МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский государственный университет аэрокосмического приборостроения»

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

КУРСОВОЙ ПРОЕКТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Преподаватель |  | 05.07.2023 |  | И.Л. Рохманько |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ  В СОСТАВЕ ПРОФЕССИОНАЛЬНОГО МОДУЛЯ  ПМ.04 Разработка, администрирование и защита баз данных  *код и наименование профессионального модуля* |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | С021к |  | 05.07.2023 |  | В.Д. Панков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на прохождение производственной практики обучающегося по специальности 09.02.07 «Информационные системы и программирование»

1. Фамилия, имя, отчество обучающегося: Панков Василий Дмитриевич
2. Группа: С021К Сроки проведения практики: с « 15 » июня 2023г. по « 05 » июля 2023г.
3. Тема задания: приобретение профессиональных умений и опыта практической деятельности, закрепление сформированных компетенций по профессиональному модулю

ПМ.04 РАЗРАБОТКА, АДМИНИСТРИРОВАНИЕ И ЗАЩИТА БАЗ ДАННЫХ

1. Вопросы, подлежащие изучению:

* Обследование предметной области . Проектирование логической схемы базы данных с использованием современных case-средств;
* Обоснование выбора СУБД. Построение схемы базы данных в конкретной СУБД. Создание хранимых процедур и триггеров;
* Администрирование БД;
* Обеспечение информационной безопасности на уровне базы данных;
* Разработка пользовательского приложения для работы с базой данных;
* Установка базы данных на сервер.

1. Выполнение комплексных работ: Разработка, администрирование и защита базы данных учёта коллекционных записей экспедиционных сборов млекопитающих
2. Содержание отчетной документации:
   1. Отчёт, включающий в себя:

* титульный лист;
* индивидуальное задание;
* материалы о выполнении индивидуального задания (содержание определяется цикловой комиссией);
* список использованных источников.
  1. Аттестационный лист.

1. Срок представления отчета заместителю декана по учебно–производственной работе: «\_05\_\_»\_\_\_июля\_\_\_\_\_2022\_\_ г.

Руководитель практики от факультета СПО:

преподаватель 15.06.23 И.Л. Рохманько

должность, уч. степень, звание подпись, дата инициалы, фамилия

СОГЛАСОВАНО

Руководитель практики от организации:

заведующей лабораторией, ведущий научный сотрудник 15.05.23 Н.И. Абрамсон

должность подпись, дата инициалы, фамилия

Задание принял к исполнению:

Обучающийся

15.06.23 В.Д. Панков

дата подпись инициалы, фамилия

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc139208019)

[1 Анализ предметной области 5](#_Toc139208020)

[1.1 Общая характеристика объекта исследования 5](#_Toc139208021)

[1.2 Постановка и развернутое описание задачи 6](#_Toc139208022)

[1.3 Средства решения 7](#_Toc139208023)

[2 Разработка и реализация проекта базы данных 12](#_Toc139208024)

[2.1 Диаграмма прецедентов 12](#_Toc139208025)

[2.2 Определение логической структуры реляционной базы данных 13](#_Toc139208026)

[2.3 Разработанные хранимые процедуры и триггеры 18](#_Toc139208027)

[2.4 Администрирование 20](#_Toc139208028)

[2.5 Реализация защиты базы данных 29](#_Toc139208029)

[2.6 Результат выполнения процедур 30](#_Toc139208030)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 31](#_Toc139208031)

[ПРИЛОЖЕНИЕ А Созданные таблицы БД 33](#_Toc139208032)

[ПРИЛОЖЕНИЕ Б Разработанные хранимые процедуры и триггеры 53](#_Toc139208033)

# ВВЕДЕНИЕ

Информационная система — это совокупность программных и аппаратных средств, предназначенных для сбора, хранения, обработки, анализа и распространения информации в организации или предприятии.

Существует множество лабораторией во всём мире, которые занимаются различными исследованиями с животными. Данную производственную практику я проходил в «Лаборатории геномики и палеогеномики» при «Государственном зоологическом институте Российской Федерации». Данная лаборатория занимается изучением геномов существующих и исчезнувших видов млекопитающих.

Целью моей работы является проектирование информационной системы, которая облегчит процесс учета коллекционных записей экспедиционных сборов млекопитающих, а также обеспечит возможность анализа полученной информации для научных исследований в данной области.

Для достижения поставленной цели необходимо решить следующие задачи:

* изучить методы и технологии учета и хранения информации о коллекционных записях экспедиционных сборов млекопитающих;
* определить функциональные требования к информационной системе;
* разработать структуру базы данных для хранения информации о коллекционных записях;
* разработать интерфейс для ввода и обработки данных;
* реализовать информационную систему на основе выбранных методов и технологий;

# Анализ предметной области

## Общая характеристика объекта исследования

В данной задаче практики предметной областью является учёт коллекционных записей экспедиционных сборов млекопитающих, а в качестве внешних пользователей могут выступать:

* заведующий лабораторией – лицо, которое может редактировать информацию о сотрудниках, редактировать топологию, а также выдавать доступ новым сотрудникам и менять им пароли;
* научный сотрудник – лицо, будет иметь доступ к просмотру всех коллекционных записей, имеет возможности редактирование и добавление записей и отправку запросов на их удаление, также пользователь может сменить свой пароль.
* гость – группа, пользователей, которая может только просматривать данные.

Научные экспедиции по изучению млекопитающих проходят в различных уголках мира, и собранные в ходе них образцы требуют качественного учёта и обработки. В информационной системе будут храниться данные о месте, времени сбора, полного названия млекопитающего, которое будет включать отряд, семейство, род и вид, его генетические характеристики.

Также некоторые экземпляры могут передавать в другие лаборатории или музеи, для их показа или исследования и иметь там свой уникальный номер.

В самой лаборатории используются собственная идентификация элементов, имеются комментарии к месту сбора и к самой записи.

## Постановка и развернутое описание задачи

Информационная система должна обеспечивать следующие функции:

* ввод, просмотр редактирование информацию о записи, авторе сборного материала, дате сбора, месте сбора, отряде, семействе, роде виде, полу, возрасту и выделена ли RNA;
* выполнять запросы для автоматизации (просмотр по месту сбора, фильтрация по множеству параметров: по автору сборного материала, промежутке сбора, месту сбора, по таксономии, полу, возрасту и выделена ли RNA;);
* возможность хранение ссылок на файлы c полногеномными данными;
* обеспечение быстрого добавления записей (указание точки на карте, выбор сборщика из списка, генерация идентификатора);
* авторизация пользователей и сохранение информации об внесённых ими изменений;
* возможность редактирования древовидной структуры таксономии.

Входные данные для проектирования информационной системы:

* информация о сборе: место сбора, сборщик материала, дата сбора, RNA, внутренний номер в лаборатории, отряд, семейство, род, вид и пол млекопитающего данных сбора.

Выходные данные: в качестве выходных данных будут результаты выполнения запросов.

Требования, предъявляемые к разрабатываемой программе:

Программа должна быть надёжна, простота в использовании и обладать достаточно высоким быстродействием.

Пользовательский интерфейс должен быть интуитивно понятен любому пользователю. В программе должна быть предусмотрена возможность корректировки базы данных.

## Средства решения

Для решения поставленной задачи были выбраны следующие средства:

Dart – объектно-ориентированный язык программированный, разработанный Google. Используется для создания быстрых и эффективных приложений на разных платформах. Одной из важных особенностей данного языка является его способность к компиляции в эффективный нативный код. Это позволяет создавать быстрые и производительные приложения, которые могут выполняться без промежуточной интерпретации. [3]

Material You – это эволюция концепции Material Design, разработанная Google. Он представляет собой дизайн-язык, который ставит на передний план персонализацию и индивидуальность пользователей.

Основные преимущества Material You:

1. персонализация: Material You позволяет пользователям настраивать визуальный стиль своих устройств и приложений, включая цвета, шрифты и стили. Это создает более индивидуальный и удовлетворяющий опыт использования.
2. адаптивность: Material You обеспечивает адаптивный интерфейс, который подстраивается под различные размеры экранов и контексты использования. Приложения выглядят и работают хорошо на разных устройствах.
3. системные возможности: Material You интегрируется с функциями операционной системы и платформы, позволяя использовать новые возможности и API для создания более глубокого пользовательского опыта.
4. совместимость с Material Design: Material You соответствует принципам Material Design, обеспечивая согласованный визуальный стиль и поведение приложений на разных устройствах и платформах.
5. поддержка разработчиков: Material You получил широкую поддержку разработчиков и сообщества. Google предоставляет ресурсы, инструменты и руководства для помощи разработчикам внедрить Material You в свои приложения. [7]

Flutter – это открытый кросс-платформенный фреймворк разработки мобильных приложений, созданный Google. Он позволяет создавать высокопроизводительные приложения для iOS, Android, веб и настольных компьютеров с использованием общего кода на языке программирования Dart. Flutter отличается своим собственным движком рендеринга, обеспечивающим быструю и плавную визуализацию. Он предлагает готовые виджеты и инструменты для удобной разработки пользовательского интерфейса. Благодаря функции горячей перезагрузки, изменения в коде отображаются мгновенно, что ускоряет разработку и упрощает тестирование приложений. Flutter также поддерживает пользовательские анимации, интеграцию с нативными функциями устройства и управление состоянием приложения.

