений. Запускаем прослушивание входящих подключений и принимаем новых клиентов в бесконечном цикле. После того, как клиент был принят, он передается на обработку новый поток в функцию обработки клиентов. На этом установление соединения между клиентом и сервером заканчивается. После установления соединения сервер в бесконечном цикле ожидает команды от клиента.

Серверная часть обрабатывает запросы от клиентов и выполняет операции с базой данных PostgreSQL с использованием SQL-запросов. База данных PostgreSQL хранит и управляет всей информацией, предоставляя серверу возможность манипулировать данными в соответствии с запросами клиентов.

Запускаем клиент и подключаемся к серверу. Создаем новый поток для приема сообщения и ожидаем сообщения в бесконечном цикле.

2.1. Разработка Сервера

2.1.1. Подключение сторонних библиотек

1) Для работы с базой данных PostgreSQL в проект необходимо добавить драйвер JDBC (Java Database Connectivity), в библиотеках проекта (см. рисунок 1,2):

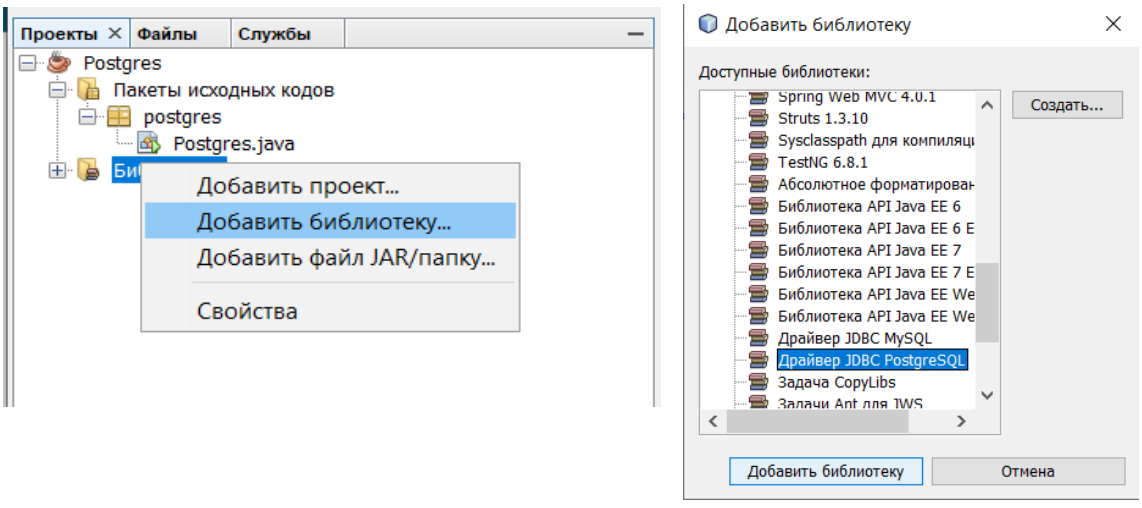


Рисунок 1,2 – Добавление драйвера JDBC

Java Database Connectivity (JDBC) — это стандартный API для взаимодействия Java-приложений с реляционными базами данных. JDBC предоставляет интерфейсы и классы для выполнения SQL-запросов, а также для обработки результатов запросов. Он является частью стандартной библиотеки Java и используется для работы с различными СУБД (Системами Управления Базами Данных) через драйверы JDBC, которые обеспечивают конкретную реализацию взаимодействия с конкретной СУБД.

Тонкости работы с JDBC:

- Драйверы JDBC: Существует четыре типа драйверов JDBC (JDBC-ODBC Bridge, Native API, Network Protocol, и Thin driver). Наиболее часто используются драйверы третьего и четвёртого типов (Network Protocol и Thin driver) из-за их производительности и переносимости.
- Управление соединениями: Одной из ключевых задач при использовании JDBC является эффективное управление соединениями с базой данных. Соединения являются дорогостоящими ресурсами, поэтому важно минимизировать их создание и закрытие. Использование пула соединений (connection pool) может значительно повысить производительность приложения.
- Обработка SQL-исключений: В JDBC все ошибки, связанные с базой данных, генерируют SQL-исключения (SQLException). Эти исключения содержат подробную информацию о проблеме, включая состояние SQL и код ошибки. Обработка и логгирование этих исключений важны для диагностики и отладки.
- PreparedStatement vs Statement: PreparedStatement используется для выполнения параметризированных SQL-запросов и является более безопасным и производительным по сравнению с Statement, поскольку он поддерживает предварительную компиляцию SQL-запросов и защиту от SQL-инъекций.
- Транзакции: JDBC поддерживает управление транзакциями с помощью методов commit() и rollback(). По умолчанию JDBC использует автокоммит, который можно отключить для явного управления транзакциями. Это позволяет сгруппировать несколько операций в одну транзакцию, что важно для обеспечения целостности данных.
- Обработка больших наборов данных: При работе с большими объемами данных необходимо учитывать память и производительность. Использование методов, таких как setFetchSize() и потоковая передача данных (streaming), может помочь эффективно обрабатывать большие наборы данных.

Плюсы JDBC:

- Стандартность и переносимость: JDBC является стандартным API, который обеспечивает переносимость Java-приложений между различными СУБД. Это позволяет разработчикам писать код, который будет работать с любой базой данных, для которой существует JDBC-драйвер.
- Гибкость: JDBC предоставляет низкоуровневый доступ к базам данных, позволяя разработчикам выполнять любые SQL-запросы и использовать возможности конкретных СУБД.
- Поддержка транзакций: JDBC поддерживает управление транзакциями, что позволяет обеспечивать атомарность и целостность операций с базой данных.
- Расширяемость: Существуют многочисленные библиотеки и фреймворки, такие как Hibernate и Spring Data, которые строятся поверх JDBC и предоставляют более высокоуровневые абстракции для работы с базами данных.

Минусы JDBC:

- Сложность и многословность: Работа с JDBC требует написания большого количества шаблонного кода для выполнения типичных задач, таких как открытие и закрытие соединений, создание и выполнение запросов, обработка исключений и управление транзакциями.
- Управление ресурсами: Неправильное управление ресурсами (соединениями, запросами, результатами) может привести к утечкам памяти и проблемам с производительностью. Разработчикам нужно быть внимательными при работе с этими ресурсами.
- Отсутствие высокоуровневых абстракций: JDBC предоставляет низкоуровневый доступ к базе данных, что требует от разработчика знания SQL и понимания внутренней работы СУБД. Это может быть сложным и требовать больше времени для разработки и отладки.
- Безопасность: Неправильное использование JDBC может привести к уязвимостям, таким как SQL-инъекции. Разработчики должны быть осторожны и использовать безопасные методы, такие как PreparedStatement, для предотвращения таких атак.

JDBC является мощным и гибким инструментом для взаимодействия с реляционными базами данных в Java-приложениях. Он предоставляет стандартный интерфейс, который поддерживает широкое разнообразие СУБД, но требует тщательного управления ресурсами и написания значительного объема шаблонного кода. Несмотря на свои недостатки, JDBC остаётся

основным выбором для низкоуровневого доступа к базам данных в Java и служит основой для многих более высокоуровневых библиотек и фреймворков.

2) JSON (JavaScript Object Notation) — это легковесный формат обмена данными, который легко читается и пишется человеком, а также легко анализируется и генерируется машинами. JSON является текстовым форматом и используется для представления структурированных данных. Его ключевые особенности включают:

- Простота и читаемость: JSON использует простой синтаксис, основанный на парах ключ-значение и массивов, что делает его легко читаемым и понимаемым.
- Языковая независимость: JSON может быть использован практически в любом языке программирования, что делает его универсальным для обмена данными между разными системами.
- Распространенность: JSON широко используется в веб-разработке для обмена данными между клиентом и сервером, в API, конфигурационных файлах и других областях.

org.json - это популярная библиотека для работы с JSON в языке программирования Java. Она предоставляет простые и удобные классы и методы для парсинга, генерации и манипулирования JSON-данными.

Плюсы библиотеки org.json:

- Простота использования: Библиотека предоставляет интуитивно понятные классы и методы для работы с JSON, что облегчает разработку.
- Легковесность: org.json — небольшая и легковесная библиотека, что делает её подходящей для большинства проектов.
- Поддержка всех основных операций: Библиотека поддерживает все необходимые операции для работы с JSON, включая парсинг, генерацию и манипуляцию данными.

Минусы библиотеки org.json

- Отсутствие расширенных возможностей: Библиотека org.json может не предоставлять некоторых расширенных функций, таких как автоматическое биндинг JSON к Java-объектам и обратно.
- Отсутствие поддержки потоковой обработки: В отличие от некоторых других библиотек (например, Jackson или Gson), org.json не поддерживает потоковую обработку JSON-данных, что может быть важно при работе с большими объемами данных.

Для работы с Json в java необходимо установить дополнительную библиотеку org.json, скачать пакет можно тут: org.json. Разместите скаченный файл в удобное место.

Переходим в Netbeans и переходим в свойства вашего проекта. Для этого можно нажать правой кнопкой мыши по проекту в окне с проектами и выбрать “Properties”. Перейти в категорию с библиотеками “Libraries” и добавить jar файл как новую библиотеку (см. рисунок 3).

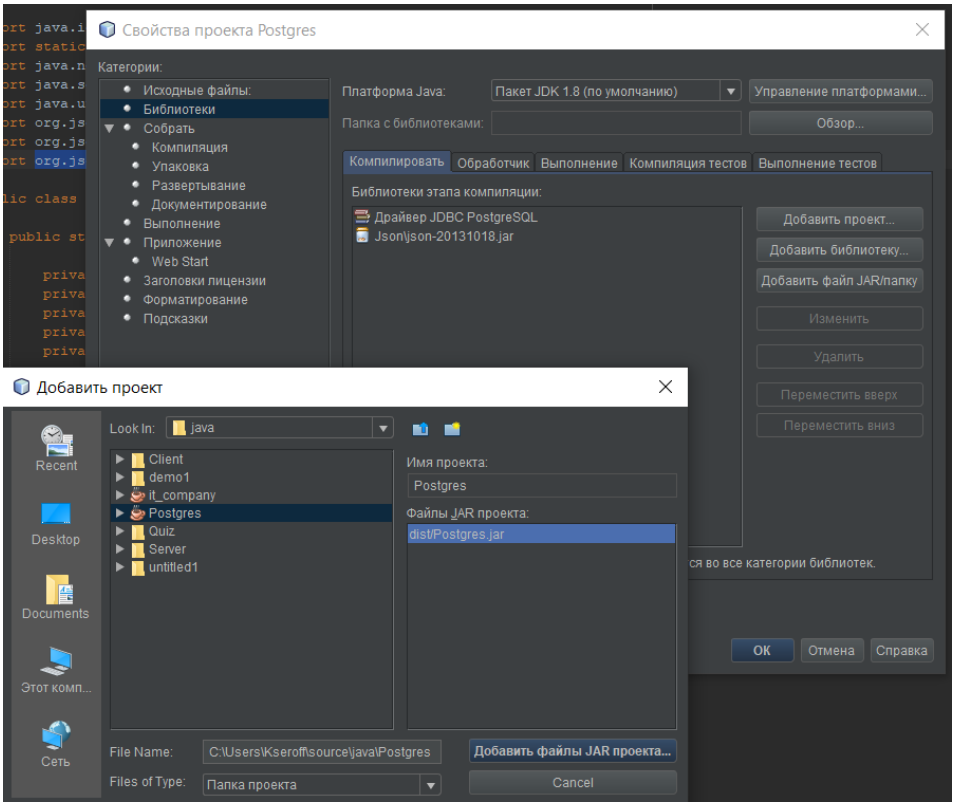


Рисунок 3 – Установка сторонних библиотек

2.1.2. Работа с базой данных

Все ответы на запросы и данные полученные и базы данных, необходимо сериализовать, а полученные новые данные десериализовывать. Для этого был выбран JSON, из-за удобства. Вот основные классы библиотеки org.json и их методы, с которыми придется работать:

JSONObject

Класс JSONObject представляет собой JSON-объект, который состоит из пар ключ-значение, напоминающие реализации Map в Java

Основные методы:

- put(String key, Object value): Добавляет или обновляет пару ключ-значение в JSON-объекте.

Пример: *jsonObject.put("name", "John Doe");*

- get(String key): Возвращает значение, связанное с заданным ключом. Значение возвращается как объект, поэтому может потребоваться приведение типов.

Пример: *String name = jsonObject.getString("name");*

- opt(String key): Возвращает значение по заданному ключу или null, если ключ отсутствует. Это позволяет избежать исключений при отсутствии ключа.

Пример: *String middleName = jsonObject.optString("middleName");*

JSONArray

Класс JSONArray представляет собой массив JSON, который используется для работы с упорядоченными коллекциями значений. Типы данных могут быть любыми от *Number*, *String*, *Bool*, *JSONArray*, и включая *JSONObject* даже в *JSONObject.NULL* объект.

Основные методы повторяются из JSONObject.

JSONTokener

Описание: Класс JSONTokener используется для токенизации JSON-строк, что полезно для обработки и разбора JSON-данных. Мы можем получить доступ к *JSONTokener* как итератор, используя *more()* метод проверки, есть ли оставшиеся элементы и *next()* чтобы получить доступ к следующему элементу

Основные методы:

- nextValue(): Возвращает следующее значение (объект или массив) в JSON-строке.

Пример: Используется внутренне классами JSONObject и JSONArray для парсинга строк.

JSONException

Описание: Исключение, которое выбрасывается при возникновении ошибок в процессе работы с JSON-данными, таких как ошибки парсинга или доступ к несуществующим ключам. Полезно для отладки и диагностики ошибок, связанных с JSON-данными.

Connection - это интерфейс, который представляет соединение с конкретной базой данных. Этот интерфейс используется для создания SQL-запросов и управления транзакциями.

Для подключения к базе данных с помощью JDBC, необходимо использовать URL-адрес базы данных, имя пользователя и пароль.

Пример подключения к базе данных psql:

String url = "jdbc:mysql://localhost:3306/mydatabase";

String username = "root";

String password = "password";

Connection connection = DriverManager.getConnection(url, username, password);

Значение этого свойства может содержать пробелы или другие специальные символы, и оно должно быть правильно закодировано, если предусмотрено в URL соединения. Пространства считаются отдельными аргументами командной строки, если только они не были удалены с помощью обратного всплеска (backslash) \ ); \\ представляет собой буквальную обратную реакцию.

Создадим объекты в нашем классе, а также в конструкторе установим соединение с базой (см. рисунок 4).

```
private Connection connection;
private String url = "jdbc:postgresql://localhost:5432/project2?characterEncoding=UTF-8";
private String user = "kseroff";
private String password = "1111";
private int limit;

private List<ClientHandler> onlineUsers;

public DataBase() throws SQLException {
    onlineUsers = new ArrayList<>();
    limit = 50;
    try {
        connection = DriverManager.getConnection(url, user, password);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Рисунок 4 – подключение к базе данных

Знакомство и использование интерфейсов JDBC:

- **DatabaseMetaData** является интерфейсом в JDBC, предоставляющим информацию о базе данных, к которой установлено соединение. Этот интерфейс позволяет получать метаданные базы данных, включая информацию о таблицах, столбцах, поддерживаемых функциях и многом другом. Запрос через DatabaseMetaData абстрагируется от конкретного SQL-диалекта, что делает код переносимым между

различными СУБД без необходимости изменять SQL-запросы для каждой из них.

Основные методы:

- getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types): Получает информацию о таблицах в базе данных.
- getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern): Получает информацию о столбцах в таблице.
- getPrimaryKeys(String catalog, String schema, String table): Получает информацию о первичных ключах таблицы.
- getImportedKeys(String catalog, String schema, String table): Получает информацию о внешних ключах, импортированных в таблицу.
- getDatabaseProductName(): Возвращает имя продукта базы данных.
- getDatabaseProductVersion(): Возвращает версию продукта базы данных.

- Интерфейс **PreparedStatement** используется для выполнения параметризованных SQL-запросов. Он предварительно компилирует SQL-запрос, что увеличивает производительность при многократном выполнении запросов с различными параметрами.

Основные методы:

- setInt(int parameterIndex, int value): Устанавливает значение типа int для указанного параметра.
- setString(int parameterIndex, String value): Устанавливает значение типа String для указанного параметра.
- setDouble(int parameterIndex, double value): Устанавливает значение типа double для указанного параметра.
- executeQuery(): Выполняет SQL-запрос, который возвращает данные (например, SELECT).
- executeUpdate(): Выполняет SQL-запрос, который изменяет данные (например, INSERT, UPDATE, DELETE).

- Интерфейс **Statement** создается через объект Connection, который представляет собой соединение с базой данных, предоставляет методы для выполнения различных видов SQL-запросов. С помощью объекта Statement вы можете выполнить запросы SELECT, INSERT, UPDATE, DELETE и другие. Использование Statement может привести к уязвимостям безопасности, таким как SQL-инъекции.

Для предотвращения этого рекомендуется использовать параметризованные запросы или PreparedStatement. Параметризованные запросы позволяют передавать параметры в SQL-запрос отдельно от кода запроса, что делает его более безопасным.

После выполнения запроса с помощью объекта Statement, вы можете получить результаты запроса с помощью объекта ResultSet. ResultSet содержит строки, соответствующие результатам запроса, и вы можете итерировать по этим строкам, чтобы получить данные из базы данных.

Основные методы:

- executeQuery(String sql): Выполняет SQL-запрос, который возвращает данные (например, SELECT).
- executeUpdate(String sql): Выполняет SQL-запрос, который изменяет данные (например, INSERT, UPDATE, DELETE).
- execute(String sql): Выполняет любой SQL-запрос.
- addBatch(String sql): Добавляет SQL-запрос в пакет для пакетного выполнения.
- executeBatch(): Выполняет пакет SQL-запросов.

- **ResultSet** - это интерфейс, представляющий результат запроса к базе данных. Он используется для перебора и чтения данных, возвращаемых из базы данных. По сути представляет собой набор строк, которые можно перебирать, чтобы получить данные из результата запроса.

ResultSet работает как курсор, указывающий на текущую строку в наборе результатов. Изначально курсор находится перед первой строкой, и нужно его переместить, чтобы начать чтение данных

При создании ResultSet через Statement или PreparedStatement, можно указать его тип. Это определяет, как курсор может перемещаться и можно ли обновлять данные:

TYPE_FORWARD_ONLY - Курсор может перемещаться только вперед.

TYPE_SCROLL_INSENSITIVE - Курсор может перемещаться в обоих направлениях, но изменения в базе данных после создания ResultSet не видны.

TYPE_SCROLL_SENSITIVE - Курсор может перемещаться в обоих направлениях, и изменения в базе данных после создания ResultSet видны.

- Основные методы:
- next(): Перемещает курсор на одну строку вперед.
- getString(String columnLabel): Получает значение столбца типа String по имени столбца.
- getInt(String columnLabel): Получает значение столбца типа int по имени столбца.
- getDouble(String columnLabel): Получает значение столбца типа double по имени столбца.
- close(): Закрывает ResultSet и освобождает все связанные с ним ресурсы.

- **ResultSetMetaData** является интерфейсом, предоставляющий метаданные о результирующем наборе данных ResultSet. Он используется для получения информации о структуре и свойствах результирующего набора данных.

Основные методы:

- getColumnCount(): Возвращает количество столбцов в ResultSet.
- getColumnName(int column): Возвращает имя указанного столбца.
- getColumnType(int column): Возвращает тип данных указанного столбца.
- getColumnDisplaySize(int column): Возвращает размер отображаемого значения указанного столбца.
- getTableName(int column): Возвращает имя таблицы, содержащей указанный столбец.

Разработка функций

Самый первый запрос, который получит сервер, будет использовать DatabaseMetaData в функции getAllTables(), для прохода по базе и получения названий всех таблиц. Нельзя исключать тот случай, что имена таблиц могут совпадать.

Чтобы получить список всех таблиц в базе данных PostgreSQL, можно выполнить следующий SQL-запрос в консоли:

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public'
AND table_type = 'BASE TABLE';
```

Запрос делает следующее: бращается к системному каталогу, содержащему информацию обо всех таблицах базы данных, ограничивает результаты до схемы public, указывает, что нас интересуют только базовые таблицы, исключая представления и другие типы таблиц.

Метод getTables интерфейса DatabaseMetaData используется для получения информации о таблицах в базе данных. Его параметры позволяют гибко настроить запрос:
ResultSet getTables(String catalog, String schemaPattern, String tableName, String[] types)

В контексте кода вызов метода выглядит так:
ResultSet tables = metaData.getTables(null, null, "%", new String[]{"TABLE"});

Этот вызов означает:
catalog = null: Не фильтрует по каталогу (база данных).
schemaPattern = null: Не фильтрует по схеме.
tableName = "%": Ищет все таблицы, независимо от имени.
types = new String[]{"TABLE"}: Интересуют только базовые таблицы, исключая представления и другие типы.

Полный код функции получения имени таблиц и сериализации ответа, можно увидеть на рисунке 5.

```
public String getAllTables() {
    JSONArray jsonArray = new JSONArray();
    try {
        DatabaseMetaData metaData = connection.getMetaData();
        ResultSet tables = metaData.getTables(null, null, "%", new String[]{"TABLE"});
        while (tables.next()) {
            String tableName = tables.getString("TABLE_NAME");
            jsonArray.put(tableName);
        }
        tables.close();
    } catch (SQLException e) {
        return e.getMessage();
    }
    return jsonArray.toString();
}
```

Рисунок 5 – функция getAllTables()

Следующая функция использует все оставшиеся интерфейсы, взаимно являясь самой ключевой функцией на сервере getDataTable(), ознакомиться можно на рисунке 6. Её задача возвращать таблицу, в зависимости от смещения по (offset) и запроса для поиска по таблице. Запрос генерируется динамически в зависимости от имени таблицы или приходящего во внутрь запроса.

Пример запросов из кода на языке psql, в консоли базы данных:

1. Общий SQL-запрос для получения данных из таблицы с лимитом и смещением:
   *SELECT * FROM tableName LIMIT ? OFFSET ?*
2. Запрос для подсчета количества строк в таблице:
   SELECT COUNT(*) AS row_count FROM tableName
3. SQL-запрос для получения данных из таблицы с условием WHERE, лимитом и смещением:
   *SELECT * FROM tableName WHERE query LIMIT ? OFFSET ?*
4. Запрос для подсчета количества строк с условием WHERE:
   SELECT COUNT(*) AS row_count FROM tableName WHERE query

```
public String getDataTable(String tableName, int offset, String query) throws SQLException {
    JSONObject tableData = new JSONObject();
    JSONArray columns = new JSONArray();
    JSONArray rows = new JSONArray();
    int rowCount = 0;

    String sqlQuery = "SELECT * FROM " + tableName;
    String countQuery = "SELECT COUNT(*) AS row_count FROM " + tableName;

    if (query != null && !query.trim().isEmpty()) {
        sqlQuery += " WHERE " + query;
        countQuery += " WHERE " + query;
    }

    sqlQuery += " LIMIT ? OFFSET ?";

    try (PreparedStatement countStmt = connection.prepareStatement(countQuery);
            PreparedStatement dataStmt = connection.prepareStatement(sqlQuery)) {

        if (query != null && !query.trim().isEmpty()) {
            try (ResultSet countResultSet = countStmt.executeQuery()) {
                if (countResultSet.next()) {
                    rowCount = countResultSet.getInt("row_count");
                }
            }
        }

        dataStmt.setInt(1, limit);
        dataStmt.setInt(2, offset);

        try (ResultSet rs = dataStmt.executeQuery()) {
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnCount = rsmd.getColumnCount();

            for (int i = 1; i <= columnCount; i++) {
                columns.put(rsmd.getColumnName(i));
            }
            tableData.put("columns", columns);

            while (rs.next()) {
                JSONObject row = new JSONObject();
                for (int i = 1; i <= columnCount; i++) {
                    row.put(rsmd.getColumnName(i), rs.getObject(i));
                }
                rows.put(row);
            }
            tableData.put("rows", rows);
        }
    } catch (SQLException e) {
        return e.getMessage();
    }

    if (query != null && !query.trim().isEmpty()) {
        tableData.put("row_count", rowCount);
    }

    return tableData.toString();
```

Рисунок 6 – функция getDataTable()

Создаются SQL-запросы для получения данных и подсчета количества строк, а также PreparedStatement для подсчета строк и получения данных. Выполняется запрос для подсчета строк, если есть фильтрация, и результат сохраняется в переменной rowCount. Добавляется в запрос LIMIT и OFFSET для получения данных с учетом параметров. Используется ResultSetMetaData для получения информации о столбцах. После извлекаются данные из ResultSet и формируются JSON-объекты для колонок и строк, возвращается результат с данными таблицы и, если нужно, с количеством строк.

***Подсчет строк в PostgreSQL:**

Остановимся на моменте подсчета строк, ведь это не совсем легкая задача в psql. Современному программисту часто приходится сталкиваться, что ему большего всего необходимо: точное количество строк или же скорость.

Известно, что подсчет строк в больших таблицах в PostgreSQL происходит медленно. Модель MVCC требует полного подсчета текущих строк для получения точного числа. Существуют обходные пути, которые значительно ускоряют это, если количество **не должно быть точным**, как это кажется в нашем случае. (Помните, что даже "точное" количество потенциально невозможно получить при одновременной загрузке на запись.)

Точное количество

Медленно для больших таблиц. При параллельных операциях записи он может устареть в тот момент, когда вы его получите.

```
SELECT count(*) AS exact_count FROM myschema.mytable;
```

Оценка

Чрезвычайно быстрый. Обычно оценка очень близка. Насколько близка, зависит от того, выполняется ли достаточно ANALYZE или VACUUM, где "достаточно" определяется уровнем активности записи в вашу таблицу.

```
SELECT reltuples AS estimate FROM pg_class where relname = 'mytable';
```

Более безопасная оценка

Вышесказанное игнорирует возможность наличия нескольких таблиц с одинаковыми именами в одной базе данных - в разных схемах. Чтобы учесть это:

```
SELECT c.reltuples::bigint AS estimate
FROM   pg_class c
JOIN   pg_namespace n ON n.oid = c.relnamespace
WHERE  c.relname = 'mytable'
AND    n.nspname = 'myschema';
```

Приведение для bigint форматирования real числа удобно, особенно при большом количестве.

Лучшая оценка

```
SELECT reltuples::bigint AS estimate
FROM   pg_class
WHERE  oid = to_regclass('myschema.mytable');
```

Быстрее, проще, безопаснее, элегантнее.

TABLESAMPLE SYSTEM (n)

*SELECT 100 * count(*) AS estimate FROM mytable TABLESAMPLE SYSTEM (1);*

Добавленное предложение для SELECT команды может быть полезно, если статистика в pg_class по какой-либо причине недостаточно актуальна. Например,

- Нет autovacuum запущено.
- Сразу после большого INSERT / UPDATE / DELETE.
- TEMPORARY таблицы (которые не охвачены autovacuum).

При этом рассматривается только случайный выбор блоков из *n* % (1 в примере) и подсчитываются строки в нем. Большая выборка увеличивает стоимость и уменьшает ошибку, выбирайте сами. Точность зависит от множества факторов:

- Распределение размера строк. Если данный блок содержит больше обычных строк, количество строк меньше обычного и т.д.
- Мертвые кортежи или а FILLFACTOR занимают место на блок. Если они неравномерно распределены по таблице, оценка может быть неправильной.
- Общие ошибки округления.

Как правило, оценка из pg_class будет более быстрой и точной.

Во-первых, нужно знать количество строк в этой таблице, если общее количество больше некоторой предопределенной константы,

Во-вторых, нужно ли это..... возможно, в тот момент, когда счетчик передаст постоянное значение, он остановит подсчет (и не будет ждать завершения подсчета, чтобы сообщить, что количество строк увеличилось).

Самый распространённый способ подсчета

Самый подходящий для запросов средних по объему таблиц. Мы можем использовать *SELECT count(*) FROM table*. Но если постоянное значение равно *500,000* и в таблице *5,000,000,000* строк, подсчет всех строк займет много времени. Можно ли прекратить подсчет, как только постоянное значение будет превышено? (Ответ: Да, если использовать подзапрос с **LIMIT**)

SELECT count(*) FROM (SELECT 1 FROM token LIMIT 500000) t;

Postgres фактически прекращает подсчет после превышения заданного предела, вы получаете точное и текущее количество до *n* строк (500000 в примере), и *n* в противном случае. Однако это далеко не так быстро, как оценка в pg_class.

Добавление в таблицу

Возвращаемся к рассмотрению функций для управления базой данных. Следующая на очереди функция addRowToTable(), добавление новой строки в таблицу. Так как у нас все происходит динамически и код может работать с абсолютно любой базой данных PSQL, возникает проблема о добавлении, т.к. мы не знаем точного типа данных поля в таблице. Стоит пояснить, что обычное добавление через запрос не позволит нам добавить данные, т.к. полученные данные из JSON строки конвертируются в string, где и возникает ошибка.

ResultSet rs = stmt.executeQuery("SELECT " + columnNames.toString() + " FROM " + tableName + " LIMIT 1");
ResultSetMetaData metaData = rs.getMetaData();
int columnType = metaData.getColumnType(i + 1);

Данный пример кода демонстрирует получения данных поля, а именно его номер. Где нам остается пройтись циклом по всем поля таблиц и установить нужное значение (см. рисунок 7)

```
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery("SELECT " + columnNames.toString() + " FROM " + tableName + " LIMIT 1");
ResultSetMetaData metaData = rs.getMetaData();

for (int i = 0; i < columns.length(); i++) {
    String columnName = columns.getString(i);
    Object value = row.get(columnName);
    int columnType = metaData.getColumnType(i + 1);

    switch (columnType) {
        case Types.BIGINT:
            preparedStatement.setLong(i + 1, row.getLong(columnName));
            break;
        case Types.INTEGER:
            preparedStatement.setInt(i + 1, row.getInt(columnName));
            break;
        case Types.BOOLEAN:
            preparedStatement.setBoolean(i + 1, row.getBoolean(columnName));
            break;
        case Types.FLOAT:
            preparedStatement.setFloat(i + 1, (float) row.getDouble(columnName));
            break;
        case Types.DOUBLE:
            preparedStatement.setDouble(i + 1, row.getDouble(columnName));
            break;
        case Types.VARCHAR:
        case Types.CHAR:
        case Types.LONGVARCHAR:
        case Types.LONGNVARCHAR:
            preparedStatement.setString(i + 1, row.getString(columnName));
            break;
        case Types.DATE:
            preparedStatement.setDate(i + 1, java.sql.Date.valueOf(row.getString(columnName)));
            break;
        case Types.TIMESTAMP:
            preparedStatement.setTimestamp(i + 1, Timestamp.valueOf(row.getString(columnName)));
            break;
        default:
            preparedStatement.setObject(i + 1, value);
            break;
    }
```

Рисунок 7 – Добавление новой строки в таблицу.

Добавление происходит в этой строчке:
"INSERT INTO " + tableName + " (" + columnNames.toString() + ") VALUES (" + columnValues.toString() + ")"

После выполняется SQL-запрос с помощью PreparedStatement методом executeUpdate() для добавления новой строки в таблицу.

Удаление из таблицы

Полная противоположность добавления, а именно доавбление происходит в функции deleteRowsFromTable() (см. рисунок 8). Одна из главных проблем с чем мы сталкиваемся, это незнаем первичного ключа в таблице, а то что мы можем от клиента, это только номер столбца. Для этого в этом коде мы используем специальный столбец ctid, который представляет собой внутренний идентификатор строки в PostgreSQL.

CTID — это скрытая и уникальная запись для каждой таблицы в PostgreSQL. Имеет тип данных системного столбца ctid и представляет собой пару (номер блока, индекс кортежа внутри блока).

Для получения ctid мы используем подзапрос для выбора строки по порядковому номеру и устанавливаем LIMIT 1, чтобы выбрать только одну строку. Затем мы удаляем выбранную строку с помощью команды DELETE.

```
public String deleteRowsFromTable(String tableName, String rowNumbersStr) {
    try {
        String[] rowNumbersArr = rowNumbersStr.replaceAll("[\\[\\]]", "").split(",\\s*");
        int[] rowNumbers = new int[rowNumbersArr.length];
        for (int i = 0; i < rowNumbersArr.length; i++) {
            rowNumbers[i] = Integer.parseInt(rowNumbersArr[i]);
        }

        Statement statement = connection.createStatement();
        for (int i = rowNumbers.length - 1; i >= 0; i--) {
            int rowNum = rowNumbers[i];
            String sql = "DELETE FROM " + tableName + " WHERE ctid = (SELECT ctid FROM " + tableName + " ORDER BY ctid OFFSET " + rowNum + " LIMIT 1)";
            statement.executeUpdate(sql);
        }

        statement.close();
        return "true";
    } catch (SQLException e) {
        return e.getMessage();
    }
}
```

Рисунок 8 – функция deleteRowsFromTable()

Обновление в таблице

Редактированием занимается функция updateRow() (см. рисунок 9) и очень схожа с удаление, тем что нам по прежнему необходимо знать первичный ключ, по которому мы можем найти нужную строчку. Снова прибегаем к методу по использованию ctid для получения индивидуального значения, только теперь в функцию дополнительно приходит JSON строка с обновленным содержимым. Используем команду UPDATE для обновления строки.

```java
public String updateRow(String tableName, String newRowJson, int EditRow) {
    try {
        JSONArray jsonArray = new JSONArray(newRowJson);
        JSONObject jsonObject = jsonArray.getJSONObject(0);
        Statement statement = connection.createStatement();

        // Формирование динамического SQL-запроса для обновления
        StringBuilder sql = new StringBuilder("UPDATE ").append(tableName).append(" SET ");
        boolean first = true;
        for (Object key : jsonObject.keySet()) {
            if (!first) {
                sql.append(", ");
            }
            sql.append(key).append(" = '").append(jsonObject.getString(key.toString())).append("'");
            first = false;
        }
        sql.append(" WHERE ctid = (SELECT ctid FROM ").append(tableName)
                .append(" ORDER BY ctid OFFSET ").append(EditRow).append(" LIMIT 1)");

        String sqlString = sql.toString();
        statement.executeUpdate(sqlString);
        return "true";
    } catch (SQLException e) {
        return e.getMessage();
    }
}
```

Рисунок 9 – функция updateRow()

Дополнительная информация для клиента

Клиенту тоже необходимо знать типы полей в таблице, для составления правильного запроса при поиске, фильтрации, получения количества строк и полей для настройки пагинации. Предполагается что клиент может задавать диапазон для числовых типов, отдельно задавать фильтрацию по слову для текстовых типов и фильтрация для булевых значений. Этим занимает функция getTableInfoAsJson() (см. рисунок 10). Механизмы запросов ничем не отличаются от выше приведенных методов и были расписаны ранее.

```java
public String getTableInfoAsJson(String tableName) {
    JSONObject tableInfo = new JSONObject();
    JSONArray jsonArray = new JSONArray();
    try {
        Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

        // Получаем количество строк в таблице
        ResultSet countResultSet = statement.executeQuery("SELECT COUNT(*) AS row_count FROM " + tableName);
        if (countResultSet.next()) {
            int rowCount = countResultSet.getInt("row_count");
            tableInfo.put("number_rows", rowCount);
        }

        // Получаем информацию о колонках
        ResultSet resultSet = statement.executeQuery("SELECT * FROM " + tableName + " LIMIT 1"); // Ограничиваем запрос од
        ResultSetMetaData metaData = resultSet.getMetaData();
        int columnCount = metaData.getColumnCount();
        tableInfo.put("number_columns", columnCount);
        for (int i = 1; i <= columnCount; i++) {
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("column_name", metaData.getColumnName(i));
            jsonObject.put("data_type", metaData.getColumnTypeName(i));
            jsonArray.put(jsonObject);
        }
        tableInfo.put("columns", jsonArray);
    } catch (SQLException | JSONException e) {
        return e.getMessage();
    }
    return tableInfo.toString();
}
```

Рисунок 10 – функция getTableInfoAsJson()

2.1.3 Разработка протокола и работа с клиентами

Протокол — это соглашение, которое используется клиентом и сервером. С помощью протокола клиент и сервер согласованно обмениваются данными.

Взаимодействие с клиентами осуществляется посредством простого протокола передачи данных через сокетов. Протокол обмена данными между сервером и клиентом основан на передаче текстовых сообщений. Клиенты отправляют текстовые запросы на сервер, и сервер отвечает соответствующими текстовыми сообщениями.