Основные принципы Flutter:

* кроссплатформенность: Flutter позволяет разрабатывать приложения для разных платформ, таких как iOS, Android, веб и настольные компьютеры, с использованием общего кода. Это позволяет сократить время разработки и упростить поддержку приложений на разных платформах;
* горячая перезагрузка: Горячая перезагрузка (Hot Reload) в Flutter позволяет мгновенно видеть изменения в коде и мгновенно обновлять интерфейс приложения без перезапуска. Это значительно ускоряет процесс разработки и позволяет разработчикам быстро итерироваться над интерфейсом и функциональностью приложения;
* однородность интерфейса: Все элементы интерфейса в Flutter являются виджетами, включая даже текст, отступы и цвета. Это создает единообразие во всем интерфейсе приложения и облегчает его масштабирование, адаптацию и изменение визуального стиля;
* производительность: Flutter обеспечивает высокую производительность и плавность работы приложений благодаря использованию собственного движка рендеринга. Он позволяет создавать быстрые и отзывчивые пользовательские интерфейсы, а также оптимизирует использование ресурсов устройства;
* открытость и активное сообщество: Flutter является открытым проектом, и у него активное сообщество разработчиков. Это обеспечивает доступ к большому количеству инструментов, библиотек и пакетов, которые расширяют возможности разработки и помогают решать разнообразные задачи;
* Material You дизайн: Flutter тесно интегрирован с Material You дизайном, стандартным дизайн-языком Google. Это позволяет разработчикам создавать приложения с современным и стильным внешним видом, соответствующим рекомендациям Google по пользовательскому опыту. [4]

Syncfusion DataGrid for Flutter – гибкий компонент пользовательского интерфейса для отображения и редактирования табличных данных в приложениях Flutter. [5]

PostgreSQL– это мощная, открытая реляционная система управления базами данных. PostgreSQL предлагает расширенный набор функций и возможностей, обеспечивая надежное хранение и эффективное управление данными.

Основные характеристики PostgreSQL:

* реляционная модель данных: PostgreSQL основан на реляционной модели данных, что позволяет организовывать данные в таблицы с определенными структурами и связями. Это обеспечивает структурированное хранение информации и возможность выполнения сложных запросов и аналитических операций;
* масштабируемость и производительность: PostgreSQL способен обрабатывать большие объемы данных и обеспечивать высокую производительность при выполнении запросов. Он поддерживает параллельную обработку, индексы, оптимизацию запросов и другие механизмы для оптимизации производительности;
* поддержка стандартов SQL: PostgreSQL полностью совместим с ANSI SQL и поддерживает широкий набор стандартных операторов, функций и запросов. Это облегчает разработку и переносимость приложений, использующих PostgreSQL;
* расширяемость и гибкость: PostgreSQL предлагает механизмы для создания пользовательских типов данных, функций и расширений. Это позволяет разработчикам расширять функциональность СУБД и адаптировать ее под уникальные требования проекта;
* транзакционная безопасность: PostgreSQL обеспечивает транзакционную безопасность, поддерживая ACID-свойства (атомарность, согласованность, изолированность, долговечность). Это гарантирует целостность данных и защиту от потери информации в случае сбоев или ошибок;
* многоязыковая поддержка: PostgreSQL поддерживает множество языков программирования, включая Python, Java, C/C++, Ruby и многие другие. Это облегчает интеграцию СУБД с различными приложениями и разработку на разных платформах; [2]

RESTful API (Representational State Transfer) – это архитектурный стиль для проектирования и взаимодействия сетевых приложений, основанный на принципах REST. Он обеспечивает стандартизированный способ обмена данными между клиентом и сервером с использованием HTTP-протокола.

PostgREST – это программное обеспечение с открытым исходным кодом, которое предоставляет простой и удобный RESTful интерфейс для работы с базой данных PostgreSQL. Он позволяет вам взаимодействовать с базой данных через стандартные HTTP-методы, такие как GET, POST, PUT и DELETE, и автоматически генерирует API-эндпоинты для каждой таблицы в базе данных. PostgREST обеспечивает функции фильтрации, сортировки, пагинации и аутентификации данных, а также поддерживает вложенные запросы и пользовательские функции. Использование PostgREST значительно упрощает разработку и создание API на основе PostgreSQL, обеспечивая гибкость и масштабируемость веб-приложений и микросервисов. [2]

Nominatim (от латинского «by name») – это инструмент для поиска данных OSM по имени и адресу и для создания синтетических адресов точек OSM (обратное геокодирование). [9]

# Разработка и реализация проекта базы данных

## Диаграмма прецедентов

Диаграмма вариантов использования – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. Диаграмма use-case для проекта представлена на рисунке 3.

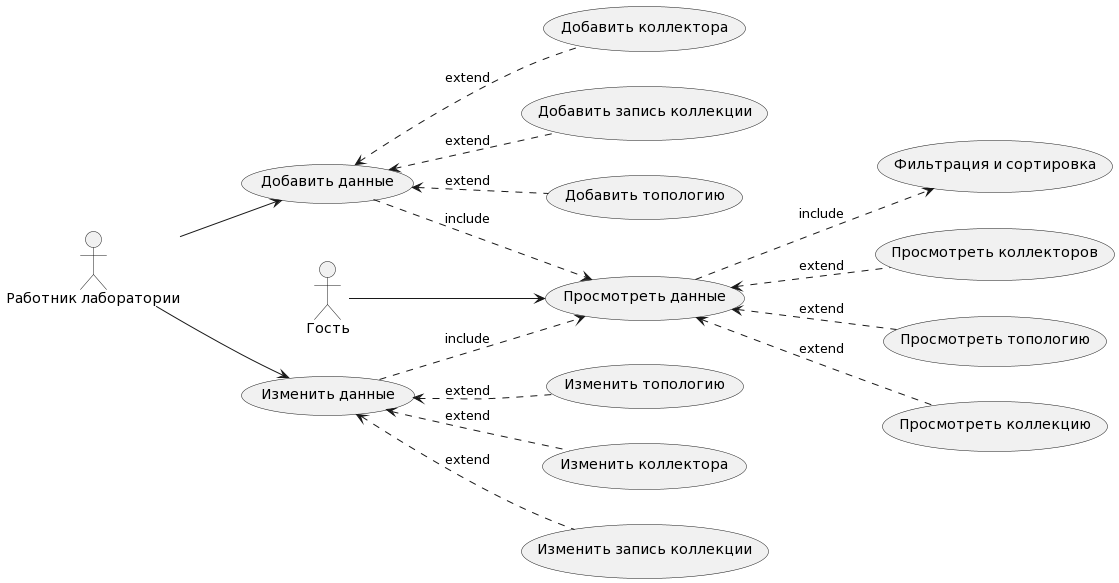


Рисунок 3 – Таблица users

## Определение логической структуры реляционной базы данных

Даталогическая структура реляционной базы данных определяется совокупностью логически связанных реляционных таблиц.

Логические связи соответствуют структурным связям между объектами в концептуальной модели, каждый объект в логической модели отображается соответствующей реляционной таблицей.

Связи между таблицами осуществляются посредством общих атрибутов. Реляционная модель проектируемой базы данных.

Таблица 1 содержит информацию о возрастах.

Таблица 1 – Описание логической структуры таблицы «age»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| - | name | Символьный (переменный) | 20 | - |

Таблица 2 содержит информацию о возрастах.

Таблица 2 – Описание логической структуры таблицы «collection»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| PK | id | Числовой | - | NOT NULL |
| - | CatalogueNumber | Текстовый | - | - |
| - | collect\_id | Текстовый | - | - |
| FK | kind\_id | Числовой | - | NOT NULL |
| FK | subregion\_id | Числовой | - | NOT NULL |
| - | gen\_bank\_id | Символьный (переменный) | 20 | - |
| - | point | География (точка) | - | - |
| FK | vouch\_inst\_id | Числовой | - | - |
| Продолжение таблицы 2 | | | | |
| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| - | rna | Логический | - | DEFAULT false NOT NULL |
| FK | sex\_id | Числовой | - | - |
| FK | age\_id | Числовой | - | - |
| - | day | Числовой | - | - |
| - | month | Числовой | - | - |
| - | year | Числовой | - | NOT NULL |
| - | comment | Текстовый | - | - |
| - | geo\_comment | Текстовый | - | - |
| - | file\_url | Текстовый | - | - |

Таблица 3 содержит информацию о коллекторах.

Таблица 3 – Описание логической структуры таблицы «collector»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| - | last\_name | Символьный (переменный) | 100 | - |
| - | first\_name | Символьный (переменный) | 100 | - |
| - | second\_name | Символьный (переменный) | 100 | - |

Таблица 4, является смежной таблицей, которая содержит информацию о соотношении коллекторов к коллекции.

Таблица 4 – Описание логической структуры таблицы «collector\_to\_collection»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK, FK | collector\_id | Числовой | - | NOT NULL |
| PK, FK | collection\_id | Числовой | - | NOT NULL |

Таблица 5 содержит информацию о странах.

Таблица 5 – Описание логической структуры таблицы «country»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| - | name | Текстовый | - | NOT NULL |

Таблица 6 содержит информацию о регионах.

Таблица 6 – Описание логической структуры таблицы «region»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| FK | country\_id | Числовой | - | NOT NULL |
| - | name | Текстовый | - |  |

Таблица 7 содержит информацию о субрегионах.

Таблица 7 – Описание логической структуры таблицы «subregion»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| PK | id | Числовой | - | NOT NULL |
| FK | region\_id | Числовой | - | NOT NULL |
| - | name | Текстовый | - |  |

Таблица 8 содержит информацию об отрядах.

Таблица 8 – Описание логической структуры таблицы «order»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| - | name | Текстовый | - |  |

Таблица 9 содержит информацию о семействах.

Таблица 9 – Описание логической структуры таблицы «family»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| FK | order\_id | Числовой | - | NOT NULL |
| - | name | Текстовый | - |  |

Таблица 10 содержит информацию о родах.

Таблица 10 – Описание логической структуры таблицы «genus»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| FK | family\_id | Числовой | - | NOT NULL |
| - | name | Текстовый | - |  |

Таблица 11 содержит информацию о видах.

Таблица 11 – Описание логической структуры таблицы «kind»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| FK | genus\_id | Числовой | - | NOT NULL |
| - | name | Текстовый | - |  |

Таблица 12 содержит информацию о полах.

Таблица 12 – Описание логической структуры таблицы «sex»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| - | name | Символьный (переменный) | 40 | NOT NULL |

Таблица 13 содержит информацию о ваучерных институтах.

Таблица 13 – Описание логической структуры таблицы «voucher\_institute»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | id | Числовой | - | NOT NULL |
| - | name | Текстовый | - | NOT NULL |

На рисунке 1 показана схема связей данных.

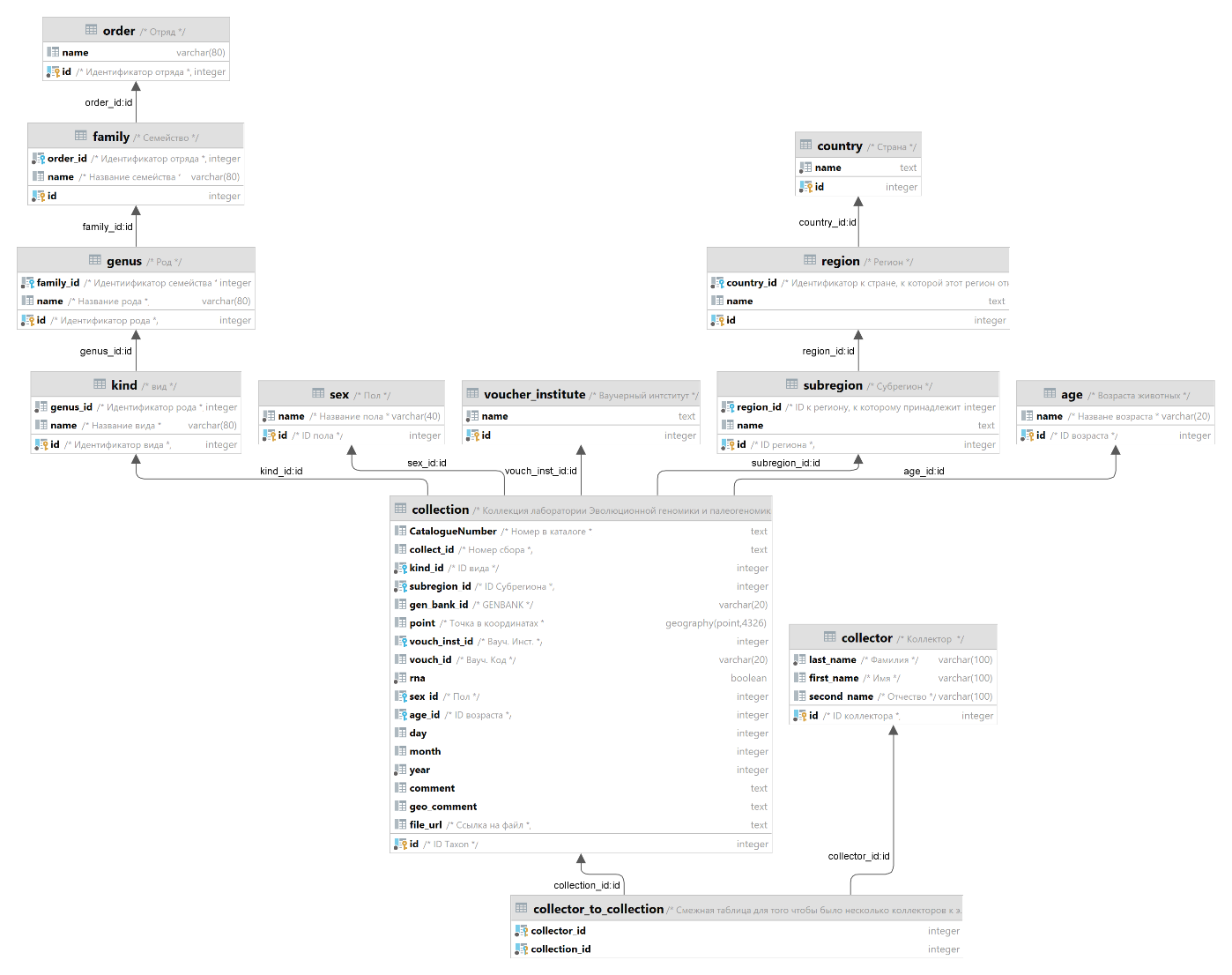


Рисунок 1 – Схема связей данных в СУБД PostgreSQL

## Разработанные хранимые процедуры и триггеры

В таблице 15 представлен список хранимых процедур и их описание.

Таблица 15 – Список хранимых процедур

| Название | Описание |
| --- | --- |
| get\_age\_id | Возвращает идентификатор возраста по его названию |
| get\_sex\_id | Возвращает идентификатор пола по его названию |
| get\_order\_id | Возвращает идентификатор порядка по его названию |
| get\_family\_id | Возвращает идентификатор семейства по его названию и порядку |
| get\_genus\_id | Возвращает идентификатор рода по его названию и семейству |
| get\_kind\_id | Возвращает идентификатор вида по его названию и роду |
| get\_country\_id | Возвращает идентификатор страны по её названию |
| get\_region\_id | Возвращает идентификатор региона по его названию и стране |
| get\_subregion\_id | Возвращает идентификатор подрегиона по его названию и региону |
| get\_vouch\_inst\_id | Возвращает идентификатор института-владельца ваучера по его названию |
| get\_collector\_id | Возвращает идентификатор коллекционера по его фамилии, имени и отчеству |
| add\_collection | Добавляет запись о коллекции в базу данных |
| remove\_collection\_by\_id | Удаляет запись о коллекции по её идентификатору |
| update\_collection\_by\_id | Обновляет запись о коллекции по её идентификатору |
| auth.login | Проверяет логин и пароль, возвращает JWT-токен в случае успешной аутентификации |
| auth.get\_user\_info | Возвращает информацию о пользователе (логин, URL аватара, роль) |
| auth.user\_role | Получение роли пользователя по его логину и паролю |
| check\_auth | Проверка авторизации пользователя |
| add\_topology | Добавление топологии, по частичному указанию, отряда, семейства, рода, вида. |

Листинг разработанных хранимых процедур находится в Приложении Б.

В таблице 16 представлен список триггеров и их описание.

Таблица 16 – Список триггеров

| Название | Описание |
| --- | --- |
| ensure\_user\_role\_exists | Проверяет, существует ли указанная роль в базе данных. Если роль не существует, генерирует ошибку. |
| encrypt\_pass | Хеширует пароль перед сохранением в таблице auth.users. |

Листинг разработанных триггеров находится в Приложении Б.

В таблице 17 представлен список представлений и их описание.

Таблица 17 – Список представлений

|  |  |
| --- | --- |
| Название | Описание |
| basic\_view | Соединяет все таблицы из схемы public и показывают коллекцию, в удобном виде для просмотра и анализа данных. |

Листинг разработанных представлений находится в Приложении Б.

## Администрирование

В базе данных были созданы следующие группы пользователей:

* web\_​anon - гость, который может просматривать все таблицы в публичном доступе (scheme public):
  + collection;
  + age;
  + family;
  + sex;
  + country;
  + voucher\_​institute;
  + region;
  + genus;
  + collector\_​to\_​collection;
  + order;
  + collector;
  + subregion;
  + kind;
  + collection.
* lab\_worker - работник лаборатории, имеет те же привелегии, что и web\_anon, но также может добавлять и изменять любые таблицы в публичном доступе (scheme public), а также функции добавления и изменения записей в таблице collection;
* head\_​lab - заведующий лабораторией, имеет те же привелегии, что и lab\_​worker, но также может выполнять все функции в публичном доступе (scheme public) и все процедуры в авторизации (scheme auth), что позволяет ему:
  + добавлять пользователя;
  + удалять пользователя;
  + изменять пользователя.
* database\_​admin - администратор базы данных обладает всеми привилегиями на обе схемы (auth и public).
* Дополнительная роль аутентификатор (authenticator), у этой роли есть права всех пользователей выше, благодаря этому модуль postgrest реализовывает распределение ролями.

В приложении Г, представлен код выдачи прав всем пользователям выше.

### Авторизация

Для авторизации была реализована отдельная схема auth.

Таблица 14 содержит информацию о пользователях.

Таблица 14 – Описание логической структуры таблицы «users»

| Признак ключа | Имя поля | Тип данных поля | Длина | Ограничения |
| --- | --- | --- | --- | --- |
| PK | login | Текстовый | - | NOT NULL |
| - | pass | Текстовый | <512 | NOT NULL |
| - | role | name(Имя роли или пользователя) | <512 | NOT NULL |
| - | avatar | Текстовый | - | - |

Хранимые процедуры, функции и триггеры данной схемы приведены в соответствующем разделе «Разработанные хранимые процедуры, триггеры и представления», а их код приведён в приложении Б.

На рисунке 2, представлено демонстрируются тестовые данные в таблице user и вид хешированное пароля.

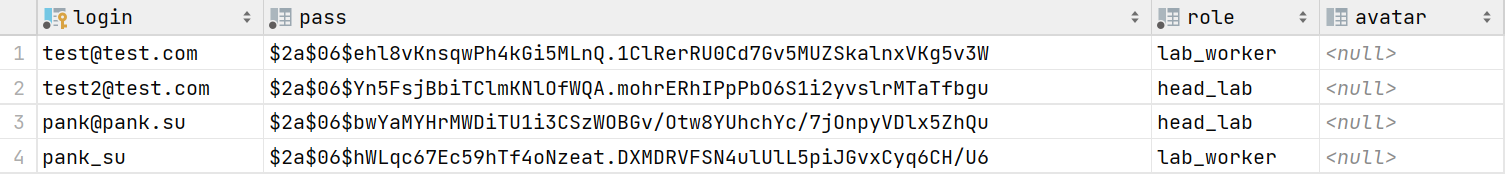


Рисунок 2 – Таблица users

Авторизация пользователя происходит через RESTful API, созданного с помощью приложения PostgREST, конфигурационный файл приведён в приложении Д.