Класс ClientHandler реализует обработку запросов клиентов и взаимодействие с базой данных. Он обрабатывает входящие сообщения от клиентов, выполняет соответствующие действия в базе данных и отправляет ответы обратно клиентам. Также он управляет списком подключенных клиентов и уведомляет их об изменениях в данных:

- Конструктор принимает сокет клиента и объект базы данных. Он добавляет текущего пользователя в список онлайн пользователей базы данных и в список клиентов clients.
- Метод run - основной метод, запускаемый в отдельном потоке, чтобы обрабатывать запросы клиентов. Метод считывает запросы от клиента и взаимодействуют с сервером посредством JSON-сообщений. Каждое сообщение содержит ключ action, указывающий на тип выполняемого действия, и дополнительные параметры, необходимые для этого действия.
- Некоторые запросы сопровождаются оповещением всех остальных пользователей, а именно для действий, изменяющих данные в таблице (добавление, удаление, обновление строк), после успешного выполнения операции вызывается метод notifyClients, уведомляющий всех подключенных клиентов об изменениях.

Протокол обмена данными между клиентом и сервером включает следующие действия:

1) GetTableNames: Возвращает имена всех таблиц.
2) GetTypeTable: Возвращает информацию о структуре таблицы.
3) GetDataTable: Возвращает данные из таблицы с учетом смещения и опционального запроса.
4) AddNewRow: Добавляет новую строку в таблицу и уведомляет всех клиентов об обновлении таблицы.
5) DeleteRow: Удаляет строки из таблицы и уведомляет всех клиентов об обновлении таблицы.

6) UpdateRow: Обновляет строки в таблице и уведомляет всех клиентов об обновлении таблицы.
7) disconnect: Завершает соединение с клиентом.

2.2 Разработка Клиента

2.1.1 Подключение сторонних библиотек

1) Клиентская сторона должна тоже подключить библиотеку для работы с JSON, а именно org.json. Её подключение было описано ранее, на стороне сервера.

2) Клиент подразумевает работу с датой, временем и форматами. Для удобства работы подключим стороннюю библиотеку JCalendar 1.1.4, которая приносит с собой весь необходимый функционал, а также несколько полей(Field) для графической составляющей программы.

После скачивания файла, его необходимо добавить в Libraries, как ранее делали org.json, а также добавить новые поля в палитру элементов управления Swing (см. рисунок 11).

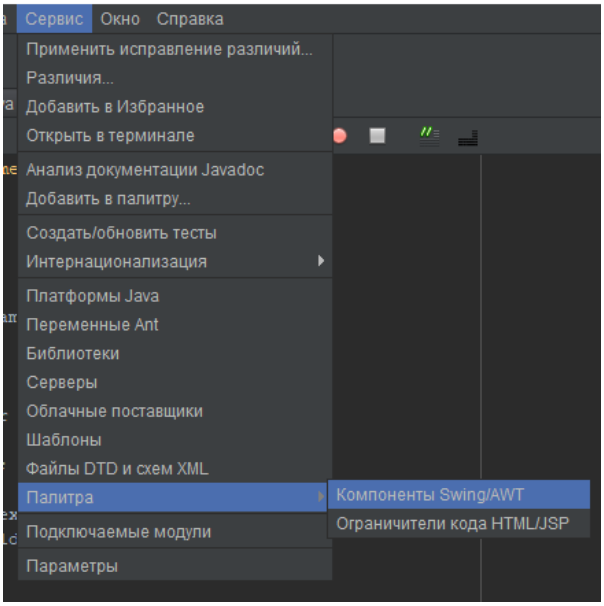


Рисунок 11 – добавления новых элементов

Следуем дальнейшим шагам:

1. Добавить из архива JAR
2. Выбираем путь к файлу
3. Выбираем компоненты (необходимо всего два)

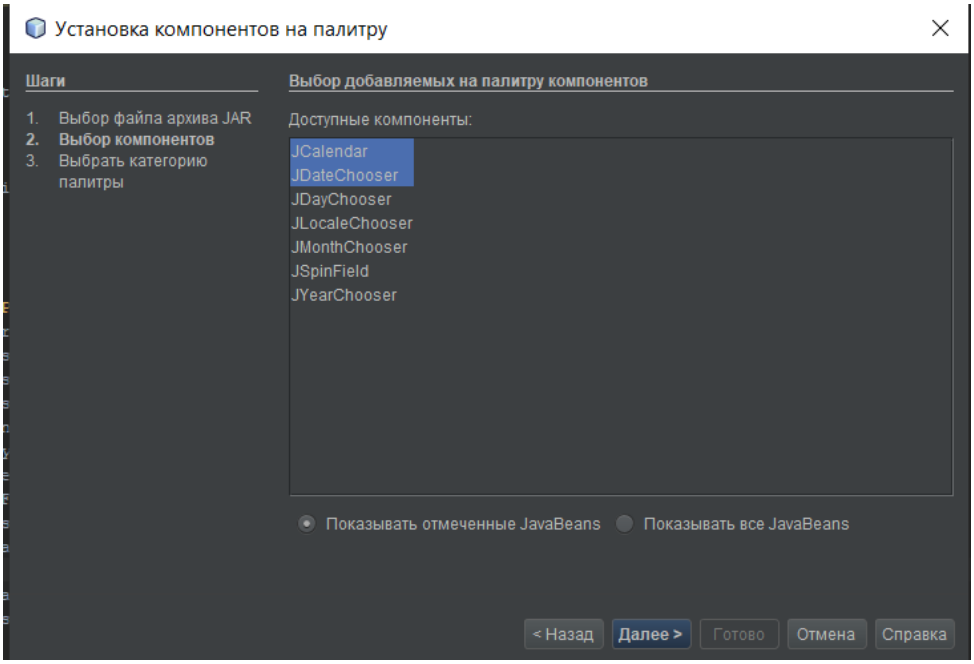


Рисунок 12 - выбираем элементы управления Swing

2.2.2 Разработка интерфейса

Добавим новый проект, для этого нужно создать форму. Правой кнопкой мыши по проекту в области "Projects" и выберите "New" -> "JFrame Form". Выберите "Swing GUI Forms" в качестве типа форм (см. на рисунке 13,14):

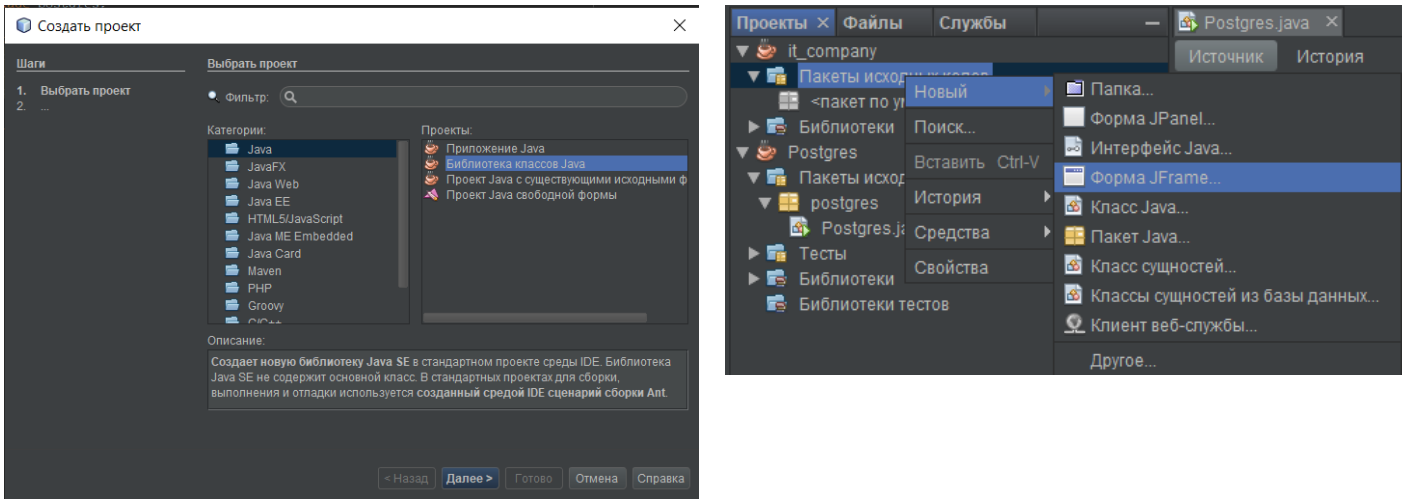


Рисунок 13,14 – Создание проекта

После создания проекта открывается пустое поле с инструментами. Необходимо будет для начала добавить “JTabbedPane”(Вкладки панели) и добавить внутрь то количество “JPanel”(Панель) сколько у нас всего таблиц в базе данных. В каждую панель в дальнейшем необходимо будет добавить “JTable”(Таблица) с “JScrollPane”, которая будет отвечать за отображение

содержимого.

Необходимо создать кнопки удаления, добавления, кнопку вызова окна для фильтрации, а также поле поиска. Интерфейсно результат должен быть схож с рисунком 15, а также минимально соответствовать дереву компонентов на рисунке 16,17.

Рисунок 15 – Интерфейс клиентской части

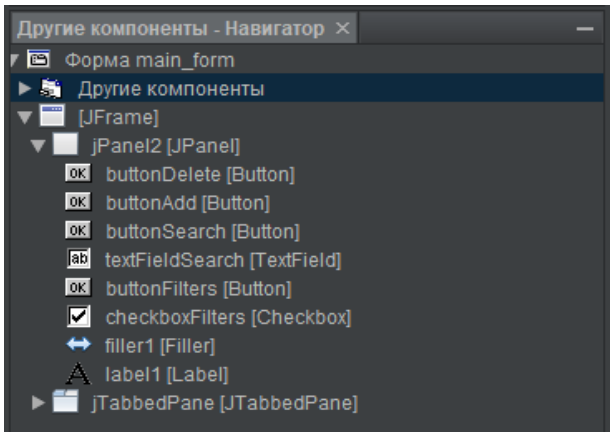


Рисунок 16 – дерево компонентов

```
// Variables declaration - do not modify
private java.awt.Button buttonAdd;
private java.awt.Button buttonDelete;
private java.awt.Button buttonFilters;
private java.awt.Button buttonSearch;
private java.awt.Checkbox checkboxFilters;
private javax.swing.JPanel jPanel2;
private static javax.swing.JTabbedPane jTabbedPane;
private java.awt.TextField textFieldSearch;
// End of variables declaration
```

Рисунок 17 - модификаторы доступа

В конструкторе класса main_form:

- устанавливается максимальное количество строк в таблице (limit)
- запрещаем изменение размеров окна с помощью команды setResizable(false)
- инициализируются компоненты GUI из разработаного интерфейса выше с помощью initComponents()
- Добавляются слушатели событий (см. рисунок 18) к различным кнопкам и элементам GUI. Каждый слушатель представляет собой объект, реализующий определенный интерфейс (например, ActionListener), который обрабатывает события, возникающие при взаимодействии пользователя с GUI.

- Блок кода (см. рисунок 19) добавляет слушателя окна, который обрабатывает событие закрытия окна. В данном случае, при закрытии окна вызывается метод dispose(), который освобождает все ресурсы, связанные с окном.

```
buttonFilters.addActionListener(new OpenFilterButton());
buttonAdd.addActionListener(new OpenAddEditButton());
checkboxFilters.addItemListener(new CheckboxFilter());
buttonSearch.addActionListener(new ButtonSearch());
buttonDelete.addActionListener(new DeleteRowTableButton());
jTabbedPane.addChangeListener(new PanelChanged());
```

Рисунок 18 – Установка слушателей

```
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        client.close();
        dispose();
    }
});
```

Рисунок 19 – Закрытие формы

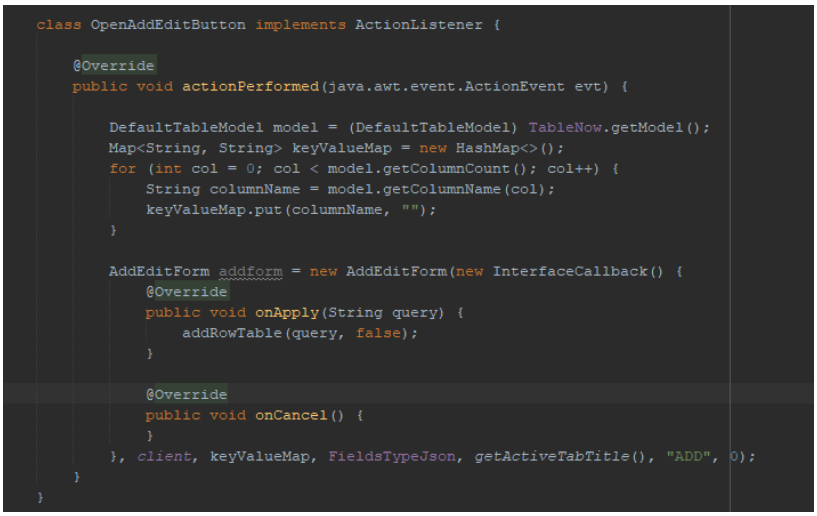
Вызывается метод createDynamicPanels (см. рисунок 20), который создает динамические панели для отображения таблиц. В этом методе используется JSON-строка, содержащая имена таблиц, которая передается в качестве аргумента. С помощью этой строки генерируются и создаются табы в JTabbedPane. Для каждой сгенерированной новой панели, создаем внутри таблицу со скроллом, запрещаем редактировать ячейки в таблице и добавляем сортировку на поля таблицы.

```
// Удаляем все существующие панели(если они есть неожиданно)
jTabbedPane.removeAll();

JSONArray jsonArray = new JSONArray(jsonString);

for (int i = 0; i < jsonArray.length(); i++) {
    String tableName = jsonArray.getString(i);
    JPanel panel = new JPanel(new BorderLayout()); // создание панели
    JTable table = new JTable(); // создание таблицы
    table.setAutoCreateRowSorter(true); // установка сортировки
    table.setModel(new DefaultTableModel() {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false; // Запрет редактирования ячеек
        }
    });

    JScrollPane scrollPane = new JScrollPane(table); // установка скролла

    panel.add(scrollPane, BorderLayout.CENTER);
    jTabbedPane.addTab(tableName, panel); // новый таб
}
```

Рисунок 20 – метод createDynamicPanels()

2.2.3 Разработка Формы добавления и редактирования

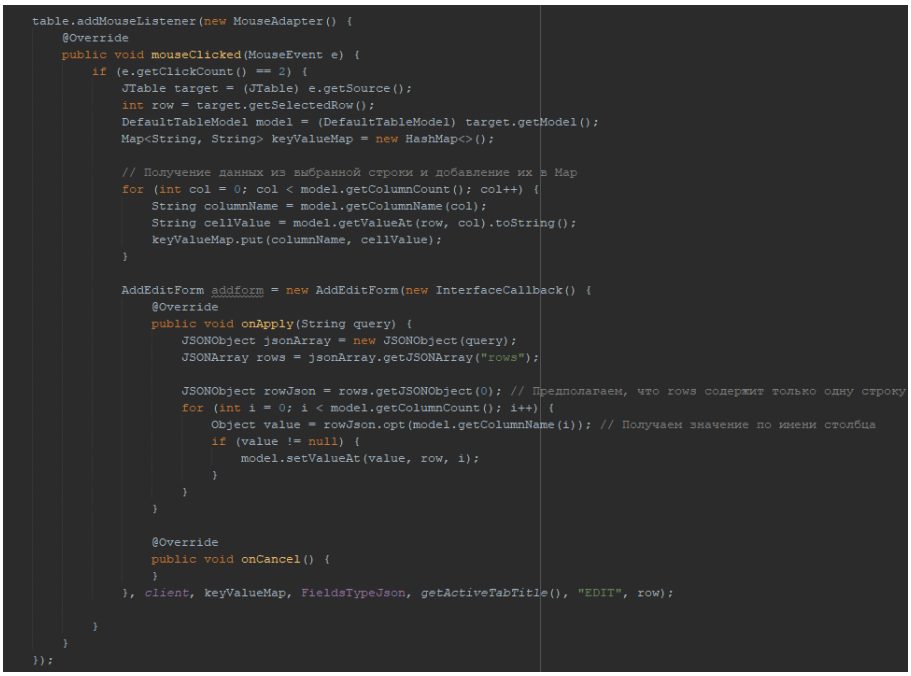
Для начала нужно разработать класс, на который мы установили прослушку при нажатии на кнопку “Добавить” (см. рисунок 21).



```
class OpenAddEditButton implements ActionListener {

    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {

        DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
        Map<String, String> keyValueMap = new HashMap<>();
        for (int col = 0; col < model.getColumnCount(); col++) {
            String columnName = model.getColumnName(col);
            keyValueMap.put(columnName, "");
        }

        AddEditForm addform = new AddEditForm(new InterfaceCallback() {
            @Override
            public void onApply(String query) {
                addRowTable(query, false);
            }

            @Override
            public void onCancel() {
            }
        }, client, keyValueMap, FieldsTypeJson, getActiveTabTitle(), "ADD", 0);
    }
}
```

Рисунок 21 – класс OpenAddEditButton

Форма оборудована функционалом не только формирования форм, но также и заполнения их, при поступившей в конструктор строки с информацией о полях. Открытие формы и автоматическое заполнение ее данными ожидает при двойном клике на строчку из таблицы, для этого используем MouseEvent из java.awt.event. При помощи e.getClickCount() можем получить число нажатий и отследить двойной клик. Добавим новый функционал в функцию createDynamicPanels() (см. рисунок 22).



```
table.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 2) {
            JTable target = (JTable) e.getSource();
            int row = target.getSelectedRow();
            DefaultTableModel model = (DefaultTableModel) target.getModel();
            Map<String, String> keyValueMap = new HashMap<>();

            // Получение данных из выбранной строки и добавление их в Map
            for (int col = 0; col < model.getColumnCount(); col++) {
                String columnName = model.getColumnName(col);
                String cellValue = model.getValueAt(row, col).toString();
                keyValueMap.put(columnName, cellValue);
            }

            AddEditForm addform = new AddEditForm(new InterfaceCallback() {
                @Override
                public void onApply(String query) {
                    JSONObject jsonArray = new JSONObject(query);
                    JSONArray rows = jsonArray.getJSONArray("rows");

                    JSONObject rowJson = rows.getJSONObject(0); // Предполагаем, что rows содержит только одну строку
                    for (int i = 0; i < model.getColumnCount(); i++) {
                        Object value = rowJson.opt(model.getColumnName(i)); // Получаем значение по имени столбца
                        if (value != null) {
                            model.setValueAt(value, row, i);
                        }
                    }
                }

                @Override
                public void onCancel() {
                }
            }, client, keyValueMap, FieldsTypeJson, getActiveTabTitle(), "EDIT", row);

        }
    }
});
```

Рисунок 22 – Обработка двойного клика мыши

Основные шаги:

1) Получение данных о строке и модели:
2) Сбор данных строки: проход по всем столбцам выбранной строки и добавление данных в keyValueMap.