### Демонстрация работы

В рамках демонстрации ограничений возможностей пользователей разных групп будет произведено наглядное представление с использованием HTTP запросов к серверу и соответствующих ответов. Кроме того, будет представлена декодировка HTTP запросов в SQL для полного понимания процесса.

Демонстрация со стороны анонимного пользователя – роль web\_anon.

Получение записи по id=1 в таблице «collection».

HTTP запрос:

GET http://localhost:3000/collection?id=eq.1

Аналогичный SQL запрос:

SELECT \* FROM collection WHERE id = 1;

Ответ:

[

{

"id": 1,

"CatalogueNumber": "ZIN-TER-M-55",

"collect\_id": "1",

"kind\_id": 1,

"subregion\_id": 1,

"gen\_bank\_id": null,

"point": "0101000020E6100000CACDE459D130554008D8BBFD41264940",

"vouch\_inst\_id": 1,

"vouch\_id": "91130",

"rna": false,

"sex\_id": 2,

"age\_id": 3,

"day": 12,

"month": 6,

"year": 2005,

"comment": "По сиквенсам со1 и цитохрома - красно-серая, первоначально определена как красная JF713496",

"geo\_comment": "Алтай, Республика, Усть-Коксинский р-он, Банное, село, р. Колчулу"

}

]

Попытка добавить запись в таблицу «age».

HTTP запрос:

POST http://localhost:3000/age

Content-Type: application/json

{"name": "test age"}

Аналогичный SQL запрос:

INSERT INTO age VALUES ('test age')

Ответ:

{

"code": "42501",

"details": null,

"hint": null,

"message": "нет доступа к таблице age"

}

Демонстрация, что таблица age доступна:

HTTP запрос:

GET http://localhost:3000/age

Аналогичный SQL запрос:

SELECT \* FROM age

Ответ:

[

{

"id": 0,

"name": "Unknown"

},

{

"id": 2,

"name": "subadult"

},

{

"id": 3,

"name": "adult"

},

{

"id": 1,

"name": "juvenile"

},

{

"id": 4,

"name": "subadult or adult"

},

{

"id": 19,

"name": "test"

}

]

Демонстрация со стороны работника лаборатории – роль lab\_worker.

Авторизация:

HTTP запрос:

POST http://localhost:3000/rpc/login

Content-Type: application/json

{"email": "test@test.com", "pass":"test"}

Аналогичный SQL запрос:

SELECT login(‘test@test.com’, ‘test’)

Ответ, при верном пароле:

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoibGFiX3dvcmtlciIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsImV4cCI6MTY4Mjc3MjEyN30.cI4CN4iIeOPw-jsMLSLF3NrtDPSYsCfm9M1EPbaqHbw"

}

Ответ, при неверном пароле или логине:

{

"code": "28P01",

"details": null,

"hint": null,

"message": "invalid user or password"

}

Добавление записи в коллекцию:

HTTP запрос:

POST http://localhost:3000/rpc/add\_collection

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoibGFiX3dvcmtlciIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsImV4cCI6MTY4Mjk1MzU3MX0.IKV4EPcylqRBEoumGboCXfegJJ9bMHsyXngEy1I84I4

Content-Type: application/json

{ "catalog\_number": "test", "collect\_id": "test", "order" : "test", "family" : "test",

"genus" : "test", "kind" : "test", "age" : "test", "sex" : "test", "vauch\_inst" : "test",

"vauch\_id" : "test", "point" : "POINT(0 0)", "country" : "test", "region" : "test", "subregion" : "test",

"geocomment": "test", "date\_collect" : "2004-4-4", "comment" : "Это тестовая запись",

"collectors" : "{{\"Панков\", \"\", \"\" }, {\"Турсунова\", \"\", \"\"}}"}

Аналогичный SQL запрос:

SELECT add\_collection(‘test’, ‘test’, ‘test’, ‘test’, ‘test’, ’test’, ‘POINT(0 0)’, ‘test’, ‘test’, ‘test’, ‘test’, ‘2004-4-4’, ‘Это тестовая запись’, ‘"{{"Панков", "", "" }, {"Турсунова", "", ""}}’)

Ответ пустой.

Проверка, что запись добавилась:

HTTP запрос:

GET http://localhost:3000/collection?id=eq.6081

Аналогичный SQL запрос:

SELECT \* FROM collection WHERE id = 6081

Ответ:

[

{

"id": 6081,

"CatalogueNumber": "test",

"collect\_id": "test",

"kind\_id": 270,

"subregion\_id": 154,

"gen\_bank\_id": null,

"point": "0101000020E610000000000000000000000000000000000000",

"vouch\_inst\_id": 46,

"vouch\_id": "test",

"rna": false,

"sex\_id": 18,

"age\_id": 19,

"day": 4,

"month": 4,

"year": 2004,

"comment": "Это тестовая запись",

"geo\_comment": "test"

}

]

Попытка удаления записи:

HTTP запрос:

POST http://localhost:3000/rpc/remove\_collection\_by\_id

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoibGFiX3dvcmtlciIsImVtYWlsIjoidGVzdEB0ZXN0LmNvbSIsImV4cCI6MTY4Mjk1NzI1N30.YaqxuxgQeeZPTQBppSYg154FDUPlrTh8WhgDvgEd5sQ

Content-Type: application/json

{"col\_id": 6081}

Аналогичный SQL запрос:

SELECT remove\_collection\_by\_id(6081)

Ответ:

{

"code": "42501",

"details": null,

"hint": null,

"message": "нет доступа к таблице collection"

}

Демонстрация со стороны заведующего лабораторией – роль head\_lab.

Попытка удаления:

HTTP запрос:

POST http://localhost:3000/rpc/remove\_collection\_by\_id

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiaGVhZF9sYWIiLCJlbWFpbCI6InRlc3QyQHRlc3QuY29tIiwiZXhwIjoxNjgyOTU4MjA3fQ.mkzS0XwjVWW\_yook7JUPx3UyypScqrWh9mkKCmWWAN0

Content-Type: application/json

{"col\_id": 6081}

Аналогичный SQL запрос:

SELECT remove\_collection\_by\_id(6081)

Пустой ответ.

Демонстрация, что запись удалена:

HTTP запрос:

GET http://localhost:3000/collection?id=eq.6081

Аналогичный SQL запрос:

SELECT \* FROM collection WHERE id = 6081

Ответ:

[]

## Реализация защиты базы данных

В данном курсовом проекте для защиты базы данных использовались расширение pgcrypto, а именно функции:

* crypt - функция, которая шифрует строку с солью, собственным алгоритмом, который работает быстрее и безопаснее, чем алгоритмы sha-1 и md-5, но при этом основывается на них.

Данная функция поддерживает проверку пароля не раскрывая соль, которая использовалась при генерации, что обеспечивает дополнительную безопасность;

* gen\_​salt - функция, которая генерирует соль и указывает основной алгоритм шифрования данных.

Защиту, также обеспечивала программа PostgREST, описание которого приведено в разделе «Администрирование», так как он не даёт возможности запуска полноценных SQL скриптов на сервере со стороны пользователя, что исключает SQL инъекции и возможность входа на базу данных, при изолировании её на отдельном сервер.

Аналогично написание многих хранимых процедур и триггеров остерегает базу данных от SQL инъекций и оберегает от возможного человеческого фактора, при администрировании базы данных.

## Результат выполнения процедур

Ниже представлены результаты выполнения процедур, исходные данные таблиц представлены в приложении А, а код процедур представлен в приложении Б.

Код запуска процедуры get\_age\_id:

SELECT get\_age\_id('adult');

Результат выполнения процедуры get\_age\_id, представлен на рисунке 4.



Рисунок 4 – Результат выполнения процедуры get\_age\_id

Код запуска процедуры get\_collector\_id:

SELECT get\_collector\_id('Стекольников');

Результат выполнения процедуры get\_collector\_id, представлен на рисунке 5.

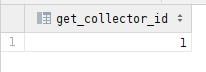


Рисунок 5 – Результат выполнения процедуры get\_collector\_id

Код запуска процедуры get\_country\_id:

SELECT get\_country\_id('Россия');

Результат выполнения процедуры get\_country\_id, представлен на рисунке 6.

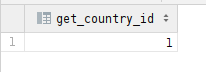


Рисунок 6 – Результат выполнения процедуры get\_country\_id

Код запуска процедуры get\_order\_id:

SELECT get\_order\_id('rodentia');

Результат выполнения процедуры get\_order\_id, представлен на рисунке 7.



Рисунок 7 – Результат выполнения процедуры get\_order\_id

Код запуска процедуры get\_family\_id:

SELECT get\_family\_id('cricetidae', 1);

Результат выполнения процедуры get\_family\_id, представлен на рисунке 8.



Рисунок 8 – Результат выполнения процедуры get\_family\_id

Код запуска процедуры get\_genus\_id:

SELECT get\_genus\_id('clethrionomys', 1);

Результат выполнения процедуры get\_genus\_id, представлен на рисунке 9.



Рисунок 9 – Результат выполнения процедуры get\_genus\_id

Код запуска процедуры get\_kind\_id:

SELECT get\_kind\_id('rufocanus', 1);

Результат выполнения процедуры get\_kind\_id, представлен на рисунке 10.



Рисунок 10 – Результат выполнения процедуры get\_kind\_id

Код запуска процедуры add\_collection:

SELECT add\_collection(vauch\_inst := 'ЗИН', date\_collect := '2023-10-1');

Добавленная запись, с помощью процедуры add\_collection, представлена на рисунке 11.

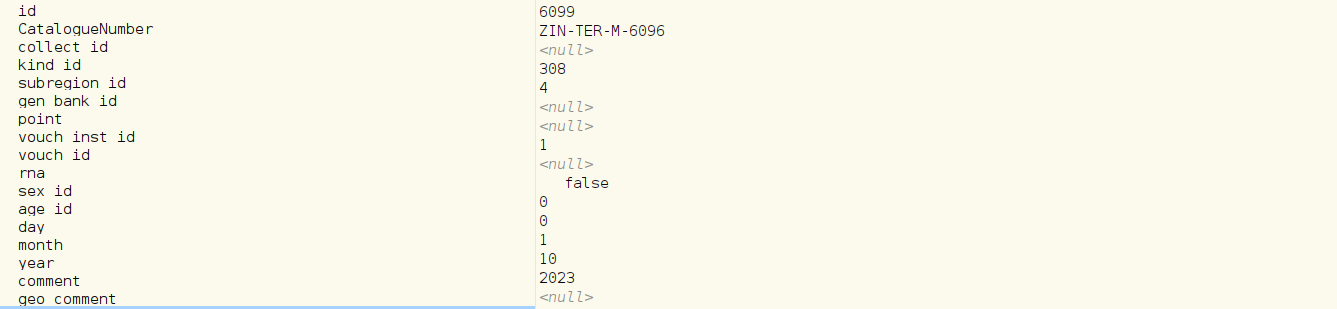


Рисунок 11 – Запись, добавленная процедурой add\_collection

Код запуска процедуры get\_region\_id:

SELECT get\_region\_id('Алтай, Республика', 1);

Результат выполнения процедуры get\_region\_id, представлен на рисунке 12.



Рисунок 12 – Результат выполнения процедуры get\_region\_id

Код запуска процедуры get\_subregion\_id:

SELECT get\_subregion\_id('Юг', 34);

Исходные данные для запроса get\_subregion\_id, представлены на рисунке 13.

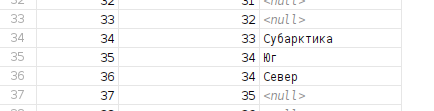


Рисунок 13 – Исходные данные для запроса get\_subregion\_id

Результат выполнения процедуры get\_subregion\_id, представлен на рисунке 14.



Рисунок 14 – Результат выполнения процедуры get\_subregion\_id

Код запуска процедуры get\_sex\_id:

SELECT get\_sex\_id('Мужской');

Результат выполнения процедуры get\_sex\_id, представлен на рисунке 15.



Рисунок 15 – Результат выполнения процедуры get\_sex\_id

Код запуска процедуры get\_vouch\_inst\_id:

SELECT get\_vouch\_inst\_id('ЗИН');

Результат выполнения процедуры get\_vouch\_inst\_id, представлен на рисунке 16.



Рисунок 16 – Результат выполнения процедуры get\_vouch\_inst\_id

Код запуска процедуры add\_topology:

SELECT add\_topology('Vasya', 'Pankov');

Записи, добавленные в таблицы order и family в результате выполнения процедуры add\_topology, представлен на рисунках 17-18.

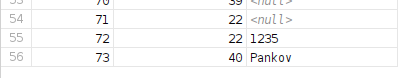


Рисунок 17 – Добавленная запись в таблицу family

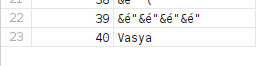


Рисунок 18 – Добавленная запись в таблицу order

# Разработка клиентского приложения

Код клиентского приложения представлен в приложении Д. На рисунках 19-20 представлены скриншоты разработанного клиентского приложения.