Одной из главных особенностей разработки формы, является использование интерфейса InterfaceCallback (см. рисунок 23). Этот интерфейс и его использование представляют собой шаблон обратного вызова (callback) для взаимодействия между различными частями вашего приложения. Давайте разберем его подробнее. Интерфейсы в Java позволяют определить методы, которые должны быть реализованы классами, которые этот интерфейс реализуют. В данном случае, интерфейс InterfaceCallback используется для обратного вызова, чтобы передать результат обратно вызывающему коду. Класс OpenAddEditButton называется: “Анонимный” класс.

Метод onApply:

```
@Override
public void onApply(String query) {
    addRowTable(query, false);
}
```

Этот метод вызывается, когда форма подтверждается (например, пользователь нажимает кнопку "Применить" или "Сохранить"). Метод onApply принимает строку query в качестве параметра и вызывает метод addRowTable, передавая ему query и false.

Метод onCancel:

```
@Override
public void onCancel() {
}
```

Этот метод вызывается, когда форма закрывается без сохранения изменений (например, пользователь нажимает кнопку "Отмена".

```
/**
 * @author Kseroff
 */
public interface InterfaceCallback {
    void onApply(String query);
    void onCancel();
}
```

Рисунок 23 - интерфейс InterfaceCallback

Переходим к разработке Формы AddEditForm.java:

Класс AddEditForm динамически создает интерфейс на основе предоставленных данных, позволяет пользователю вводить или изменять

данные, а затем отправляет их обратно через интерфейс обратного вызова AddEditForm. Конструктор в себя принимает значения и отправляет их в функцию для создания формы:

1. callback: Интерфейс обратного вызова для обработки результата (сохранение или отмена).
2. client: Экземпляр клиента для отправки запросов.
3. keyValueMap: Карта ключ-значение, содержащая данные строки, которые будут редактироваться.
4. FieldsTypeJson: JSON строка, описывающая структуру полей (название столбца и тип данных).
5. tableName: Имя таблицы для обновления данных.
6. action: Действие, которое нужно выполнить ("ADD" или "EDIT").
7. editRow: Индекс строки для редактирования.

Далее генерируется форма, создается панель rightPanel с GridBagLayout для размещения компонентов. Также создается Map для хранения соответствий меток и компонентов ввода. Настраиваются ограничения компоновки для размещения компонентов. Цикл проходит по всем столбцам JSON-массива и создает соответствующие компоненты ввода данных для каждого столбца.

В зависимости от типа данных столбца выбирается соответствующий компонент ввода:

- Для столбцов типа "date" или "timestamp" используется компонент JDateChooser.
- Для булевых значений создается панель с радиокнопками "Yes" и "No". Для текстовых создается одно текстовое поле.
- Для числовых тоже создается текстовое поле, но с фильтром ввода запрещающий вводить символы, для этого используется KeyEvent (см. рисунок 24).

Поля создаются пустыми или заполненными, в зависимости от keyValueMap, который настраивается при редактировании.

```
textField.addKeyListener(new KeyAdapter() {
    @Override
    public void keyTyped(KeyEvent e) {
        char c = e.getKeyChar();
        if (!Character.isDigit(c) && c != KeyEvent.VK_BACK_SPACE && c != KeyEvent.VK_DELETE) {
            e.consume();
        }
    }
});
```

Рисунок 24 – Фильтрация ввода

Создается кнопка "Сохранить" и добавляется к ней обработчик событий. При нажатии кнопки происходит сбор данных из полей формы, их преобразование в формат JSON и отправка на сервер для добавления или обновления записи. Если операция успешна, вызывается метод callback.onApply и окно закрывается. В случае ошибки показывается сообщение об ошибке. В зависимости от action будут отрабатывать разные запросы. Если ошибок при Добавлении/Изменении не возникли, то изменения применятся, и форма удачно закроется, в противном случае появится диалоговое окно с ошибкой.

Создается кнопка "Отмена" и добавляется к ней обработчик событий. При нажатии кнопки вызывается метод callback.onCancel и окно закрывается без сохранения изменений.

За добавление строк в таблицу отвечает функция addRowTable() (см. рисунок 25). Так как **при сериализации строки в JSON для добавления/обновления, поля и значения могут терять очередность** в зависимотси от полей таблицы, находясь в хаотичных местах. Этот недостаток несет с собой усложение в реализации добавления. Прежде чем добавлять значения, необходимо отсортировать их расположение в соответствие с полями таблицы.

```
public void addRowTable(String infoJson, boolean addToStart) {
    JSONObject jsonObject = new JSONObject(infoJson);
    JSONArray columns = jsonObject.getJSONArray("columns");
    JSONArray rows = jsonObject.getJSONArray("rows");

    DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
    Map<String, Integer> columnIndices = new HashMap<>();

    // Получаем индексы столбцов по их названиям
    for (int columnIndex = 0; columnIndex < columns.length(); columnIndex++) {
        String columnName = columns.getString(columnIndex);
        columnIndices.put(columnName, columnIndex);
    }

    for (int i = 0; i < rows.length(); i++) {
        JSONObject rowJsonObject = rows.getJSONObject(i);
        Object[] rowData = new Object[columns.length()];
        for (int j = 0; j < columns.length(); j++) {
            String columnName = columns.getString(j);
            int columnIndex = columnIndices.get(columnName);
            rowData[columnIndex] = rowJsonObject.get(columnName);
        }

        if (addToStart) {
            model.insertRow(i, rowData);
        } else {
            model.addRow(rowData);
        }
    }
}
```

Рисунок 25 – функция addRowTable()

Рассмотрим заодно реализацию обработчика, отвечающего за удаление, а именно класс DeleteRowTableButton (см. рисунок 26). В создании используя поток (stream), далее преобразуем каждый индекс выделенной строки, добавляя к нему значение deleteRow. Затем преобразуем полученные значения в строки и собираем их в массив строк, отправляем запрос на удаление строк через метод deleteRow клиента. Если результат запроса равен "true", это означает, что удаление на сервере прошло успешно. Если удаление на сервере прошло успешно, то в цикле удаляем строки из модели таблицы. **Идем в обратном порядке** (от конца к началу), чтобы избежать сдвигов индексов при удалении строк.

```
class DeleteRowTableButton implements ActionListener {

    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        int[] selected = TableNow.getSelectedRows();
        String[] a = Arrays.stream(selected)
                .map(i -> i + deleteRow)
                .mapToObj(String::valueOf)
                .toArray(String[]::new);
        String QuerySelected = Arrays.toString(a);

        if ("true".equals(client.deleteRow(getActiveTabTitle(), QuerySelected))) {

            DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
            for (int i = selected.length - 1; i >= 0; i--) {
                model.removeRow(selected[i]);
            }
        }
    }
}
```

Рисунок 26 - класс DeleteRowTableButton

2.2.4 Разработка Формы фильтрации и поиск

Создаем обработчики на кнопку фильтрации, а также чекбокс, который будет регулировать состояние запроса на фильтрацию в таблице. Обработчики внутри ничем не отличаются и только вызывают метод SearchQuery(), где будет происходить правильность формирования запроса поиска в таблице.

Метод SearchQuery() предназначен для выполнения поискового запроса к базе данных на основе введенного пользователем текста. Вот описание его деталей реализации:

1) Получение текста поискового запроса из текстового поля textFieldSearch.
2) Если текст поиска не пустой:
   - Получение модели таблицы для доступа к информации о столбцах.
   - Инициализация Map, содержащего информацию о типах данных столбцов.

- Проверка, является ли введенный текст числом, делается для возможности применения введённого числа к полю с числовым типом. Поля типа bool в свою очередь игнорируются полностью.
- Перебор каждого столбца.
- Формирование условий поиска в зависимости от типа данных столбца (см. рисунок 27). Связанно это с тем, что поиск с применением LIKE возможен только для текстовых типов, поэтому используется CAST для числовых и TO_CHAR с парнингов формата для даты, чтобы явно могли сконвертировать эти поля в строчку и применить поиск по всем столбцам таблицы.

3) Добавление условий фильтрации, если выбрана опция фильтров, результат берется из результата обработки формы FilterForm.
4) Получение окончательного SQL-запроса из StringBuilder.
5) Применнение результата к таблице.

```
for (int col = 0; col < model.getColumnCount(); col++) {
    String columnName = model.getColumnName(col);
    String dataType = (String) columnInfo.get(columnName);
    if (dataType.equals("bool")) {
        continue;
    } else if (dataType.contains("int") || dataType.contains("serial")
            || dataType.contains("decimal") || dataType.contains("numeric")) {
        if (isNumeric) {
            if (queryBuilder.length() > 1) {
                queryBuilder.append(" OR ");
            }
            queryBuilder.append("CAST(").append(columnName).append(" AS TEXT) LIKE '%").append(searchText).append("%'");
        }
    } else if (dataType.equals("date") || dataType.equals("timestamp")) {
        if (isNumeric) {
            if (queryBuilder.length() > 1) {
                queryBuilder.append(" OR ");
            }
            queryBuilder.append("TO_CHAR(").append(columnName).append(", 'YYYY-MM-DD') LIKE '%").append(searchText).append("%'");
        }
    } else {
        if (queryBuilder.length() > 1) {
            queryBuilder.append(" OR ");
        }
        queryBuilder.append(columnName).append(" LIKE '%").append(searchText).append("%'");
    }
}
```

Рисунок 27 – создание запроса для поиска

Подробнее рассмотрим создание формы для фильтрации. Устанавливаем обработчик для просушки на кнопку (см. рисунок 27), также применяем интерфейс InterfaceCallback для получения результата. Ожидается что:

1. Форма должна уметь правильно заполняться данными при повторном ее открытие, этим занимается функция createFilterFieldsFromConditions().
2. Форма также выдавать результат в виде сериализованной строчки, этим занимается buildQuery().

```
class CheckboxFilter implements ItemListener {

    @Override
    public void itemStateChanged(java.awt.event.ItemEvent evt) {
        SearchQuery(getActiveTabTitle());
    }
}

class ButtonSearch implements ActionListener {

    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SearchQuery(getActiveTabTitle());
    }
}
```

Рисунок 28 – Просушка Фильтрации

Параметры конструктора:

- callback: объект InterfaceCallback, используемый для обратных вызовов при применении или отмене фильтров.
- json: строка в формате JSON, описывающая колонки таблицы и их типы данных.
- FillFilterFields: строка с условиями фильтрации, которые должны быть предустановлены в форме.

Конструктор создает и настраивает окно, считывает JSON для получения структуры колонок и добавляет элементы управления для каждой колонки. Генерация заключается в том, что каждый тип имеет свою разную реализацию, тем самым поиск. Числовые типы и Дата умеют устанавливать значения в диапазоне, иначе применяя ключевое слово BETWEEN, например, 2 BETWEEN 16 или 2024-05-01 BETWEEN 2024-07-30. Поля для определенного поля в таблице могут генерироваться бесконечно, т.к. с ними создается специальная кнопка для создания, тем самым позволяя пользователя задать несколько диапазонов или одиночных значений. текстовые поля в свою очередь могут принимать в себя только один текст, булевые только свое значение, числовые ограниченны на ввод только чисел, а дата должна обязательно совпадать с ее внутренним форматом.

Таким образом, поля, хранящиеся в отдельных панелях, можно легко сортировать, применять и доставать из них информацию, чем и занимается функция buildQuery() (см. рисунок 29). Каждое Панель тесно связана с чекбоксом, который регулирует необходимость применения содержимого поля или полей к запросу на поиск. При нажатии на кнопку сохранить мы проходимся по всем панелям, у которых включен флаг и считываем информацию из принадлежащим этой панели текстовым полям.

```
for (int i = 0; i < filterPanels.size(); i++) {
    JPanel filterPanel = filterPanels.get(i);
    JPanel checkBoxPanel = (JPanel) filterPanel.getParent().getComponent(0);
    JCheckBox checkBox = (JCheckBox) checkBoxPanel.getComponent(0);
    if (checkBox.isSelected()) {
        Component[] components = filterPanel.getComponents();
        StringBuilder condition = new StringBuilder();
        boolean isFirstFieldInPanel = true;
        for (Component component : components) {
            if (component instanceof JPanel) {
                JPanel fieldPanel = (JPanel) component;
                Component[] fields = fieldPanel.getComponents();
                String minValue = null;
                String maxValue = null;
                boolean isRange = false;
                int textFieldCount = 0;
                for (Component field : fields) {
                    if (field instanceof JTextField) {
                        textFieldCount++;
                    }
                }
                for (Component field : fields) {
```

Рисунок 29 – функция buildQuery()

Отметить стоит сложность заполнения полей фильтрации, при повторном открытии, а именно парсинг входящей строки. На вход функция createFilterFieldsFromConditions() принимает сериализованую строку и должна правильно ее пропарсить для удобного прохождения по циклу, и получению конечного результата (см. рисунок 30)

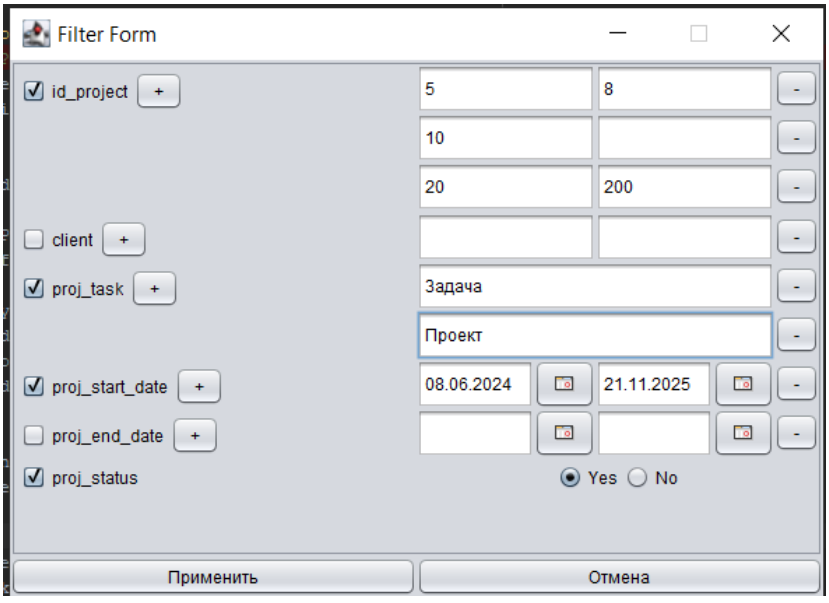


Рисунок 30 – Фильтрация

Из рисунка 29 понимаем какая на вход должна быть JSON строка, а именно: *(id_project BETWEEN '5' AND '8' OR id_project = '10' OR id_project BETWEEN '20' AND '200') AND (proj_task = 'Задача' OR proj_task = 'Проект') AND (proj_start_date BETWEEN '2024-06-08' AND '2025-11-21') AND (proj_status = 'Yes').* Разберем действия отдельно:

Разберем детально моменты получения динамического списка из строки, (см. рисунок 31):

1) Первое регулярное выражение \\((.*?)\\) находит все подстроки, заключенные в круглые скобки. Таким образом, оно будет искать группы условий, например, (id_project BETWEEN '5' AND '8' OR id_project = '10' OR id_project BETWEEN '20' AND '200').

2) Используем Парсинг каждой группы условий: fieldPattern равен условию регулярного выражения: ("^\\s*([^\\s=]+)\\s*") для поиска поля до оператора (например, id_project в id_project BETWEEN '5' AND '8'). После fieldPattern.matcher(conditionGroup): создает объект Matcher для поиска поля в условии

3) Разделяем условия по оператору OR: conditionGroup.split("\\s*(?i)OR\\s*") разбивает группу условий по ключевому слову, игнорируя регистр (благодаря (?i)), cleanedConditions.add(c.trim()) добавляет каждое условие в список cleanedConditions, убирая лишние пробелы.

Ожидаемый результат:

Condition Group:
id_project BETWEEN '5' AND '8'
id_project = '10'
id_project BETWEEN '20' AND '200'
Condition Group:
proj_task = 'Задача'
proj_task = 'Проект'
Condition Group:
proj_start_date BETWEEN '2024-06-08' AND '2025-11-21'
Condition Group:
proj_status = 'Yes'

```
public void createFilterFieldsFromConditions(String conditions) {
 String conditionRegex = "\\((.*?)\\)";
 Pattern conditionPattern = Pattern.compile(conditionRegex);
 Matcher conditionMatcher = conditionPattern.matcher(conditions);
 int i = 0;

    while (conditionMatcher.find()) {
        String conditionGroup = conditionMatcher.group(1); //группа
        Pattern fieldPattern = Pattern.compile("^\\s*([^\\s=]+)\\s*");
        Matcher fieldMatcher = fieldPattern.matcher(conditionGroup);

        String[] conditionsArray = conditionGroup.split("\\s*(?i)OR\\s*");
        List<String> cleanedConditions = new ArrayList<>();
        for (String c : conditionsArray) {
            cleanedConditions.add(c.trim()); // условия
        }

            if (fieldMatcher.find()){
                String fieldName = fieldMatcher.group(1); //название

                for (int j = i; j < filterPanels.size(); j++) {
                    JPanel filterPanel = filterPanels.get(j);
                    JPanel checkBoxPanel = (JPanel) filterPanel.getParent().getComponent(0);
                    // включение чекбокса
                    JCheckBox checkBox = (JCheckBox) checkBoxPanel.getComponent(0);
                    if(checkBox.getText().equals(fieldName)){
                        checkBox.setSelected(true);
```

Рисунок 31 – парсинг строчки

2.2.5 Разработка пагинации

Пагинация (от англ. "pagination") — это процесс разделения большого набора данных на более мелкие части (страницы), что позволяет загружать и отображать данные постепенно. Это особенно важно при работе с базами данных и веб-приложениями, где отображение большого количества данных сразу может быть неэффективным и неудобным.

Плюсы и преимущества использования пагинации:

1) Производительность: Пагинация уменьшает объем данных, загружаемых и отображаемых за один раз, что улучшает производительность и снижает нагрузку на сервер и клиент.
2) Удобство пользователя: Пользователи могут легче просматривать и находить нужные данные, так как данные представлены в более управляемых частях.
3) Снижение использования памяти: Поскольку только часть данных загружается и отображается, это снижает использование оперативной памяти.
4) Быстрая загрузка страниц: Страницы загружаются быстрее, так как передается меньше данных.