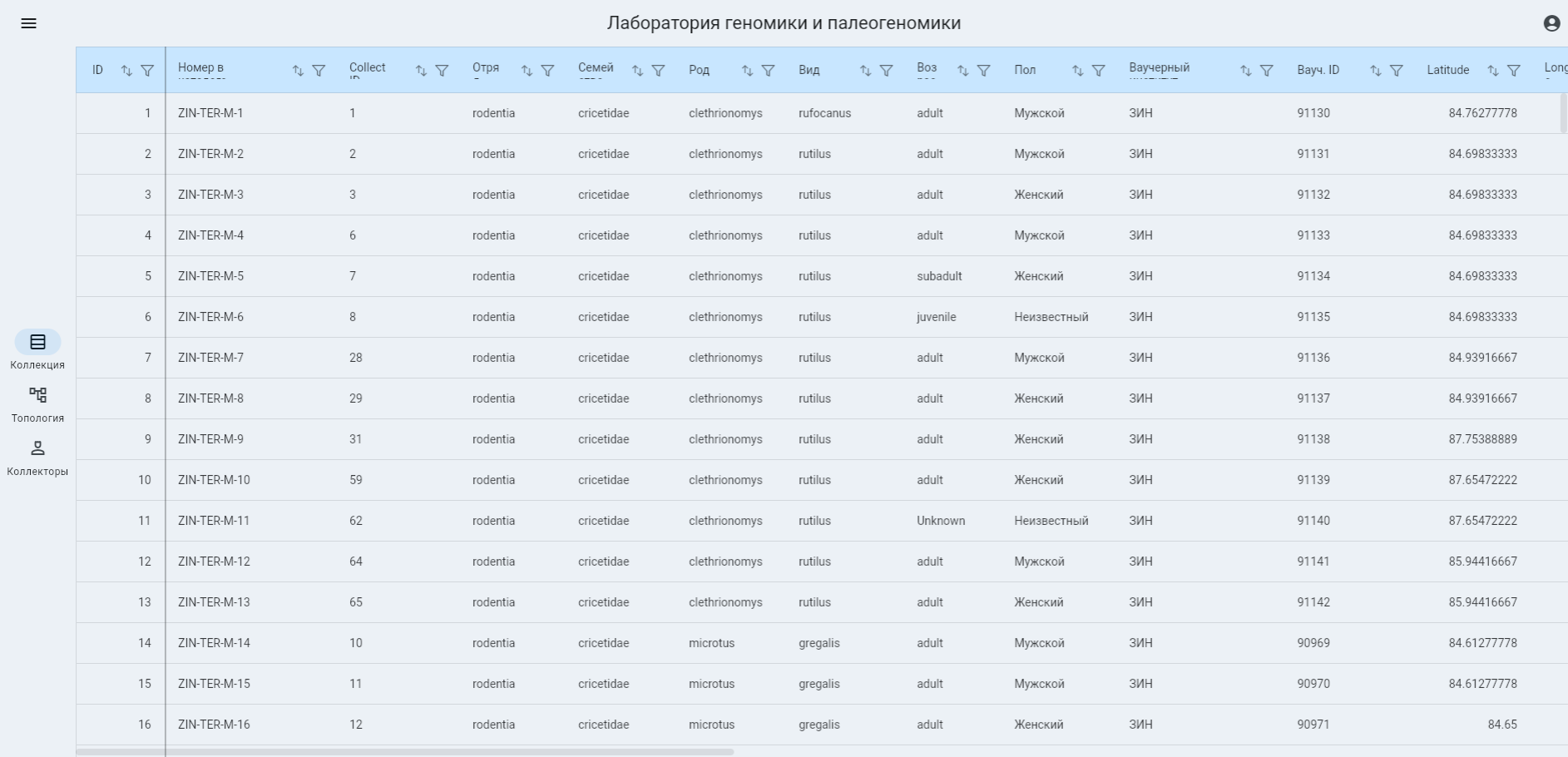


Рисунок 19 – Таблица коллекция

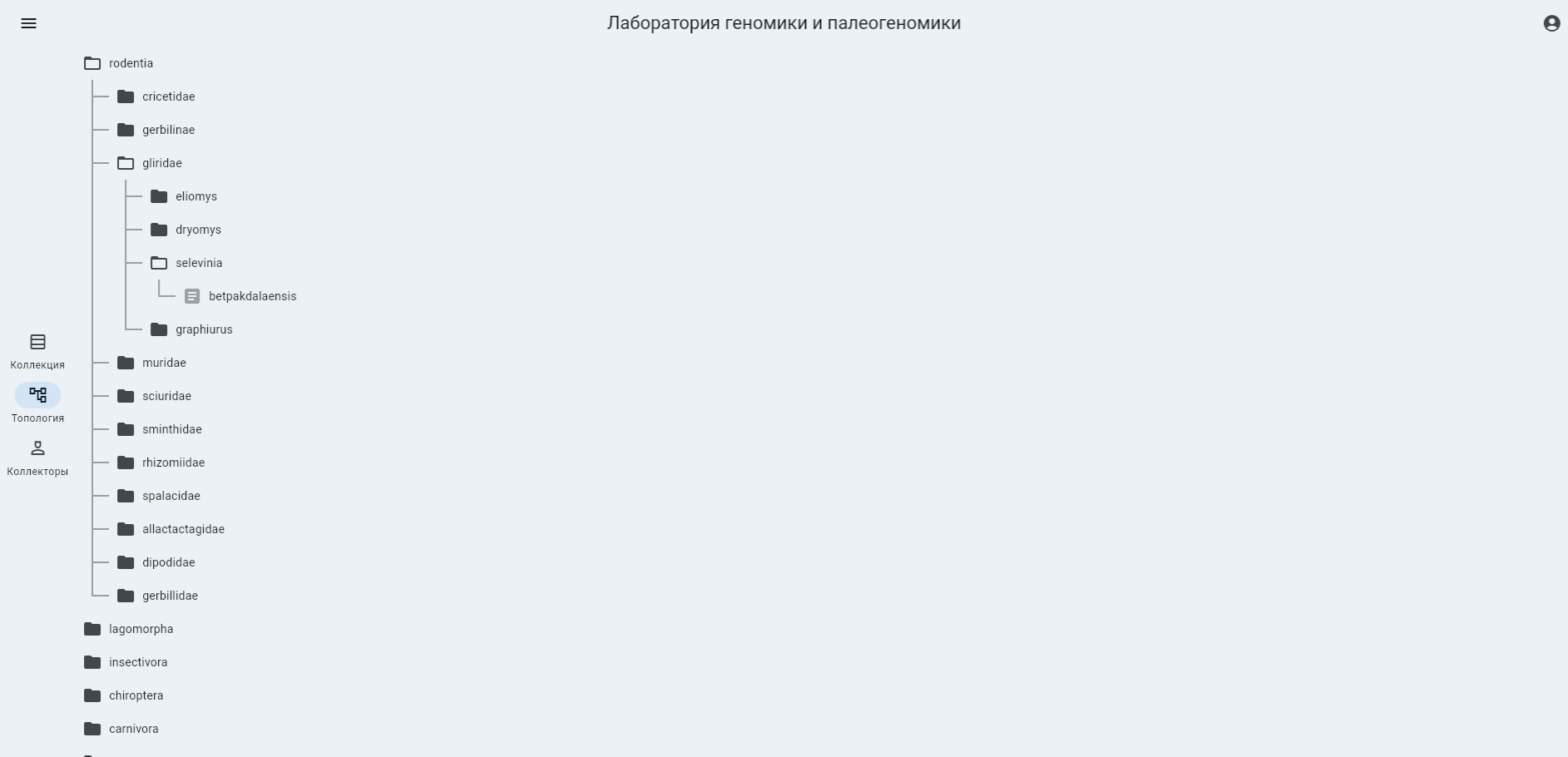


Рисунок 20 – Отображение топологии

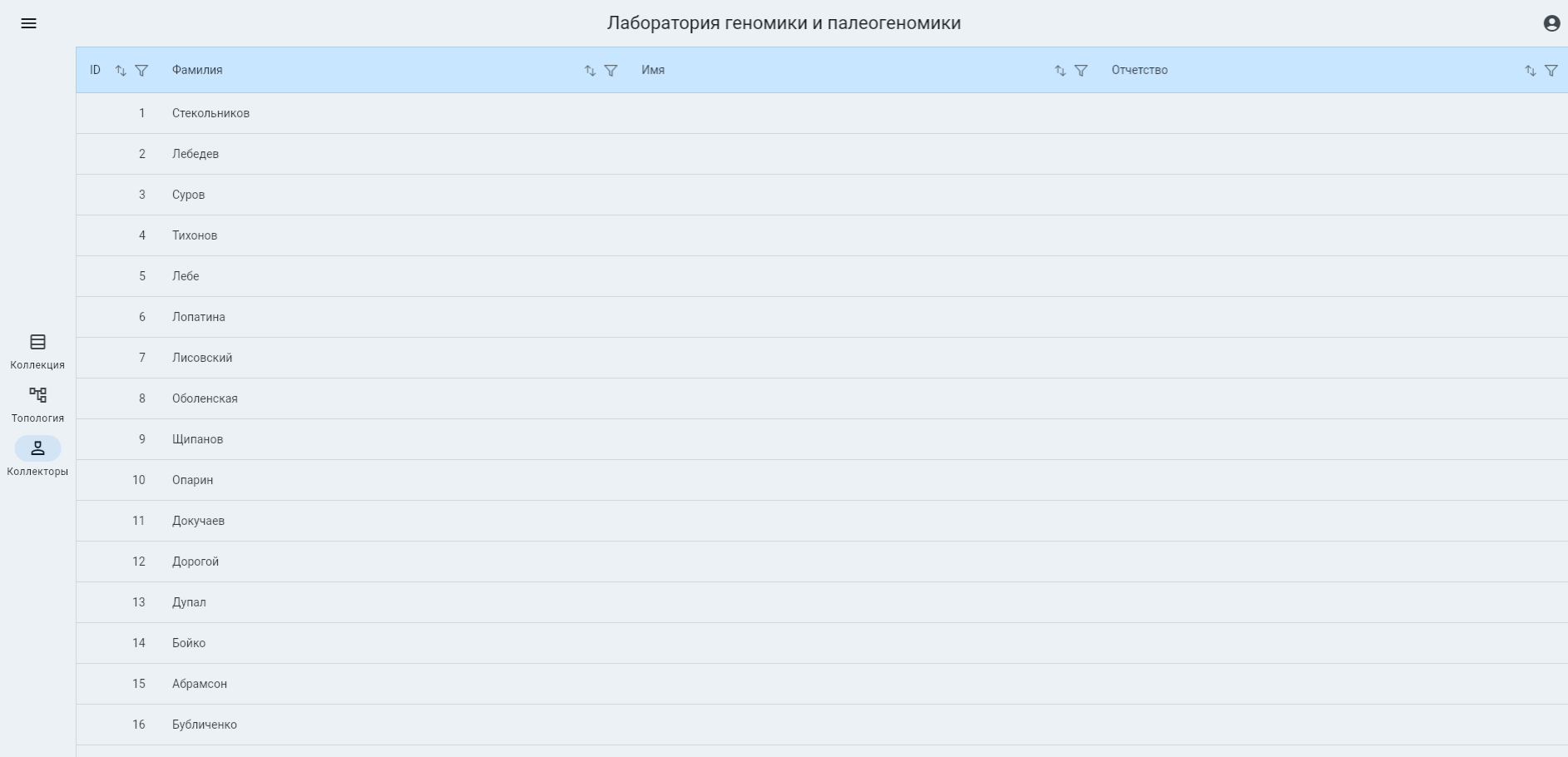


Рисунок 21 – Отображение коллекторов

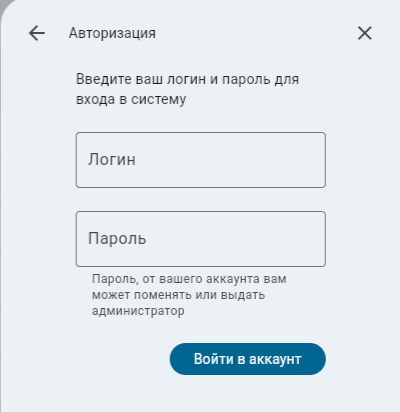


Рисунок 22 – Отображение авторизации

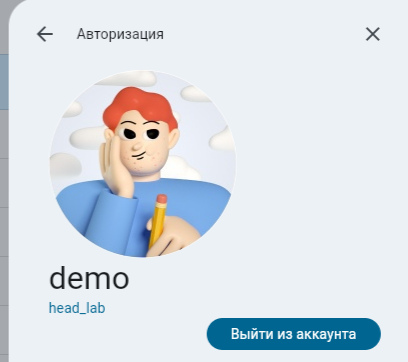


Рисунок 23 – Отображение профиля пользователя

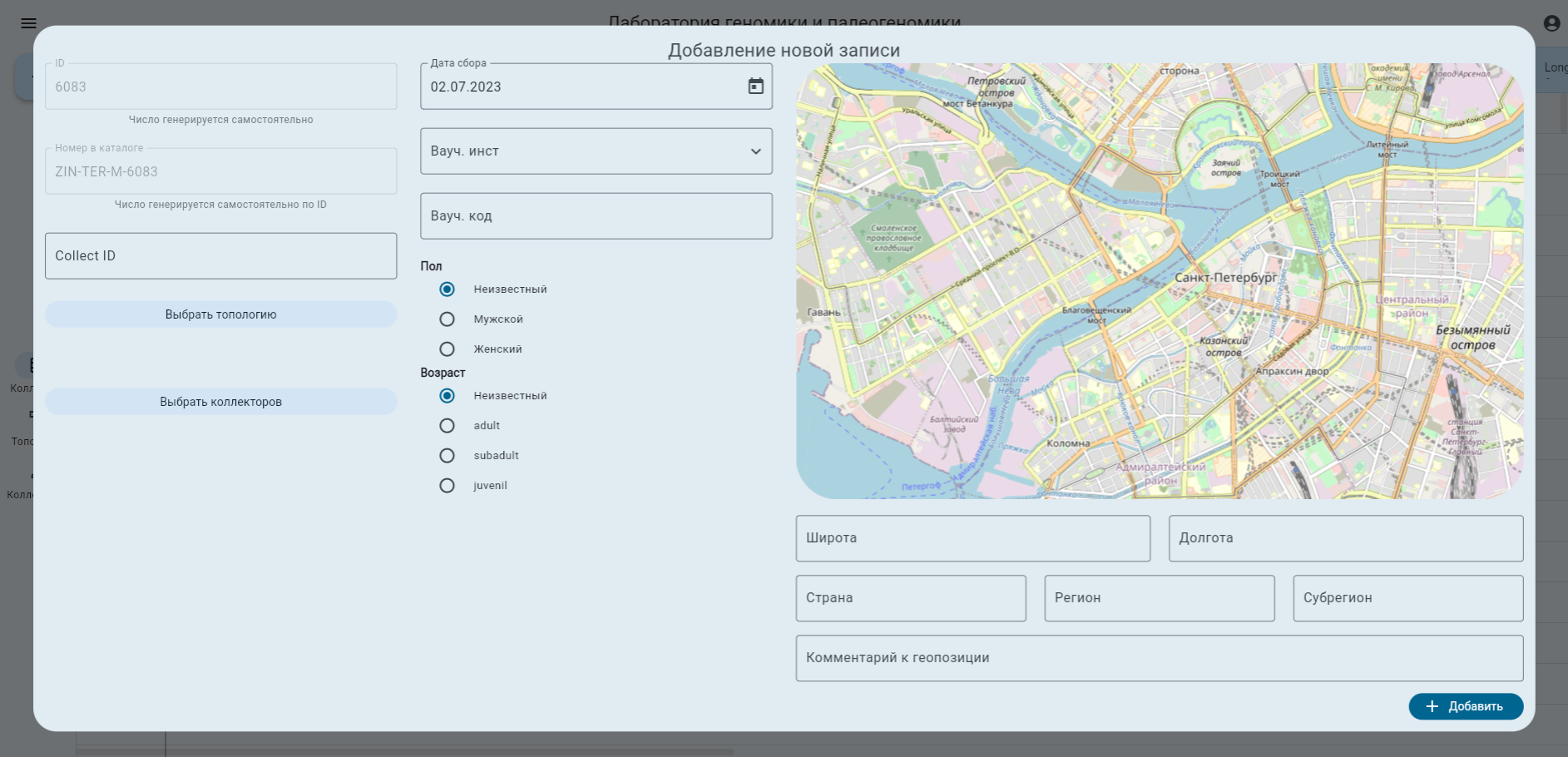


Рисунок 24 – Отображение добавления новой записи в коллекцию

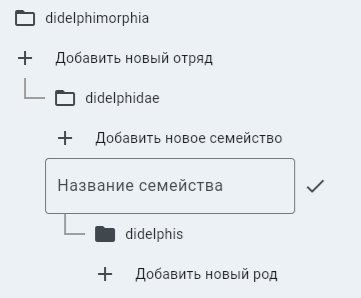


Рисунок 25 – Отображение добавления новой топологии

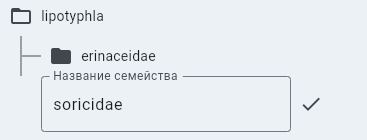


Рисунок 26 – Отображение редактирования топологии

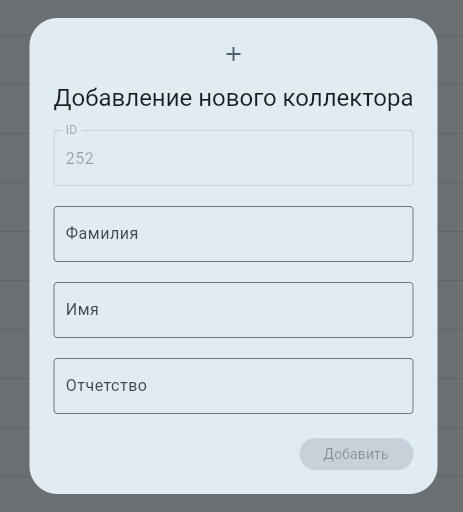


Рисунок 27 – Отображение добавления нового коллектора

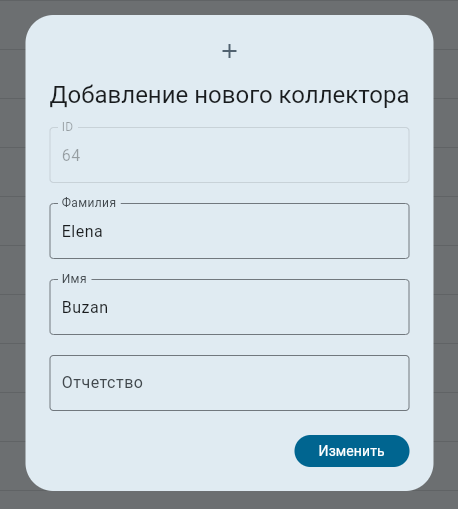


Рисунок 28 – Отображение изменение коллектора

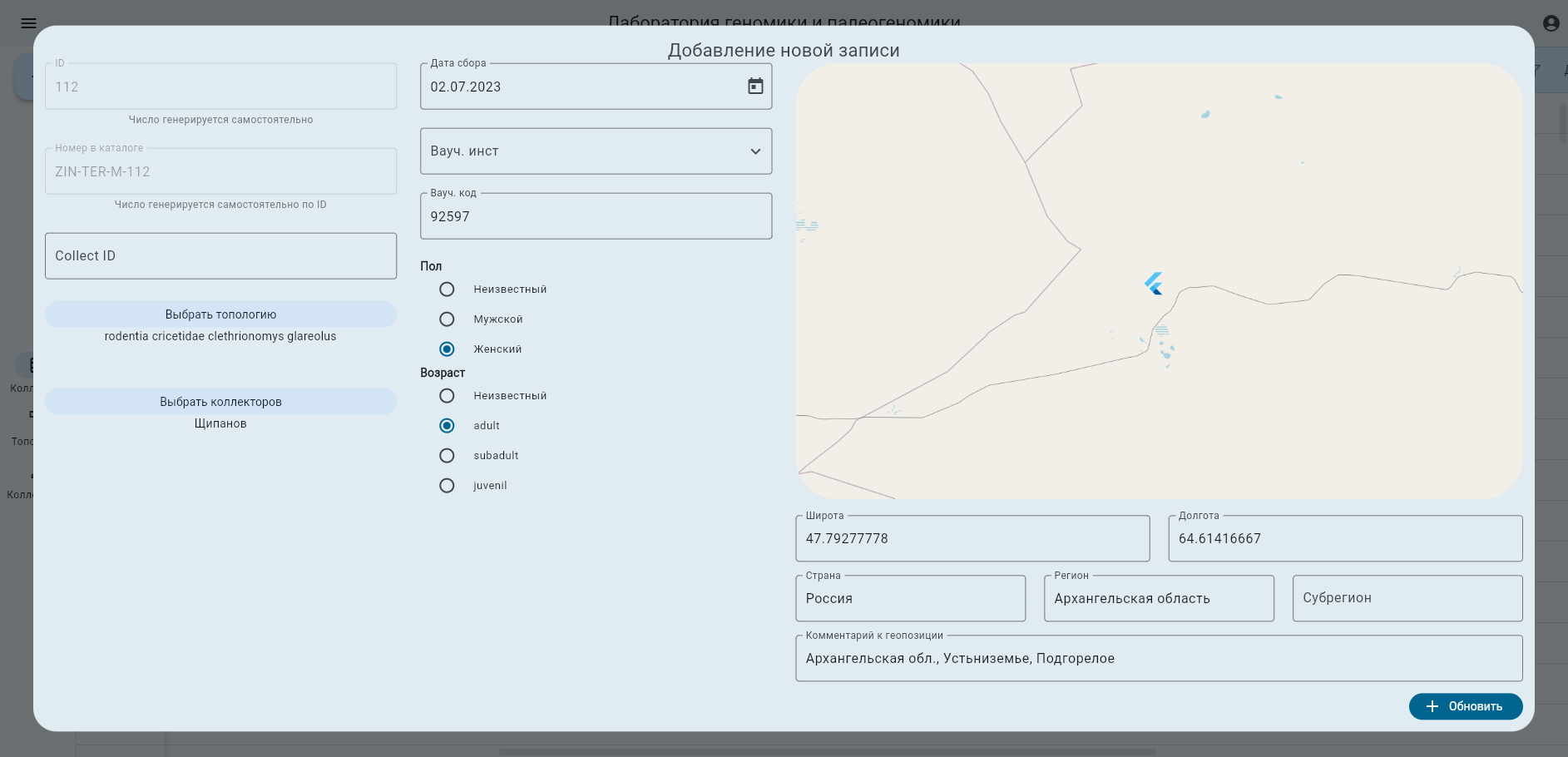


Рисунок 29 – Отображение изменение записи коллекции

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Документация расширения pgcrypto для шифрования PostgreSQL – URL: https://www.postgresql.org/docs/current/pgcrypto.html (дата обращения 01.06.2023).

2 Документация программы для создания restful API на основе базы данных PostgreSQL – URL: https://postgrest.org/en/stable/ (дата обращения 01.06.2023).

3 Документация языка программирования Dart – URL: https://dart.dev/guides (дата обращения 01.06.2023).