Когда и как должна применяться пагинация:

Пагинация должна применяться всякий раз, когда объем данных слишком велик, чтобы отображаться сразу. Это может быть при работе с таблицами в базах данных, отображении списков элементов на веб-страницах и в мобильных приложениях.

Реализация пагинации на стороне клиента:

Пагинация реализуется путем добавления слушателя AdjustmentListener для вертикального скроллбара “JScrollPane” внутри функции createDynamicPanels(). В фрагменте кода (см. рисунок 32) реализуется динамическая пагинация с использованием вертикального скроллбара. Когда пользователь скроллит таблицу вниз или вверх, загружаются соответствующие данные.

```
int extent = verticalScrollBar.getModel().getExtent();
int maximum = verticalScrollBar.getMaximum();
int value = verticalScrollBar.getValue();

if ((value + extent == maximum) && numberColumns > offset + limit) {
    wasAtTop = false;
    offset += limit;

    String Jsonresult = client.getDataTable(tableName, offset, SearchValue);
    JSONObject jsonResult = new JSONObject(Jsonresult);
    if (jsonResult.has("row_count")) {
        this.numberColumns = jsonResult.getInt("row_count");
    }
    addRowTable(Jsonresult, false);

    if ((offset - limit * 2) > -1) {
        deleteRow += limit;
        int rowHeight = table.getRowHeight();
        int rowsToDelete = Math.min(limit, model.getRowCount());
        for (int j = 0; j < limit; j++) {
            model.removeRow(0);
        }
        int adjustment = rowsToDelete * rowHeight;
        int newScrollValue = Math.max(value - adjustment, 0);
        SwingUtilities.invokeLater(() -> verticalScrollBar.setValue(newScrollValue));
    }

} else if (value == 0 && deleteRow > 0) {
    deleteRow -= limit;
    if(!wasAtTop)
        offset -= limit;
    offset -= limit;

    wasAtTop = true;
    int rowHeight = table.getRowHeight();

    String Jsonresult = client.getDataTable(tableName, offset, SearchValue);
    JSONObject jsonResult = new JSONObject(Jsonresult);
    if (jsonResult.has("row_count")) {
        this.numberColumns = jsonResult.getInt("row_count");
    }
    addRowTable(Jsonresult, true);

    int adjustment = limit * rowHeight;
    int newScrollValue = value + adjustment;

    SwingUtilities.invokeLater(() -> verticalScrollBar.setValue(newScrollValue));

    for (int j = 1; j <= limit; j++) {
        model.removeRow(model.getRowCount() - 1);
    }
```

Рисунок 32 – Реализация Пагинации

Как это работает:

1) Инициализация лимита и оффсета: Устанавливаются начальные значения лимита (limit = 50) и оффсета (offset = 0). limit задает количество строк, которые будут загружаться за один раз, а offset смещение.

2) Прослушивание событий скролла: Когда пользователь скроллит таблицу, прослушивать событий скроллбара проверяет текущую позицию скролла. С помощью полученных переменных:

- value: Это текущее положение скроллбара. Он показывает, насколько далеко пользователь прокрутил содержимое. Значение value изменяется при перемещении ползунка скроллбара.
- extent: Это видимая часть содержимого. Он показывает размер видимой области в пикселях. Например, если у вас есть панель высотой 1000 пикселей, но окно просмотра показывает только 200 пикселей, extent будет равен 200

3) Загрузка новых данных при скролле вниз: Когда пользователь доходит до конца таблицы выполняется проверка на “value + extent == maximum” (Проверяется, находится ли скроллбар внизу) и “numberColumns > offset + limit” (Убедиться, что еще есть данные для загрузки). После код увеличивает offset, чтобы загрузить следующую порцию данных, и применяет к таблице результат запроса.

Стоит отметить что больше 100 строк в таблице находиться не может, поэтому при скролле вниз возможен вариант удаления первых строк таблиц, соответствуя величине limit. После удаления необходимо востановить скролл к месту загрузки новых данных с помощью:

S*wingUtilities.invokeLater(() -> verticalScrollBar.setValue(newScrollValue))* чтобы пользователь не почувствовал изменений. Сделано это во благо оптимизации и разгрузки клиентской стороны.

4) Загрузка предыдущих данных при скролле вверх: когда пользователь скроллит вверх до начала таблицы (value == 0), код уменьшает offset и загружает предыдущую порцию данных. Следуя по принципу, пока не восстановим все удаленные строки, загружается предыдущая порция данных и добавляется в таблицу. Чтобы интерфейс оставался плавным, вновь загруженные строки корректно позиционируются в таблице.

2.2.6 Разработка взаимодействия с сервером

Стоит познакомиться с классом Request реализующий клиентскую часть приложения, которая взаимодействует с сервером посредством сокетов. Основные задачи, которые решает этот код:

- Синхронное выполнение запросов: Каждый запрос отправляется на сервер и ожидает ответа, прежде чем продолжить выполнение. Запросы к серверу выполняются последовательно, что упрощает логику обработки ответов и поддержание согласованного состояния.
  - Метод listenForMessages() (см. рисунок 33) выполняется в отдельном потоке и непрерывно читает входящие сообщения от сервера.

- Если сообщение начинается с {, оно рассматривается как JSON-объект.
- Если в JSON-сообщении есть поле action, выполняются соответствующие действия. Например, если action равно "UpdateTable", происходит обновление информации о таблице в пользовательском интерфейсе.
- Если сообщение не распознано, оно добавляется в очередь responseQueue.

- Асинхронное получение уведомлений: Клиент может получать уведомления от сервера, не блокируя основное выполнение программы. Клиент может получать уведомления и обновления от сервера в реальном времени без блокировки основного потока выполнения.
  - Метод sendRequest() (см. рисунок 34) принимает JSON-запрос, отправляет его на сервер и блокируется до получения ответа из очереди responseQueue.
  - Возвращает полученный ответ.

```
private void listenForMessages() {
    try {
        String message;
        while ((message = in.readLine()) != null) {
            if (message.startsWith("{")) {
                JSONObject jsonMessage = new JSONObject(message);
                if (jsonMessage.has("action")) {
                    String action = jsonMessage.getString("action");
                    if (action.equals("UpdateTable")) {
                        String tableName = jsonMessage.getString("tableName");
                        if(tableName.equals(main_form.getActiveTabTitle())){
                            main_form.labelInfo.setText("Таблица не актуальна");
                            main_form.jPanelInfo.setVisible(true);
                        }
                    }
                    // else if() тут могут быть другие оповещения с сервера
                    else {
                        responseQueue.put(message);
                    }
                } else {
                    responseQueue.put(message);
                }
            } else {
                responseQueue.put(message);
            }
        }
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
    }
}
```

Рисунок 33 – функция listenForMessages()

```
public String sendRequest(JSONObject request) {
    try {
        out.println(request.toString());
        return responseQueue.take();
    } catch (InterruptedException e) {
        e.printStackTrace();
        return null;
    }
}
```

Рисунок 33 – функция sendRequest ()

Плюсы подхода

1) Разделение задач:
   - Использование отдельного потока для прослушивания сообщений от сервера позволяет разделить асинхронные уведомления и синхронные запросы.
   - Синхронные запросы выполняются в основном потоке, что упрощает логику вызова и обработки ответов.
2) Асинхронное уведомление:
   - Асинхронная обработка сообщений от сервера позволяет своевременно обновлять интерфейс или выполнять другие действия в ответ на серверные события, не блокируя основной поток.
3) Безопасность потоков:
   - Использование BlockingQueue обеспечивает безопасную работу с очередью ответов в многопоточном окружении, избегая проблем с синхронизацией.
4) Удобство разработки:
   - Синхронные запросы упрощают логику клиента, так как каждый запрос блокируется до получения ответа, что облегчает обработку результатов.

Для системы оповещения необходимо модернизировать интерфейс нашей программы. Добавим сплывающее окно (см. рисунок 35) уведомляющее нас об изменениях в таблице и ее неактуальности. Окно представляет собой JPanel (в свойстах задан border для выделения) содержащий Jlabel и JButton “Обновить”. Так как нужные на компоненты находятся в нашем основном классе main_form, а использоваться они будут в классе Request, всем новым элементам необходимо задать подификатор доступа *protected* и сделать поля *статическими “static”*. Это можно сделать в ствойстах элемента (см. рисунок 36).

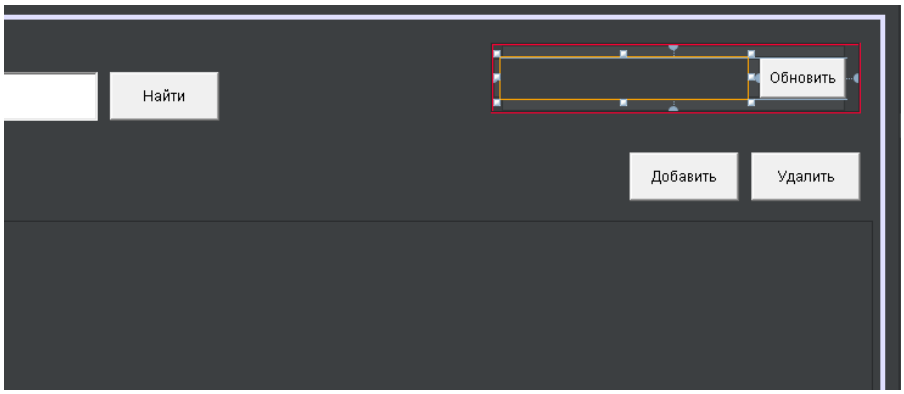


Рисунок 35 – всплавыющее окно

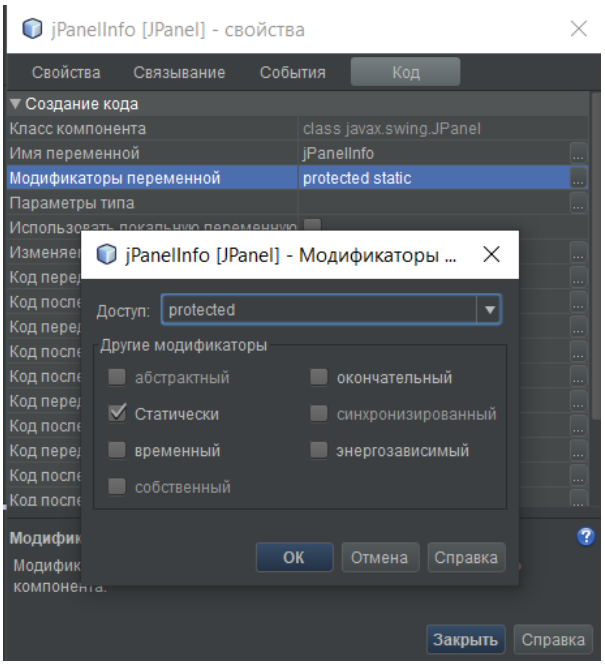


Рисунок 36 – Свойства элемента

Запросы

Запросы реализованы с использованием JSON-объектов для передачи данных между клиентом и сервером. Это позволяет легко структурировать и отправлять данные, а также получать ответы от сервера. Давайте рассмотрим каждый из запросов более подробно:

1) Получение имен таблиц < getTableNames >:

Описание: Этот метод создает JSON-запрос с действием GetTableNames для получения списка имен таблиц из базы данных.

JSON-запрос: {"action": "GetTableNames"}

Ответ: Ожидается, что сервер вернет JSON-объект с именами таблиц.

2) Получение типа таблицы < getTypeTable >:

Описание: Этот метод отправляет запрос с действием GetTypeTable, указывая имя таблицы, чтобы получить информацию о типе таблицы.

JSON-запрос: {"action": "GetTypeTable", "tableName": "имя_таблицы"}

Ответ: Ожидается, что сервер вернет информацию о типе таблицы.

3) Получение данных таблицы с пагинацией < getDataTable >:

Описание: Этот метод отправляет запрос для получения данных из таблицы с учетом пагинации.

JSON-запрос: {"action": "GetDataTable", "tableName": "имя_таблицы", "offset": 0, "query": "условие"}

Ответ: Ожидается, что сервер вернет данные таблицы с учетом указанного смещения (offset) и условий (query).

4) Добавление новой строки < addNewRow >:

Описание: Этот метод отправляет запрос для добавления новой строки в указанную таблицу.

JSON-запрос: {"action": "AddNewRow", "tableName": "имя_таблицы", "row": "данные_строки"}

Ответ: Ожидается, что сервер вернет подтверждение о добавлении новой строки.

5) Удаление строки < deleteRow >:

Описание: Этот метод отправляет запрос на удаление строки (или строк) из указанной таблицы.

JSON-запрос: {"action": "DeleteRow", "tableName": "имя_таблицы", "rows": "данные_для_удаления"}

Ответ: Ожидается, что сервер вернет подтверждение о выполнении удаления.

6) Обновление строки < updateRow >:

Описание: Этот метод отправляет запрос на обновление существующей строки в указанной таблице.

JSON-запрос: {"action": "UpdateRow", "tableName": "имя_таблицы", "NewRows": "новые_данные_строки", "EditRow": "номер_строки"}

Ответ: Ожидается, что сервер вернет подтверждение о выполнении обновления.

Заключение

В ходе данного курсового проекта была разработана клиент-серверная архитектура, обеспечивающая эффективное взаимодействие между пользователями и системой управления базами данных PostgreSQL. Клиентская часть предоставляет удобные и интуитивно понятные инструменты для работы с данными, включая таблицы, поиск, фильтрацию и сортировку. Серверная часть отвечает за обработку запросов, управление базой данных и поддержание её целостности и безопасности.

Разработанный протокол передачи информации между клиентом и сервером позволяет обеспечить надежную аутентификацию и авторизацию пользователей, а также управление сессиями и обработку ошибок. Это способствует высокой безопасности и устойчивости системы. Использование PostgreSQL позволяет реализовать сложные и производительные операции с данными, обеспечивая высокую степень гибкости и функциональности системы.

Литература

1) "PostgreSQL: Up and Running", Regina Obe, Leo Hsu

2) "Java. Полное руководство”, Герберт Шилдт

3) "Java Network Programming", Elliotte Rusty Harold

4) "JDBC API Tutorial and Reference", Maydene Fisher, Jon Ellis, Jonathan Bruce

Приложение А. Листинг программы

Сервер:

Postgres.java:

```
import java.io.*;
import static java.lang.System.in;
import java.net.*;
import java.sql.*;
import java.util.*;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * @author Kseroff
 */

public class Postgres {

    public static class DataBase {

        private Connection connection;
        private String url = "jdbc:postgresql://localhost:5432/project2?characterEncoding=UTF-8";
        private String user = "kseroff";
        private String password = "1111";
        private int limit;

        private List<ClientHandler> onlineUsers;

        public DataBase() throws SQLException {
            onlineUsers = new ArrayList<>();
            limit = 50;
            try {
                connection = DriverManager.getConnection(url, user, password);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        public synchronized void addOnlineUser(ClientHandler user) {
            onlineUsers.add(user);
        }

        public synchronized void removeOnlineUser(ClientHandler user) {
            onlineUsers.remove(user);
        }

        public String getTableInfoAsJson(String tableName) {
            JSONObject tableInfo = new JSONObject();
            JSONArray jsonArray = new JSONArray();
            try {
                Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

                // Получаем количество строк в таблице
                ResultSet countResultSet = statement.executeQuery("SELECT COUNT(*) AS row_count FROM " +
tableName);
```

```
            if (countResultSet.next()) {
                int rowCount = countResultSet.getInt("row_count");
                tableInfo.put("number_rows", rowCount);
            }

            // Получаем информацию о колонках
            ResultSet resultSet = statement.executeQuery("SELECT * FROM " + tableName + " LIMIT 1"); //
Ограничиваем запрос одной строкой для получения метаданных
            ResultSetMetaData metaData = resultSet.getMetaData();
            int columnCount = metaData.getColumnCount();
            tableInfo.put("number_columns", columnCount);
            for (int i = 1; i <= columnCount; i++) {
                JSONObject jsonObject = new JSONObject();
                jsonObject.put("column_name", metaData.getColumnName(i));
                jsonObject.put("data_type", metaData.getColumnTypeName(i));
                jsonArray.put(jsonObject);
            }
            tableInfo.put("columns", jsonArray);
        } catch (SQLException | JSONException e) {
            return e.getMessage();
        }
        return tableInfo.toString();
    }

    public String getDataTable(String tableName, int offset, String query) throws SQLException {
        JSONObject tableData = new JSONObject();
        JSONArray columns = new JSONArray();
        JSONArray rows = new JSONArray();
        int rowCount = 0;

        String sqlQuery = "SELECT * FROM " + tableName;
        String countQuery = "SELECT COUNT(*) AS row_count FROM " + tableName;

        if (query != null && !query.trim().isEmpty()) {
            sqlQuery += " WHERE " + query;
            countQuery += " WHERE " + query;
        }

        sqlQuery += " LIMIT ? OFFSET ?";

        try (PreparedStatement countStmt = connection.prepareStatement(countQuery);
             PreparedStatement dataStmt = connection.prepareStatement(sqlQuery)) {

            if (query != null && !query.trim().isEmpty()) {
                try (ResultSet countResultSet = countStmt.executeQuery()) {
                    if (countResultSet.next()) {
                        rowCount = countResultSet.getInt("row_count");
                    }
                }
            }

            dataStmt.setInt(1, limit);
            dataStmt.setInt(2, offset);

            try (ResultSet rs = dataStmt.executeQuery()) {
                ResultSetMetaData rsmd = rs.getMetaData();
                int columnCount = rsmd.getColumnCount();

                for (int i = 1; i <= columnCount; i++) {
                    columns.put(rsmd.getColumnName(i));
                }
                tableData.put("columns", columns);
```

```
                while (rs.next()) {
                    JSONObject row = new JSONObject();
                    for (int i = 1; i <= columnCount; i++) {
                        row.put(rsmd.getColumnName(i), rs.getObject(i));
                    }
                    rows.put(row);
                }
                tableData.put("rows", rows);
            }
        } catch (SQLException e) {
            return e.getMessage();
        }

        if (query != null && !query.trim().isEmpty()) {
            tableData.put("row_count", rowCount);
        }

        return tableData.toString();
    }

    public String addRowToTable(String tableName, String rowJson) throws SQLException {
        try {
            JSONObject rowObject = new JSONObject(rowJson);
            JSONArray columns = rowObject.getJSONArray("columns");
            JSONArray rows = rowObject.getJSONArray("rows");

            if (rows.length() == 0) {
                throw new IllegalArgumentException("No rows provided");
            }

            JSONObject row = rows.getJSONObject(0);

            StringBuilder columnNames = new StringBuilder();
            StringBuilder columnValues = new StringBuilder();

            for (int i = 0; i < columns.length(); i++) {
                columnNames.append(columns.getString(i));
                columnValues.append("?");
                if (i < columns.length() - 1) {
                    columnNames.append(", ");
                    columnValues.append(", ");
                }
            }

            String query = "INSERT INTO " + tableName + " (" + columnNames.toString() + ") VALUES (" +
columnValues.toString() + ")";
            PreparedStatement preparedStatement = connection.prepareStatement(query);

            // Получение информации о типах столбцов
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT " + columnNames.toString() + " FROM " + tableName + "
LIMIT 1");
            ResultSetMetaData metaData = rs.getMetaData();

            for (int i = 0; i < columns.length(); i++) {
                String columnName = columns.getString(i);
                Object value = row.get(columnName);
                int columnType = metaData.getColumnType(i + 1);

                switch (columnType) {
                    case Types.BIGINT:
                        preparedStatement.setLong(i + 1, row.getLong(columnName));
                        break;
```

```
                    case Types.INTEGER:
                        preparedStatement.setInt(i + 1, row.getInt(columnName));
                        break;
                    case Types.BOOLEAN:
                        preparedStatement.setBoolean(i + 1, row.getBoolean(columnName));
                        break;
                    case Types.FLOAT:
                        preparedStatement.setFloat(i + 1, (float) row.getDouble(columnName));
                        break;
                    case Types.DOUBLE:
                        preparedStatement.setDouble(i + 1, row.getDouble(columnName));
                        break;
                    case Types.VARCHAR:
                    case Types.CHAR:
                    case Types.LONGVARCHAR:
                    case Types.LONGNVARCHAR:
                        preparedStatement.setString(i + 1, row.getString(columnName));
                        break;
                    case Types.DATE:
                        preparedStatement.setDate(i + 1, java.sql.Date.valueOf(row.getString(columnName)));
                        break;
                    case Types.TIMESTAMP:
                        preparedStatement.setTimestamp(i + 1, Timestamp.valueOf(row.getString(columnName)));
                        break;
                    default:
                        preparedStatement.setObject(i + 1, value);
                        break;
                }
            }

            preparedStatement.executeUpdate();
            preparedStatement.close();
            rs.close();
            stmt.close();

            return "true";

        } catch (SQLException e) {
            return e.getLocalizedMessage();
        }
    }

    public String deleteRowsFromTable(String tableName, String rowNumbersStr) {
        try {
            String[] rowNumbersArr = rowNumbersStr.replaceAll("[\\[\\]]", "").split(",\\s*");
            int[] rowNumbers = new int[rowNumbersArr.length];
            for (int i = 0; i < rowNumbersArr.length; i++) {
                rowNumbers[i] = Integer.parseInt(rowNumbersArr[i]);
            }

            Statement statement = connection.createStatement();
            for (int i = rowNumbers.length - 1; i >= 0; i--) {
                int rowNum = rowNumbers[i];
                String sql = "DELETE FROM " + tableName + " WHERE ctid = (SELECT ctid FROM " + tableName
+ " ORDER BY ctid OFFSET " + rowNum + " LIMIT 1)";
                statement.executeUpdate(sql);
            }

            statement.close();
            return "true";
        } catch (SQLException e) {
            return e.getMessage();
        }
```

```
    }

    public String updateRow(String tableName, String newRowJson, int EditRow) {
        try {
            JSONArray jsonArray = new JSONArray(newRowJson);
            JSONObject jsonObject = jsonArray.getJSONObject(0);
            Statement statement = connection.createStatement();

            // Формирование динамического SQL-запроса для обновления
            StringBuilder sql = new StringBuilder("UPDATE ").append(tableName).append(" SET ");
            boolean first = true;
            for (Object key : jsonObject.keySet()) {
                if (!first) {
                    sql.append(", ");
                }
                sql.append(key).append(" = '").append(jsonObject.getString(key.toString())).append("'");
                first = false;
            }
            sql.append(" WHERE ctid = (SELECT ctid FROM ").append(tableName)
                    .append(" ORDER BY ctid OFFSET ").append(EditRow).append(" LIMIT 1)");

            String sqlString = sql.toString();
            statement.executeUpdate(sqlString);
            return "true";
        } catch (SQLException e) {
            return e.getMessage();
        }
    }

    public String getAllTables() {
        JSONArray jsonArray = new JSONArray();
        try {
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet tables = metaData.getTables(null, null, "%", new String[]{"TABLE"});
            while (tables.next()) {
                String tableName = tables.getString("TABLE_NAME");
                jsonArray.put(tableName);
            }
            tables.close();
        } catch (SQLException e) {
            return e.getMessage();
        }
        return jsonArray.toString();
    }
}

public static class ClientHandler extends Thread {

    private final Socket clientSocket;
    private final DataBase database;
    private static final List<ClientHandler> clients = Collections.synchronizedList(new ArrayList<>());
    private PrintWriter out;

    public ClientHandler(Socket socket, DataBase database) {
        this.clientSocket = socket;
        this.database = database;
        this.database.addOnlineUser(this);
        clients.add(this);
    }

    public void run() {
        try (BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))) {
            out = new PrintWriter(clientSocket.getOutputStream(), true);
```

```
    String request;
    while ((request = in.readLine()) != null) {
        JSONObject jsonRequest = new JSONObject(request);
        String action = jsonRequest.getString("action");

        switch (action) {
            case "GetTableNames":
                out.println(database.getAllTables());
                break;
            case "GetTypeTable":
                String tableName = jsonRequest.getString("tableName");
                out.println(database.getTableInfoAsJson(tableName));
                break;
            case "GetDataTable":
                tableName = jsonRequest.getString("tableName");
                int offset = jsonRequest.getInt("offset");
                String query = jsonRequest.optString("query", "");
                out.println(database.getDataTable(tableName, offset, query));
                break;
            case "AddNewRow":
                tableName = jsonRequest.getString("tableName");
                String row = jsonRequest.getString("row");
                String addResult = database.addRowToTable(tableName, row);
                if ("true".equals(addResult)) {
                    out.println("true");
                    notifyClients("UpdateTable", tableName);
                } else {
                    out.println("Error: " + addResult.replace("\n", ""));
                }
                break;
            case "DeleteRow":
                tableName = jsonRequest.getString("tableName");
                String rowsToDelete = jsonRequest.getString("rows");
                String deleteResult = database.deleteRowsFromTable(tableName, rowsToDelete);
                if ("true".equals(deleteResult)) {
                    out.println("true");
                    notifyClients("UpdateTable", tableName);
                } else {
                    out.println("Error: " + deleteResult.replace("\n", ""));
                }
                break;
            case "UpdateRow":
                tableName = jsonRequest.getString("tableName");
                String NewRows = jsonRequest.getString("NewRows");
                int EditRow = jsonRequest.getInt("EditRow");
                String UpfateResult = database.updateRow(tableName, NewRows, EditRow);
                if ("true".equals(UpfateResult)) {
                    out.println("true");
                    notifyClients("UpdateTable", tableName);
                } else {
                    out.println("Error: " + UpfateResult.replace("\n", ""));
                }
                break;
            case "disconnect":
                disconnect();
                return;
            default:
                out.println("Unknown action: " + action);
        }
    }
} catch (IOException | SQLException e) {
    e.printStackTrace();
```

```
            } finally {
                database.removeOnlineUser(this);
                clients.remove(this);
                try {
                    clientSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        private void disconnect() {
            try {
                in.close();
                out.close();
                clientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        private void notifyClients(String action, String tableName) {
            synchronized (clients) {
                for (ClientHandler client : clients) {
                    if (client != this) {
                        client.sendMessage(action, tableName);
                    }
                }
            }
        }

        private void sendMessage(String action, String tableName) {
            if (out != null) {
                JSONObject jsonMessage = new JSONObject();
                jsonMessage.put("action", action);
                jsonMessage.put("tableName", tableName);
                out.println(jsonMessage.toString());
            }
        }
    }

    public static void main(String[] args) {
        int port = 12345; // Укажите нужный порт
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            InetAddress inetAddress = InetAddress.getLocalHost();
            System.out.println("Server started...");
            System.out.println("IP Address: " + inetAddress.getHostAddress());
            System.out.println("Зort " + port);
            System.out.println("Host Name: " + inetAddress.getHostName());
            DataBase database = new DataBase();

            while (true) {
                Socket clientSocket = serverSocket.accept();

                ClientHandler clientHandler = new ClientHandler(clientSocket, database);
                clientHandler.start();
            }
        } catch (IOException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Клиент:

main_form.java:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Arrays;
import org.json.JSONArray;
import org.json.JSONObject;
import java.util.HashMap;
import java.util.Map;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

/**
 * @author Kseroff
 */
public class main_form extends javax.swing.JFrame {

    final int limit;
    int deleteRow;
    boolean wasAtTop;
    int offset;
    int numberColumns;
    String FieldsTypeJson;
    JTable TableNow;
    Map<String, Map<String, String>> FilterInfo;
    String SearchValue;

    public static Request client;

    public main_form() {
        this.limit = 50;
        setResizable(false);
        initComponents();

        client = new Request("localhost", 12345);

        this.FilterInfo = new HashMap<>();
        this.offset = 0;
        this.deleteRow = 0;
        this.numberColumns = 0;
        this.FieldsTypeJson = "";
        this.SearchValue = "";
        this.wasAtTop = true;

        buttonFilters.addActionListener(new OpenFilterButton());
        buttonAdd.addActionListener(new OpenAddEditButton());
        checkboxFilters.addItemListener(new CheckboxFilter());
        buttonSearch.addActionListener(new ButtonSearch());
        buttonDelete.addActionListener(new DeleteRowTableButton());
        jTabbedPane.addChangeListener(new PanelChanged());
        buttonUpdate.addActionListener(new ButtonUpdate());

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
```

```
            client.close();
            dispose();
        }
    });

    createDynamicPanels(client.getTableNames());
}

private void createDynamicPanels(String jsonString) {
    // Удаляем все существующие панели(если они есть неожиданно)
    jTabbedPane.removeAll();

    JSONArray jsonArray = new JSONArray(jsonString);

    for (int i = 0; i < jsonArray.length(); i++) {
        String tableName = jsonArray.getString(i);
        JPanel panel = new JPanel(new BorderLayout());
        JTable table = new JTable();
        table.setAutoCreateRowSorter(true);
        table.setModel(new DefaultTableModel() {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false; // Запрет редактирования ячеек
            }
        });

        table.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() == 2) {
                    JTable target = (JTable) e.getSource();
                    int row = target.getSelectedRow();
                    DefaultTableModel model = (DefaultTableModel) target.getModel();
                    Map<String, String> keyValueMap = new HashMap<>();

                    // Получение данных из выбранной строки и добавление их в Map
                    for (int col = 0; col < model.getColumnCount(); col++) {
                        String columnName = model.getColumnName(col);
                        String cellValue = model.getValueAt(row, col).toString();
                        keyValueMap.put(columnName, cellValue);
                    }

                    AddEditForm addform = new AddEditForm(new InterfaceCallback() {
                        @Override
                        public void onApply(String query) {
                            JSONObject jsonArray = new JSONObject(query);
                            JSONArray rows = jsonArray.getJSONArray("rows");

                            JSONObject rowJson = rows.getJSONObject(0); // Предполагаем, что rows содержит только
одну строку
                            for (int i = 0; i < model.getColumnCount(); i++) {
                                Object value = rowJson.opt(model.getColumnName(i)); // Получаем значение по имени
столбца
                                if (value != null) {
                                    model.setValueAt(value, row, i);
                                }
                            }
                        }

                        @Override
                        public void onCancel() {
                        }
                    }, client, keyValueMap, FieldsTypeJson, getActiveTabTitle(), "EDIT", row);
```

```
            }
        }
    });

    JScrollPane scrollPane = new JScrollPane(table);

    // Добавляем AdjustmentListener для вертикального скроллбара
    JScrollBar verticalScrollBar = scrollPane.getVerticalScrollBar();
    verticalScrollBar.addAdjustmentListener((AdjustmentEvent e) -> {
        if (!e.getValueIsAdjusting() && (tableName == null ? getActiveTabTitle() == null :
tableName.equals(getActiveTabTitle()))) {
            DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
            int extent = verticalScrollBar.getModel().getExtent();
            int maximum = verticalScrollBar.getMaximum();
            int value = verticalScrollBar.getValue();

            if ((value + extent == maximum) && numberColumns > offset + limit) {
                wasAtTop = false;
                offset += limit;

                String Jsonresult = client.getDataTable(tableName, offset, SearchValue);
                JSONObject jsonResult = new JSONObject(Jsonresult);
                if (jsonResult.has("row_count")) {
                    this.numberColumns = jsonResult.getInt("row_count");
                }
                addRowTable(Jsonresult, false);

                if ((offset - limit * 2) > -1) {
                    deleteRow += limit;
                    int rowHeight = table.getRowHeight();
                    int rowsToDelete = Math.min(limit, model.getRowCount());
                    for (int j = 0; j < limit; j++) {
                        model.removeRow(0);
                    }
                    int adjustment = rowsToDelete * rowHeight;
                    int newScrollValue = Math.max(value - adjustment, 0);
                    SwingUtilities.invokeLater(() -> verticalScrollBar.setValue(newScrollValue));
                }

            } else if (value == 0 && deleteRow > 0) {
                deleteRow -= limit;
                if(!wasAtTop)
                    offset -= limit;
                offset -= limit;

                wasAtTop = true;
                int rowHeight = table.getRowHeight();

                String Jsonresult = client.getDataTable(tableName, offset, SearchValue);
                JSONObject jsonResult = new JSONObject(Jsonresult);
                if (jsonResult.has("row_count")) {
                    this.numberColumns = jsonResult.getInt("row_count");
                }
                addRowTable(Jsonresult, true);

                int adjustment = limit * rowHeight;
                int newScrollValue = value + adjustment;

                SwingUtilities.invokeLater(() -> verticalScrollBar.setValue(newScrollValue));

                for (int j = 1; j <= limit; j++) {
                    model.removeRow(model.getRowCount() - 1);
```

```
                }

            }
        }
    });

    panel.add(scrollPane, BorderLayout.CENTER);
    jTabbedPane.addTab(tableName, panel);
  }
}

private void FillTableWithInfor() {
  offset = 0;
  wasAtTop = true;
  jPanelInfo.setVisible(false);

  checkboxFilters.setState(false);
  String tableName = getActiveTabTitle();

  String InfoJson = client.getDataTable(tableName, offset, "");

  FieldsTypeJson = client.getTypeTable(tableName);
  JSONObject jsonArray = new JSONObject(FieldsTypeJson);
  this.numberColumns = jsonArray.getInt("number_rows");

  jsonArray = new JSONObject(InfoJson);
  JSONArray columns = jsonArray.getJSONArray("columns");

  DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
  model.setRowCount(0);
  model.setColumnCount(0);

  for (int i = 0; i < columns.length(); i++) {
    model.addColumn(columns.getString(i));
  }

  addRowTable(InfoJson, false);

  if (checkboxFilters.getState()) {
    SearchQuery(getActiveTabTitle());
  }
}

public void addRowTable(String infoJson, boolean addToStart) {
 JSONObject jsonObject = new JSONObject(infoJson);
 JSONArray columns = jsonObject.getJSONArray("columns");
 JSONArray rows = jsonObject.getJSONArray("rows");

 DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
 Map<String, Integer> columnIndices = new HashMap<>();

 for (int columnIndex = 0; columnIndex < columns.length(); columnIndex++) {
   String columnName = columns.getString(columnIndex);
   columnIndices.put(columnName, columnIndex);
 }

 for (int i = 0; i < rows.length(); i++) {
   JSONObject rowJsonObject = rows.getJSONObject(i);
   Object[] rowData = new Object[columns.length()];
   for (int j = 0; j < columns.length(); j++) {
     String columnName = columns.getString(j);
     int columnIndex = columnIndices.get(columnName);
     rowData[columnIndex] = rowJsonObject.get(columnName);
```

```
        }

        if (addToStart) {
            model.insertRow(i, rowData);
        } else {
            model.addRow(rowData);
        }
    }
}

    public static String getActiveTabTitle() {
        int index = jTabbedPane.getSelectedIndex();
        if (index != -1) {
            return jTabbedPane.getTitleAt(index);
        } else {
            return null; // Если нет активной вкладки
        }
    }

    private void SearchQuery(String tableName) {
        String searchText = textFieldSearch.getText().trim();
        StringBuilder queryBuilder = new StringBuilder("");
        if (!searchText.equals("")) {
            TableModel model = TableNow.getModel();
            queryBuilder.append("(");

            Map<String, Object> columnInfo = new HashMap<>();

            JSONObject jsonObject = new JSONObject(FieldsTypeJson);
            JSONArray columns = jsonObject.getJSONArray("columns");

            boolean isNumeric = searchText.matches("-?\\d+");

            for (int i = 0; i < columns.length(); i++) {
                JSONObject column = columns.getJSONObject(i);
                String columnName = column.getString("column_name");
                String dataType = column.getString("data_type");
                columnInfo.put(columnName, dataType);
            }

            for (int col = 0; col < model.getColumnCount(); col++) {
                String columnName = model.getColumnName(col);
                String dataType = (String) columnInfo.get(columnName);
                if (dataType.equals("bool")) {
                    continue;
                } else if (dataType.contains("int") || dataType.contains("serial")
                        || dataType.contains("decimal") || dataType.contains("numeric")) {
                    if (isNumeric) {
                        if (queryBuilder.length() > 1) {
                            queryBuilder.append(" OR ");
                        }
                        queryBuilder.append("CAST(").append(columnName).append(" AS TEXT) LIKE
'%").append(searchText).append("%'");
                    }
                } else if (dataType.equals("date") || dataType.equals("timestamp")) {
                    if (isNumeric) {
                        if (queryBuilder.length() > 1) {
                            queryBuilder.append(" OR ");
                        }
                        queryBuilder.append("TO_CHAR(").append(columnName).append(", 'YYYY-MM-DD') LIKE
'%").append(searchText).append("%'");
                    }
                } else {
```

```
                if (queryBuilder.length() > 1) {
                    queryBuilder.append(" OR ");
                }
                queryBuilder.append(columnName).append(" LIKE '%").append(searchText).append("%'");
            }
        }

        // Завершаем построение строки запроса
        queryBuilder.append(")");
    }

    if (checkboxFilters.getState()) {
        if (!(queryBuilder.toString().equals("") || queryBuilder == null)) {
            queryBuilder.append(" AND ");
        }
        Map<String, String> result = FilterInfo.get(tableName);
        if (result != null) {
            queryBuilder.append(result.get("FilterQuery") == null ? "" : result.get("FilterQuery"));
        }
    }

    String result = queryBuilder.toString();
    SearchValue = result;
    if ("".equals(result)) {
        FillTableWithInfor();
    } else {
        DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
        model.setRowCount(0);
        wasAtTop = true;
        String Jsonresult = client.getDataTable(tableName, 0, result);
        JSONObject jsonResult = new JSONObject(Jsonresult);
        if (jsonResult.has("row_count")) {
            this.numberColumns = jsonResult.getInt("row_count");
        }
        addRowTable(Jsonresult, false);
    }
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jPanel2 = new javax.swing.JPanel();
    buttonDelete = new java.awt.Button();
    buttonAdd = new java.awt.Button();
    buttonSearch = new java.awt.Button();
    textFieldSearch = new java.awt.TextField();
    buttonFilters = new java.awt.Button();
    checkboxFilters = new java.awt.Checkbox();
    jPanelInfo = new javax.swing.JPanel();
    buttonUpdate = new java.awt.Button();
    labelInfo = new javax.swing.JLabel();
    filler1 = new javax.swing.Box.Filler(new java.awt.Dimension(0, 0), new java.awt.Dimension(0, 0), new
java.awt.Dimension(32767, 0));
    label1 = new java.awt.Label();
    jTabbedPane = new javax.swing.JTabbedPane();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