4 Документация библиотеки для разработки кроссплатформенных приложений Flutter – URL: https://docs.flutter.dev (дата обращения 01.06.2023).

5 Документация библиотеки для Flutter для отображения таблиц Syncfusion DataGrid – URL: https://help.syncfusion.com/flutter/datagrid/getting-started (дата обращения 01.06.2023).

6 Расширение для создание jwt токенов в PostgrSQL – URL: https://github.com/michelp/pgjwt (дата обращения 01.06.2023).

7 Дизайн система от Google Material 3 – URL: https://m3.material.io (дата обращения 01.06.2023).

8 Библиотека для организации http запросов в языке программировании Dart – URL: https://pub.dev/packages/http (дата обращения 01.06.2023).

9 Nominatim API – API для поиска мест по адресу или по координатам – URL: https://nominatim.org/release-docs/latest/api/Overview/ (дата обращения 01.06.2023).

10 Библиотека для работы с картами – Flutter Map – URL: https://pub.dev/packages/flutter\_map (дата обращения 01.06.2023).

11 Классификация организмов 9 класс – URL: https://www.yaklass.ru/p/biologia/9-klass/osnovy-evoliutcionnogo-ucheniia-246743/poniatie-vida-printcipy-sistematiki-246744/re-32dcb240-dd92-4865-bee7-96de4ee12880#:~:text=%D0%91%D0%BB%D0%B8%D0%B7%D0%BA%D0%B8%D0%B5%20(%D1%80%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5)%20%D0%B2%D0%B8%D0%B4%D1%8B%20%D0%B6%D0%B8%D0%B2%D0%BE%D1%82%D0%BD%D1%8B%D1%85%20%D0%BE%D0%B1%D1%8A%D0%B5%D0%B4%D0%B8%D0%BD%D1%8F%D1%8E%D1%82%D1%81%D1%8F,%2C%20%D0%B0%20%D1%82%D0%B8%D0%BF%D1%8B%20%E2%80%94%20%D0%B2%20%D1%86%D0%B0%D1%80%D1%81%D1%82%D0%B2%D0%BE (дата обращения 01.06.2023).