```
        setTitle("rootFrame");
        setBackground(new java.awt.Color(255, 255, 255));

        buttonDelete.setActionCommand("buttonDelete");
        buttonDelete.setLabel("Удалить");

        buttonAdd.setActionCommand("buttonAdd");
        buttonAdd.setLabel("Добавить");

        buttonSearch.setActionCommand("buttonSearch");
        buttonSearch.setLabel("Найти");

        textFieldSearch.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        textFieldSearch.setFont(new java.awt.Font("Dialog", 0, 20)); // NOI18N

        buttonFilters.setActionCommand("buttonFilters");
        buttonFilters.setLabel("Фильтры");

        checkboxFilters.setLabel("Включить фильтр");

        jPanelInfo.setBorder(javax.swing.BorderFactory.createEtchedBorder(new java.awt.Color(255, 0, 51), null));
        jPanelInfo.setToolTipText("");

        buttonUpdate.setLabel("Обновить");

        javax.swing.GroupLayout jPanelInfoLayout = new javax.swing.GroupLayout(jPanelInfo);
        jPanelInfo.setLayout(jPanelInfoLayout);
        jPanelInfoLayout.setHorizontalGroup(
            jPanelInfoLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanelInfoLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(labelInfo, javax.swing.GroupLayout.DEFAULT_SIZE, 203, Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(buttonUpdate, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        jPanelInfoLayout.setVerticalGroup(
            jPanelInfoLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanelInfoLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelInfoLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(buttonUpdate, javax.swing.GroupLayout.DEFAULT_SIZE, 34, Short.MAX_VALUE)
                    .addComponent(labelInfo, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );

        buttonUpdate.getAccessibleContext().setAccessibleName("buttonUpdate");

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addComponent(buttonFilters, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(checkboxFilters, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 482,
Short.MAX_VALUE)
                .addComponent(filler1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(120, 120, 120)
                .addComponent(buttonAdd, javax.swing.GroupLayout.PREFERRED_SIZE, 90,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(buttonDelete, javax.swing.GroupLayout.PREFERRED_SIZE, 90,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(label1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addComponent(textFieldSearch, javax.swing.GroupLayout.PREFERRED_SIZE, 368,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(buttonSearch, javax.swing.GroupLayout.PREFERRED_SIZE, 90,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jPanelInfo, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addContainerGap())
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addGap(13, 13, 13)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addComponent(jPanelInfo, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
                        .addComponent(label1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(4, 4, 4)
                        .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(textFieldSearch, javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(buttonSearch, javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)))
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
                        .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(buttonFilters, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(checkboxFilters, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(filler1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(32, 32, 32))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
                        .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(buttonDelete, javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(buttonAdd, javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
                .addContainerGap())))
        );

        buttonFilters.getAccessibleContext().setAccessibleName("buttonFilters");
        jPanelInfo.getAccessibleContext().setAccessibleName("jPanelInfo");

        jTabbedPane.setBackground(new java.awt.Color(0, 0, 0));

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jTabbedPane)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(0, 0, Short.MAX_VALUE)))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jTabbedPane, javax.swing.GroupLayout.DEFAULT_SIZE, 322, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    class OpenFilterButton implements ActionListener {

        @Override
        public void actionPerformed(java.awt.event.ActionEvent evt) {

            String tableName = getActiveTabTitle();

            Map<String, String> result = FilterInfo.get(tableName);
            String FillFilterFields = "";
            if (result != null) {
                FillFilterFields = result.get("FilterQuery") == null ? "" : result.get("FilterQuery");
            }

            FilterForm filterForm = new FilterForm(new InterfaceCallback() {
                @Override
                public void onApply(String query) {
                    Map<String, String> value = new HashMap<>();
                    value.put("FilterQuery", query);
                    FilterInfo.put(tableName, value);

                    if (checkboxFilters.getState()) {
                        SearchQuery(tableName);
                    }
                }

                @Override
                public void onCancel() {
                    // закрытие формы
```

```
            }
        }, FieldsTypeJson, FillFilterFields);
    }
}

class OpenAddEditButton implements ActionListener {

    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {

        DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
        Map<String, String> keyValueMap = new HashMap<>();
        for (int col = 0; col < model.getColumnCount(); col++) {
            String columnName = model.getColumnName(col);
            keyValueMap.put(columnName, "");
        }

        AddEditForm addform = new AddEditForm(new InterfaceCallback() {
            @Override
            public void onApply(String query) {
                addRowTable(query, false);
            }

            @Override
            public void onCancel() {
            }
        }, client, keyValueMap, FieldsTypeJson, getActiveTabTitle(), "ADD", 0);
    }
}

class CheckboxFilter implements ItemListener {

    @Override
    public void itemStateChanged(java.awt.event.ItemEvent evt) {
        SearchQuery(getActiveTabTitle());
    }
}

class ButtonSearch implements ActionListener {

    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SearchQuery(getActiveTabTitle());
    }
}

class ButtonUpdate implements ActionListener {

    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        FillTableWithInfor();
    }
}

class DeleteRowTableButton implements ActionListener {

    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        int[] selected = TableNow.getSelectedRows();
        String[] a = Arrays.stream(selected)
                .map(i -> i + deleteRow)
                .mapToObj(String::valueOf)
                .toArray(String[]::new);
```

```
        String QuerySelected = Arrays.toString(a);

        if ("true".equals(client.deleteRow(getActiveTabTitle(), QuerySelected))) {

            DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
            for (int i = selected.length - 1; i >= 0; i--) {
                model.removeRow(selected[i]);
            }
        }
    }
}

class PanelChanged implements ChangeListener {

    @Override
    public void stateChanged(ChangeEvent evt) {
        // очистка предыдущей таблицы
        if (TableNow != null) {
            DefaultTableModel model = (DefaultTableModel) TableNow.getModel();
            model.setRowCount(0);
            model.setColumnCount(0);
        }
        jPanelInfo.setVisible(false);
        TableNow = getTableByTabName(getActiveTabTitle());
        FillTableWithInfor();
    }
}

public JTable getTableByTabName(String tabName) {
    int tabIndex = jTabbedPane.indexOfTab(tabName);
    if (tabIndex == -1) {
        return null; // Таблица не найдена
    }

    Component component = jTabbedPane.getComponentAt(tabIndex);
    if (component instanceof JPanel) {
        JPanel panel = (JPanel) component;
        for (Component comp : panel.getComponents()) {
            if (comp instanceof JScrollPane) {
                JScrollPane scrollPane = (JScrollPane) comp;
                JViewport viewport = scrollPane.getViewport();
                Component view = viewport.getView();
                if (view instanceof JTable) {
                    return (JTable) view;
                }
            }
        }
    }

    return null; // Таблица не найдена
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
```

```
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(main_form.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(main_form.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(main_form.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(main_form.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    }
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new main_form().setVisible(true);

        }
    });
}

// Variables declaration - do not modify
private java.awt.Button buttonAdd;
private java.awt.Button buttonDelete;
private java.awt.Button buttonFilters;
private java.awt.Button buttonSearch;
private java.awt.Button buttonUpdate;
private java.awt.Checkbox checkboxFilters;
private javax.swing.Box.Filler filler1;
private javax.swing.JPanel jPanel2;
protected static javax.swing.JPanel jPanelInfo;
private static javax.swing.JTabbedPane jTabbedPane;
private java.awt.Label label1;
protected static javax.swing.JLabel labelInfo;
private java.awt.TextField textFieldSearch;
// End of variables declaration
}
```

Request.java:

```
import javax.swing.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import org.json.JSONObject;

public class Request {

    private Socket socket;
    private PrintWriter out;
    private BufferedReader in;
```

```
private BlockingQueue<String> responseQueue;

public Request(String address, int port) {
    try {
        socket = new Socket(address, port);
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        responseQueue = new LinkedBlockingQueue<>();
        System.out.println("Connected to server");

        // прослушка с сервера
        new Thread(this::listenForMessages).start();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private void listenForMessages() {
    try {
        String message;
        while ((message = in.readLine()) != null) {
            if (message.startsWith("{")) {
                JSONObject jsonMessage = new JSONObject(message);
                if (jsonMessage.has("action")) {
                    String action = jsonMessage.getString("action");
                    if (action.equals("UpdateTable")) {
                        String tableName = jsonMessage.getString("tableName");
                        if(tableName.equals(main_form.getActiveTabTitle())){
                            main_form.labelInfo.setText("Таблица не актуальна");
                            main_form.jPanelInfo.setVisible(true);
                        }
                    }
                    // else if() тут могут быть другие оповещения с сервера
                    else {
                        responseQueue.put(message);
                    }
                } else {
                    responseQueue.put(message);
                }
            } else {
                responseQueue.put(message);
            }
        }
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
    }
}

public String sendRequest(JSONObject request) {
    try {
        out.println(request.toString());
        return responseQueue.take();
    } catch (InterruptedException e) {
        e.printStackTrace();
        return null;
    }
}

public String getTableNames() {
    JSONObject request = new JSONObject();
    request.put("action", "GetTableNames");
    return sendRequest(request);
}
```

```
    public String getTypeTable(String tableName) {
        JSONObject request = new JSONObject();
        request.put("action", "GetTypeTable");
        request.put("tableName", tableName);
        return sendRequest(request);
    }

    public String getDataTable(String tableName, int offset, String query) {
        JSONObject request = new JSONObject();
        request.put("action", "GetDataTable");
        request.put("tableName", tableName);
        request.put("offset", offset);
        request.put("query", query);
        return sendRequest(request);
    }

    public String addNewRow(String tableName, String row) {
        JSONObject request = new JSONObject();
        request.put("action", "AddNewRow");
        request.put("tableName", tableName);
        request.put("row", row);
        return sendRequest(request);
    }

    public String deleteRow(String tableName, String rows) {
        JSONObject request = new JSONObject();
        request.put("action", "DeleteRow");
        request.put("tableName", tableName);
        request.put("rows", rows);
        return sendRequest(request);
    }

    public String updateRow(String tableName, String newRowJson, int row) {
        JSONObject request = new JSONObject();
        request.put("action", "UpdateRow");
        request.put("tableName", tableName);
        request.put("NewRows", newRowJson);
        request.put("EditRow", row);
        return sendRequest(request);
    }

    public void close() {
        try {
            JSONObject request = new JSONObject();
            request.put("action", "disconnect");
            out.println(request.toString());

            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

AddEditForm.java:

```
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;
import java.util.HashMap;
import java.util.Map;
import org.json.JSONArray;
import org.json.JSONObject;
import com.toedter.calendar.JDateChooser;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * @author Kseroff
 */
public final class AddEditForm {

    public AddEditForm(InterfaceCallback callback, Request client, Map<String, String> keyValueMap,
        String FieldsTypeJson, String tableName, String action, int editRow) {
        // Создание главного окна
        JSONObject jsonObj = new JSONObject(FieldsTypeJson);
        JSONArray jsonArray = jsonObj.getJSONArray("columns");
        JFrame frame = new JFrame("Dynamic Form");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setResizable(false); // Запрет изменения размера окна
        frame.setLayout(new BorderLayout());

        // Панель для текстовых полей
        JPanel rightPanel = new JPanel(new GridBagLayout());
        rightPanel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

        Map<JLabel, JComponent> labelFieldMap = new HashMap<>();

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.anchor = GridBagConstraints.WEST;
        gbc.insets = new Insets(5, 5, 5, 5);

        int row = 0;
        for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject fieldObject = jsonArray.getJSONObject(i);
            String columnName = fieldObject.getString("column_name");
            String dataType = fieldObject.getString("data_type");

            JLabel label = new JLabel(columnName);
            JComponent fieldComponent;

            if (dataType.contains("date") || dataType.contains("timestamp")) {
                JDateChooser dateChooser = new JDateChooser();
                String value = keyValueMap.get(columnName);
                if (value != null && !value.isEmpty()) {
                    try {
                        Date date = new SimpleDateFormat("yyyy-MM-dd").parse(value);
                        dateChooser.setDate(date);
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }
                }
                fieldComponent = dateChooser;
            } else if (dataType.contains("bool")) {
                JPanel radioPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
                JRadioButton yesRadioButton = new JRadioButton("Yes");
                JRadioButton noRadioButton = new JRadioButton("No");
                ButtonGroup group = new ButtonGroup();
                group.add(yesRadioButton);
```

```
            group.add(noRadioButton);
            radioPanel.add(yesRadioButton);
            radioPanel.add(noRadioButton);
            String value = keyValueMap.get(columnName);
            if (value != null) {
                if (value.equalsIgnoreCase("true")) {
                    yesRadioButton.setSelected(true);
                } else if (value.equalsIgnoreCase("false")) {
                    noRadioButton.setSelected(true);
                }
            }
            fieldComponent = radioPanel;
        } else if (dataType.contains("int") || dataType.contains("serial") || dataType.contains("decimal") ||
dataType.contains("numeric")) {
            JTextField textField = new JTextField(20);
            textField.addKeyListener(new KeyAdapter() {
                @Override
                public void keyTyped(KeyEvent e) {
                    char c = e.getKeyChar();
                    if (!Character.isDigit(c) && c != KeyEvent.VK_BACK_SPACE && c != KeyEvent.VK_DELETE)
{
                        e.consume();
                    }
                }
            });
            String value = keyValueMap.get(columnName);
            if (value != null) {
                textField.setText(value);
            }
            fieldComponent = textField;
        } else {
            JTextField textField = new JTextField(20);
            String value = keyValueMap.get(columnName);
            if (value != null) {
                textField.setText(value);
            }
            fieldComponent = textField;
        }

        labelFieldMap.put(label, fieldComponent);

        gbc.gridx = 0;
        gbc.gridy = row;
        gbc.weightx = 0.1;
        gbc.anchor = GridBagConstraints.WEST;
        rightPanel.add(label, gbc);

        gbc.gridx = 1;
        gbc.weightx = 1.0;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.anchor = GridBagConstraints.EAST;
        rightPanel.add(fieldComponent, gbc);

        row++;
    }

    frame.add(rightPanel, BorderLayout.CENTER);
    // Панель для кнопок "Сохранить" и "Отмена"
    JPanel buttonPanelBottom = new JPanel(new FlowLayout(FlowLayout.RIGHT));

    // Кнопка "Сохранить"
    JButton saveButton = new JButton("Сохранить");
    saveButton.addActionListener(new ActionListener() {
```

```
public void actionPerformed(ActionEvent e) {
    Map<String, Object> savedData = new HashMap<>();
    for (Map.Entry<JLabel, JComponent> entry : labelFieldMap.entrySet()) {
        String key = entry.getKey().getText();
        JComponent component = entry.getValue();

        if (component instanceof JTextField) {
            savedData.put(key, ((JTextField) component).getText());
        } else if (component instanceof JDateChooser) {
            Date date = ((JDateChooser) component).getDate();
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            savedData.put(key, sdf.format(date));
        } else if (component instanceof JPanel) {
            for (Component comp : ((JPanel) component).getComponents()) {
                if (comp instanceof JRadioButton && ((JRadioButton) comp).isSelected()) {
                    savedData.put(key, ((JRadioButton) comp).getText().equals("Yes"));
                }
            }
        }
    }

    // Создание JSON структуры
    JSONObject jsonObject = new JSONObject();
    JSONArray columnsArray = new JSONArray();
    JSONArray rowsArray = new JSONArray();

    for (Map.Entry<String, Object> entry : savedData.entrySet()) {
        columnsArray.put(entry.getKey());
    }

    JSONObject rowObject = new JSONObject();
    for (Map.Entry<String, Object> entry : savedData.entrySet()) {
        if (entry.getValue() instanceof byte[]) {
            byte[] byteArray = (byte[]) entry.getValue();
            JSONArray jsonArray = new JSONArray();
            for (byte b : byteArray) {
                jsonArray.put(b);
            }
            rowObject.put(entry.getKey(), jsonArray);
        } else {
            rowObject.put(entry.getKey(), entry.getValue());
        }
    }
    rowsArray.put(rowObject);

    jsonObject.put("columns", columnsArray);
    jsonObject.put("rows", rowsArray);

    String jsonData = jsonObject.toString();

    String req;
    if ("ADD".equals(action)) {
        req = main_form.client.addNewRow(tableName, jsonData);
    } else {
        JSONObject NewjsonArray = new JSONObject(jsonData);
        JSONArray NewrowsJS = NewjsonArray.getJSONArray("rows");

        req = main_form.client.updateRow(tableName, NewrowsJS.toString(), editRow);
    }
    if ("true".equals(req)) {
        callback.onApply(jsonData);
        frame.dispose();
    } else {
```

```
                JOptionPane.showMessageDialog(null, req, "Error", JOptionPane.WARNING_MESSAGE);
            }
        }
    });
    buttonPanelBottom.add(saveButton);

    // Кнопка "Отмена"
    JButton cancelButton = new JButton("Отмена");
    cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            callback.onCancel();
            frame.dispose();
        }
    });
    buttonPanelBottom.add(cancelButton);

    frame.add(buttonPanelBottom, BorderLayout.SOUTH);

    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
  }

}
```

FilterForm.java:

```
import org.json.JSONArray;
import org.json.JSONObject;
import com.toedter.calendar.JDateChooser;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * @author Kseroff
 */
public final class FilterForm extends JFrame {
    private List<JPanel> filterPanels = new ArrayList<>();
    private InterfaceCallback callback;

    public FilterForm(InterfaceCallback callback, String json, String FillFilterFields) {
        this.callback = callback;
        JSONObject jsonObj = new JSONObject(json);
        JSONArray jsonArray = jsonObj.getJSONArray("columns");
        setTitle("Filter Form");
        setSize(600, 150);
        setResizable(false);
        setLayout(new BorderLayout());
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
        JScrollPane scrollPane = new JScrollPane(mainPanel);
        add(scrollPane, BorderLayout.CENTER);
```

```
for (int i = 0; i < jsonArray.length(); i++) {
    JPanel MainFiledPanel = new JPanel(new GridLayout(1, 2));
    mainPanel.add(MainFiledPanel);

    JSONObject jsonObject = jsonArray.getJSONObject(i);
    String columnName = jsonObject.getString("column_name");
    String dataType = jsonObject.getString("data_type");

    JPanel filterPanel = new JPanel(new GridLayout(0, 1));
    filterPanels.add(filterPanel);
    JPanel checkBoxPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
    JCheckBox checkBox = new JCheckBox(columnName);
    checkBoxPanel.add(checkBox, BorderLayout.WEST);

    if (!dataType.equals("bool")) {
        JButton addButton = new JButton("+");
        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                addFilterField(filterPanel, dataType, MainFiledPanel);
                revalidate();
                repaint();
            }
        });
        checkBoxPanel.add(addButton, BorderLayout.EAST);
        addButton.doClick();
    } else {
        JPanel fieldPanel = new JPanel();
        JRadioButton yesRadioButton = new JRadioButton("Yes");
        JRadioButton noRadioButton = new JRadioButton("No");
        yesRadioButton.doClick();
        fieldPanel.add(yesRadioButton);
        fieldPanel.add(noRadioButton);
        filterPanel.add(fieldPanel, BorderLayout.WEST);
    }
    MainFiledPanel.add(checkBoxPanel);
    MainFiledPanel.add(filterPanel);
}

JPanel buttonPanel = new JPanel(new GridLayout(1, 2));
JButton applyButton = new JButton("Применить");
applyButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String query = buildQuery();
        callback.onApply(query);
        dispose();
    }
});
buttonPanel.add(applyButton);

JButton exitButton = new JButton("Отмена");
exitButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        callback.onCancel();
        dispose();
    }
});
buttonPanel.add(exitButton);
add(buttonPanel, BorderLayout.SOUTH);
```

```
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setVisible(true);

        createFilterFieldsFromConditions(FillFilterFields);
    }

    private void addFilterField(JPanel filterPanel, String dataType, JPanel MainFiledPanel) {
        JPanel fieldPanel = new JPanel();
        fieldPanel.setLayout(new BoxLayout(fieldPanel, BoxLayout.X_AXIS));

        if (dataType.contains("int") || dataType.contains("serial")
                || dataType.contains("decimal") || dataType.contains("numeric")) {
            JTextField minValueField = new JTextField();
            minValueField.setPreferredSize(new Dimension(100, 30));
            minValueField.addKeyListener(new KeyAdapter() {
                @Override
                public void keyTyped(KeyEvent e) {
                    char c = e.getKeyChar();
                    JTextField textField = (JTextField) e.getComponent();
                    String text = textField.getText();
                    if (!Character.isDigit(c) && c != KeyEvent.VK_BACK_SPACE && c != KeyEvent.VK_DELETE
&& c != '.'
                            || (c == '.' && text.contains("."))) {
                        e.consume();
                    }
                }
            });
            JTextField maxValueField = new JTextField();
            maxValueField.setPreferredSize(new Dimension(100, 30));
            maxValueField.addKeyListener(new KeyAdapter() {
                @Override
                public void keyTyped(KeyEvent e) {
                    char c = e.getKeyChar();
                    JTextField textField = (JTextField) e.getComponent();
                    String text = textField.getText();
                    if (!Character.isDigit(c) && c != KeyEvent.VK_BACK_SPACE && c != KeyEvent.VK_DELETE
&& c != '.'
                            || (c == '.' && text.contains("."))) {
                        e.consume();
                    }
                }
            });

            fieldPanel.add(minValueField);
            fieldPanel.add(maxValueField);
        } else if (dataType.equals("date") || dataType.equals("timestamp")) {
            JDateChooser minDateChooser = new JDateChooser();
            minDateChooser.setPreferredSize(new Dimension(100, 30));
            JDateChooser maxDateChooser = new JDateChooser();
            maxDateChooser.setPreferredSize(new Dimension(100, 30));
            fieldPanel.add(minDateChooser);
            fieldPanel.add(maxDateChooser);
        } else {
            JTextField textField = new JTextField();
            textField.setPreferredSize(new Dimension(200, 30));
            fieldPanel.add(textField);
        }

        fieldPanel.setPreferredSize(new Dimension(fieldPanel.getWidth(), fieldPanel.getHeight() + 35));
        MainFiledPanel.setPreferredSize(new Dimension(MainFiledPanel.getWidth(), MainFiledPanel.getHeight() +
fieldPanel.getHeight()));
        JButton removeButton = new JButton("-");
        removeButton.addActionListener(new ActionListener() {
```

```
        @Override
        public void actionPerformed(ActionEvent e) {
            if (filterPanel.getComponentCount() > 1) {
                MainFiledPanel.setPreferredSize(new Dimension(MainFiledPanel.getWidth(),
MainFiledPanel.getHeight() - (fieldPanel.getHeight() * 2)));
                setSize(getWidth(), getHeight() - fieldPanel.getHeight());
                filterPanel.remove(fieldPanel);
                filterPanel.revalidate();
                filterPanel.repaint();
            }
            else{
                Component[] components = fieldPanel.getComponents();
                for (Component component : components) {
                    if (component instanceof JTextField) {
                        JTextField textField = (JTextField) component;
                        textField.setText("");
                    } else if (component instanceof JDateChooser) {
                        JDateChooser dateChooser = (JDateChooser) component;
                        dateChooser.setDate(null);
                    }
                }
            }
        }
    });

    fieldPanel.add(removeButton);
    filterPanel.add(fieldPanel);
    setSize(getWidth(), getHeight() + 35);
    filterPanel.revalidate();
    filterPanel.repaint();
  }

  private String buildQuery() {
   StringBuilder queryBuilder = new StringBuilder();
   SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
   boolean firstCondition = true;

   for (int i = 0; i < filterPanels.size(); i++) {
      JPanel filterPanel = filterPanels.get(i);
      JPanel checkBoxPanel = (JPanel) filterPanel.getParent().getComponent(0);
      JCheckBox checkBox = (JCheckBox) checkBoxPanel.getComponent(0);
      if (checkBox.isSelected()) {
         Component[] components = filterPanel.getComponents();
         StringBuilder condition = new StringBuilder();
         boolean isFirstFieldInPanel = true;
         for (Component component : components) {
            if (component instanceof JPanel) {
               JPanel fieldPanel = (JPanel) component;
               Component[] fields = fieldPanel.getComponents();
               String minValue = null;
               String maxValue = null;
               boolean isRange = false;
               int textFieldCount = 0;
               for (Component field : fields) {
                  if (field instanceof JTextField) {
                     textFieldCount++;
                  }
               }
               for (Component field : fields) {
                  if (field instanceof JTextField) {
                     JTextField textField = (JTextField) field;
                     String text = textField.getText();
                     if (!text.isEmpty()) {
```

```
                        if (textFieldCount == 2 ) {
                            if (minValue == null) {
                                minValue = text;
                            } else {
                                maxValue = text;
                                isRange = true;
                            }
                        } else if (textFieldCount == 1) {
                            minValue = text;
                        }

                    }
                } else if (field instanceof JDateChooser) {
                    JDateChooser dateChooser = (JDateChooser) field;
                    if (dateChooser.getDate() != null) {
                        if (minValue == null) {
                            minValue = dateFormat.format(dateChooser.getDate());
                        } else {
                            maxValue = dateFormat.format(dateChooser.getDate());
                            isRange = true;
                        }
                    }
                } else if (field instanceof JRadioButton) {
                    JRadioButton radioButton = (JRadioButton) field;
                    if (radioButton.isSelected()) {
                        minValue = radioButton.getText();
                    }
                }
            }
            if (minValue != null && maxValue != null) {
                try {
                    double minVal = Double.parseDouble(minValue);
                    double maxVal = Double.parseDouble(maxValue);
                    if (minVal > maxVal) {
                        String temp = minValue;
                        minValue = maxValue;
                        maxValue = temp;
                    }
                } catch (NumberFormatException e) {
                    e.printStackTrace();
                }
            }

            if (minValue != null) {
                if (!isFirstFieldInPanel) {
                    condition.append(" OR ");
                } else {
                    condition.append("(");
                    isFirstFieldInPanel = false;
                }
                if (isRange && !minValue.equals(maxValue)) {
                    condition.append(checkBox.getText()).append(" BETWEEN '").append(minValue).append("'
AND '").append(maxValue).append("'");
                } else {
                    condition.append(checkBox.getText()).append(" = '").append(minValue).append("'");
                }
            }
        }
    }
    if (condition.length() > 0) {
        condition.append(")");
        if (!firstCondition && queryBuilder.indexOf("AND") != queryBuilder.length() - 4) {
            queryBuilder.append(" AND ");
```

```
            }
            queryBuilder.append(condition);
            firstCondition = false;
        }
    }
  }

  return queryBuilder.toString();
}

  public void createFilterFieldsFromConditions(String conditions) {
   String conditionRegex = "\\((.*?)\\)";
   Pattern conditionPattern = Pattern.compile(conditionRegex);
   Matcher conditionMatcher = conditionPattern.matcher(conditions);
   int i = 0;

      while (conditionMatcher.find()) {
         String conditionGroup = conditionMatcher.group(1); //группа
         Pattern fieldPattern = Pattern.compile("^\\s*([^\\s=]+)\\s*");
         Matcher fieldMatcher = fieldPattern.matcher(conditionGroup);

         String[] conditionsArray = conditionGroup.split("\\s*(?i)OR\\s*");
         List<String> cleanedConditions = new ArrayList<>();
         for (String c : conditionsArray) {
            cleanedConditions.add(c.trim()); // условия
         }

            if (fieldMatcher.find()){
               String fieldName = fieldMatcher.group(1); //название

               for (int j = i; j < filterPanels.size(); j++) {
                  JPanel filterPanel = filterPanels.get(j);
                  JPanel checkBoxPanel = (JPanel) filterPanel.getParent().getComponent(0);
                  // включение чекбокса
                  JCheckBox checkBox = (JCheckBox) checkBoxPanel.getComponent(0);
                  if(checkBox.getText().equals(fieldName)){
                     checkBox.setSelected(true);


                    for (int z = 0; z < cleanedConditions.size(); z++) {

                       if(z>=1){
                          JButton addButton = null;
                          for (Component component : checkBoxPanel.getComponents()) {
                             if (component instanceof JButton) {
                                addButton = (JButton) component;
                                break;
                             }
                          }
                          if (addButton != null) addButton.doClick();
                       }

                       Pattern pattern = Pattern.compile("^(.*?)\\s+(BETWEEN|=)\\s+'(.*?)'\\s+AND\\s+'(.*?)'$");
                       Matcher matcher = pattern.matcher(cleanedConditions.get(z));

                       Component component = filterPanel.getComponents()[z];
                       Component[] fields = null;
                       if (component instanceof JPanel) {
                          JPanel fieldPanel = (JPanel) component;
                          fields = fieldPanel.getComponents();
                       }

                       if (matcher.find()) {
```

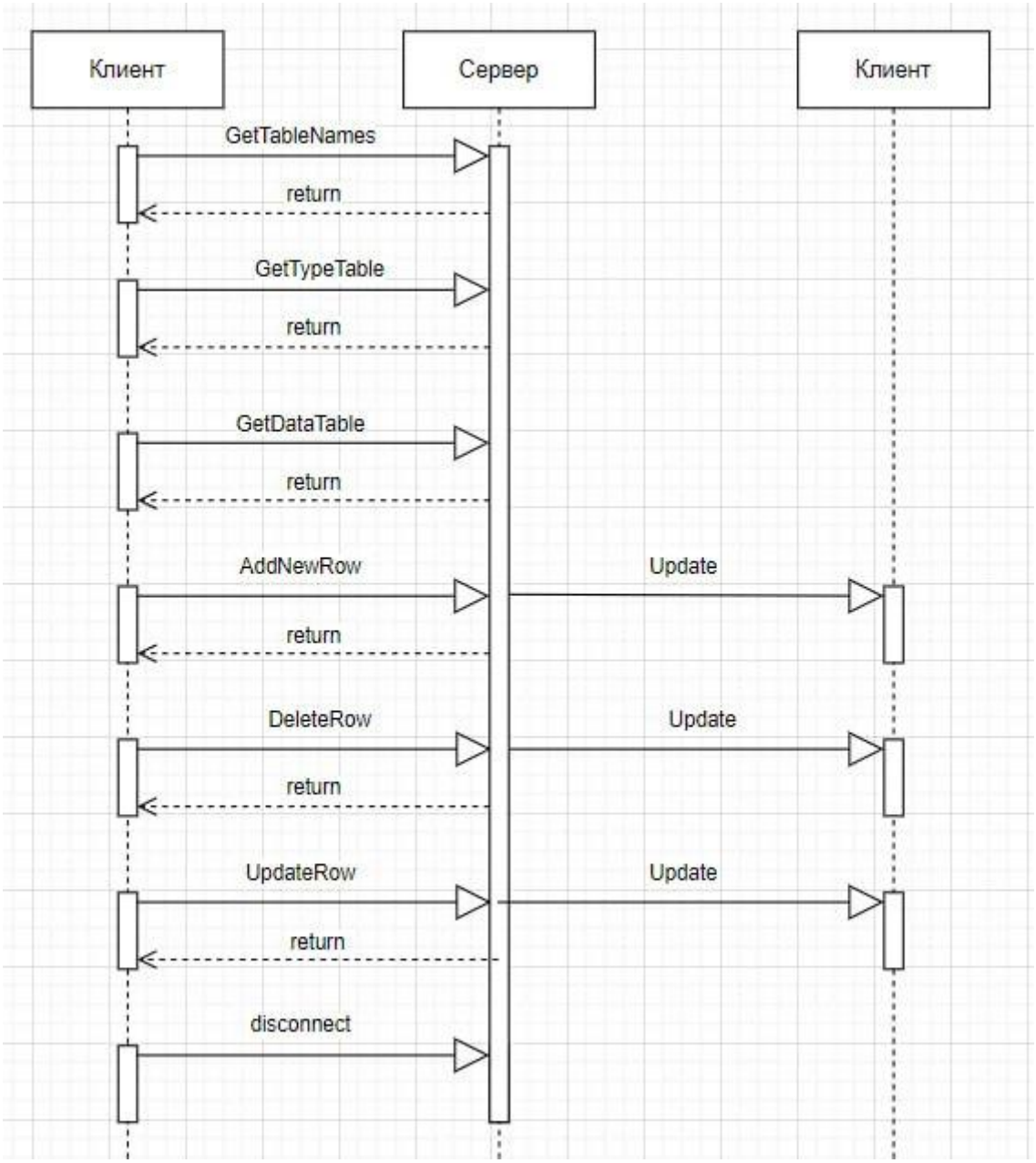
```
                        String minValue = matcher.group(3);
                        String maxValue = matcher.group(4);

                        if (fields[0] instanceof JTextField){
                            JTextField textChooser1 = (JTextField) fields[0];
                            JTextField textChooser2 = (JTextField) fields[1];
                            textChooser1.setText(minValue);
                            textChooser2.setText(maxValue);
                        } else if (fields[0] instanceof JDateChooser) {
                            JDateChooser dateChooser1 = (JDateChooser) fields[0];
                            JDateChooser dateChooser2 = (JDateChooser) fields[1];
                            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
                            try {
                                Date date1 = sdf.parse(minValue);
                                Date date2 = sdf.parse(maxValue);
                                dateChooser1.setDate(date1);
                                dateChooser2.setDate(date2);
                            } catch (ParseException e) {
                                e.printStackTrace();
                            }
                        }

                    } else { // Если не удалось разобрать по шаблону, значит оператор = и только одно
значение
                        pattern = Pattern.compile("^(.*?)\\s+=\\s+'(.*?)'$");
                        matcher = pattern.matcher(cleanedConditions.get(z));

                        if (matcher.find()) {
                            String value = matcher.group(2);

                        if (fields[0] instanceof JTextField){
                            JTextField textChooser = (JTextField) fields[0];
                             textChooser.setText(value);
                        } else if (fields[0] instanceof JRadioButton){
                            JRadioButton yesRadioButton = (JRadioButton) fields[0];
                            JRadioButton noRadioButton = (JRadioButton) fields[1];
                            if (value.equalsIgnoreCase("Yes")) {
                                yesRadioButton.setSelected(true);
                            } else {
                                noRadioButton.setSelected(true);
                            }
                        }

                        }
                    }

                }

                break;
            }
        }


    }

    i++;
  }
 }

}
```

InterfaceCallback.java:

```
/**
 * @author Kseroff
 */
public interface InterfaceCallback {
    void onApply(String query);
    void onCancel();
}
```

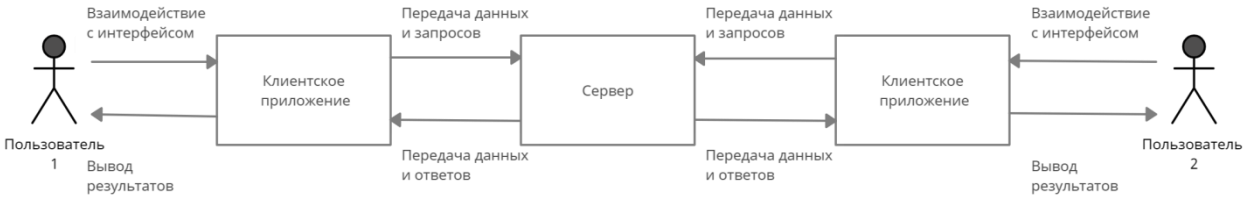
Приложение В. UML-диаграммы

Приложение Б.1. UML-диаграмма протокола взаимодействия клиента и сервера



UML-диаграмма протокола взаимодействия клиента и сервера

Приложение Б.2. UML-диаграмма взаимодействия пользователей



UML-диаграмма кооперации