# ПРИЛОЖЕНИЕ А Созданные таблицы БД

Код созданных таблиц:

CREATE TABLE auth.users (

login text NOT NULL,

pass text NOT NULL,

role name NOT NULL,

avatar text,

CONSTRAINT users\_pass\_check CHECK ((length(pass) < 512)),

CONSTRAINT users\_role\_check CHECK ((length((role)::text) < 512))

);

CREATE TABLE public.age (

id integer NOT NULL,

name character varying(20)

);

CREATE SEQUENCE public.age\_id\_seq

START WITH 0

INCREMENT BY 1

MINVALUE 0

NO MAXVALUE

CACHE 1;

CREATE TABLE public.collection (

id integer NOT NULL,

"CatalogueNumber" text,

collect\_id text,

kind\_id integer NOT NULL,

subregion\_id integer NOT NULL,

gen\_bank\_id character varying(20),

point public.geography(Point,4326),

vouch\_inst\_id integer,

vouch\_id character varying(20),

rna boolean DEFAULT false NOT NULL,

sex\_id integer,

age\_id integer,

day integer,

month integer,

year integer NOT NULL,

comment text,

geo\_comment text,

file\_url text,

CONSTRAINT genbank\_check CHECK ((((gen\_bank\_id)::text ~ 'OP\d+'::text) OR (gen\_bank\_id IS NULL)))

);

CREATE SEQUENCE public.collection\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.collector (

id integer NOT NULL,

last\_name character varying(100) NOT NULL,

first\_name character varying(100),

second\_name character varying(100)

);

CREATE SEQUENCE public.collector\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.collector\_to\_collection (

collector\_id integer NOT NULL,

collection\_id integer NOT NULL

);

CREATE TABLE public.country (

id integer NOT NULL,

name text NOT NULL

);

CREATE SEQUENCE public.country\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.family (

id integer NOT NULL,

order\_id integer NOT NULL,

name character varying(80)

);

CREATE SEQUENCE public.family\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.genus (

id integer NOT NULL,

family\_id integer NOT NULL,

name character varying(80)

);

CREATE SEQUENCE public.genus\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.kind (

id integer NOT NULL,

genus\_id integer NOT NULL,

name character varying(80)

);

CREATE SEQUENCE public.kind\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public."order" (

id integer NOT NULL,

name character varying(80)

);

CREATE SEQUENCE public.order\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.region (

id integer NOT NULL,

country\_id integer NOT NULL,

name text

);

CREATE SEQUENCE public.region\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.sex (

id integer NOT NULL,

name character varying(40) NOT NULL

);

CREATE SEQUENCE public.sex\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.subregion (

id integer NOT NULL,

region\_id integer NOT NULL,

name text

);

CREATE SEQUENCE public.subregion\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

CREATE TABLE public.voucher\_institute (

id integer NOT NULL,

name text NOT NULL

);

CREATE SEQUENCE public.voucher\_institute\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE ONLY public.age ALTER COLUMN id SET DEFAULT nextval('public.age\_id\_seq'::regclass);

ALTER TABLE ONLY public.collection ALTER COLUMN id SET DEFAULT nextval('public.collection\_id\_seq'::regclass);

ALTER TABLE ONLY public.collector ALTER COLUMN id SET DEFAULT nextval('public.collector\_id\_seq'::regclass);

--

-- Name: family id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.family ALTER COLUMN id SET DEFAULT nextval('public.family\_id\_seq'::regclass);

--

-- Name: genus id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.genus ALTER COLUMN id SET DEFAULT nextval('public.genus\_id\_seq'::regclass);

--

-- Name: kind id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.kind ALTER COLUMN id SET DEFAULT nextval('public.kind\_id\_seq'::regclass);

--

-- Name: order id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public."order" ALTER COLUMN id SET DEFAULT nextval('public.order\_id\_seq'::regclass);

--

-- Name: region id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.region ALTER COLUMN id SET DEFAULT nextval('public.region\_id\_seq'::regclass);

--

-- Name: sex id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.sex ALTER COLUMN id SET DEFAULT nextval('public.sex\_id\_seq'::regclass);

--

-- Name: subregion id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.subregion ALTER COLUMN id SET DEFAULT nextval('public.subregion\_id\_seq'::regclass);

--

-- Name: voucher\_institute id; Type: DEFAULT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.voucher\_institute ALTER COLUMN id SET DEFAULT nextval('public.voucher\_institute\_id\_seq'::regclass);

--

-- Name: users users\_pkey; Type: CONSTRAINT; Schema: auth; Owner: postgres

--

ALTER TABLE ONLY auth.users

ADD CONSTRAINT users\_pkey PRIMARY KEY (login);

--

-- Name: age age\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.age

ADD CONSTRAINT age\_pk PRIMARY KEY (id);

--

-- Name: collection collection\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collection

ADD CONSTRAINT collection\_pk PRIMARY KEY (id);

--

-- Name: collector collector\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collector

ADD CONSTRAINT collector\_pk PRIMARY KEY (id);

--

-- Name: collector\_to\_collection collector\_to\_collection\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collector\_to\_collection

ADD CONSTRAINT collector\_to\_collection\_pk PRIMARY KEY (collector\_id, collection\_id);

--

-- Name: country country\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.country

ADD CONSTRAINT country\_pk PRIMARY KEY (id);

--

-- Name: family family\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.family

ADD CONSTRAINT family\_pk PRIMARY KEY (id);

--

-- Name: genus genus\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.genus

ADD CONSTRAINT genus\_pk PRIMARY KEY (id);

--

-- Name: kind kind\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.kind

ADD CONSTRAINT kind\_pk PRIMARY KEY (id);

--

-- Name: order order\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public."order"

ADD CONSTRAINT order\_pk PRIMARY KEY (id);

--

-- Name: region region\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.region

ADD CONSTRAINT region\_pk PRIMARY KEY (id);

--

-- Name: sex sex\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.sex

ADD CONSTRAINT sex\_pk PRIMARY KEY (id);

--

-- Name: subregion subregion\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.subregion

ADD CONSTRAINT subregion\_pk PRIMARY KEY (id);

--

-- Name: voucher\_institute voucher\_institute\_pk; Type: CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.voucher\_institute

ADD CONSTRAINT voucher\_institute\_pk PRIMARY KEY (id);

--

-- Name: users encrypt\_pass; Type: TRIGGER; Schema: auth; Owner: postgres

--

CREATE TRIGGER encrypt\_pass BEFORE INSERT OR UPDATE ON auth.users FOR EACH ROW EXECUTE FUNCTION auth.encrypt\_pass();

--

-- Name: collection collection\_age\_id\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collection

ADD CONSTRAINT collection\_age\_id\_fk FOREIGN KEY (age\_id) REFERENCES public.age(id) ON UPDATE CASCADE ON DELETE RESTRICT;

--

-- Name: collection collection\_kind\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collection

ADD CONSTRAINT collection\_kind\_null\_fk FOREIGN KEY (kind\_id) REFERENCES public.kind(id);

--

-- Name: collection collection\_sex\_id\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collection

ADD CONSTRAINT collection\_sex\_id\_fk FOREIGN KEY (sex\_id) REFERENCES public.sex(id) ON UPDATE CASCADE ON DELETE RESTRICT;

--

-- Name: collection collection\_subregion\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collection

ADD CONSTRAINT collection\_subregion\_null\_fk FOREIGN KEY (subregion\_id) REFERENCES public.subregion(id);

--

-- Name: collection collection\_voucher\_institute\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collection

ADD CONSTRAINT collection\_voucher\_institute\_null\_fk FOREIGN KEY (vouch\_inst\_id) REFERENCES public.voucher\_institute(id);

--

-- Name: collector\_to\_collection collector\_to\_collection\_collection\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collector\_to\_collection

ADD CONSTRAINT collector\_to\_collection\_collection\_null\_fk FOREIGN KEY (collection\_id) REFERENCES public.collection(id) ON UPDATE CASCADE ON DELETE CASCADE;

--

-- Name: collector\_to\_collection collector\_to\_collection\_collector\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.collector\_to\_collection

ADD CONSTRAINT collector\_to\_collection\_collector\_null\_fk FOREIGN KEY (collector\_id) REFERENCES public.collector(id) ON UPDATE CASCADE ON DELETE CASCADE;

--

-- Name: family family\_order\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.family

ADD CONSTRAINT family\_order\_null\_fk FOREIGN KEY (order\_id) REFERENCES public."order"(id);

--

-- Name: genus genus\_family\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.genus

ADD CONSTRAINT genus\_family\_null\_fk FOREIGN KEY (family\_id) REFERENCES public.family(id);

--

-- Name: region region\_country\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.region

ADD CONSTRAINT region\_country\_null\_fk FOREIGN KEY (country\_id) REFERENCES public.country(id);

--

-- Name: subregion subregion\_region\_null\_fk; Type: FK CONSTRAINT; Schema: public; Owner: postgres

--

ALTER TABLE ONLY public.subregion

ADD CONSTRAINT subregion\_region\_null\_fk FOREIGN KEY (region\_id) REFERENCES public.region(id);

Данные в таблицах:

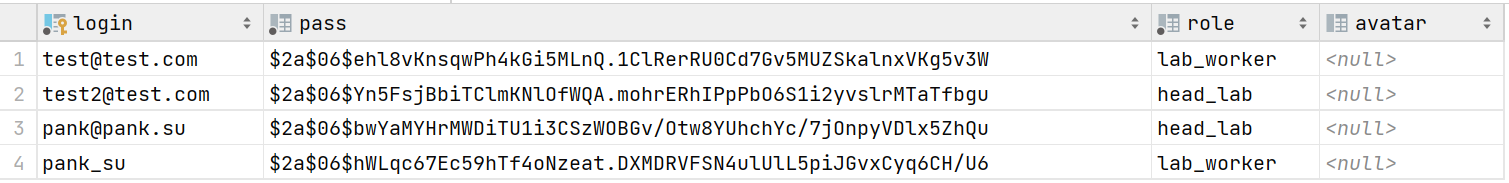


Рисунок A.1 – Таблица users

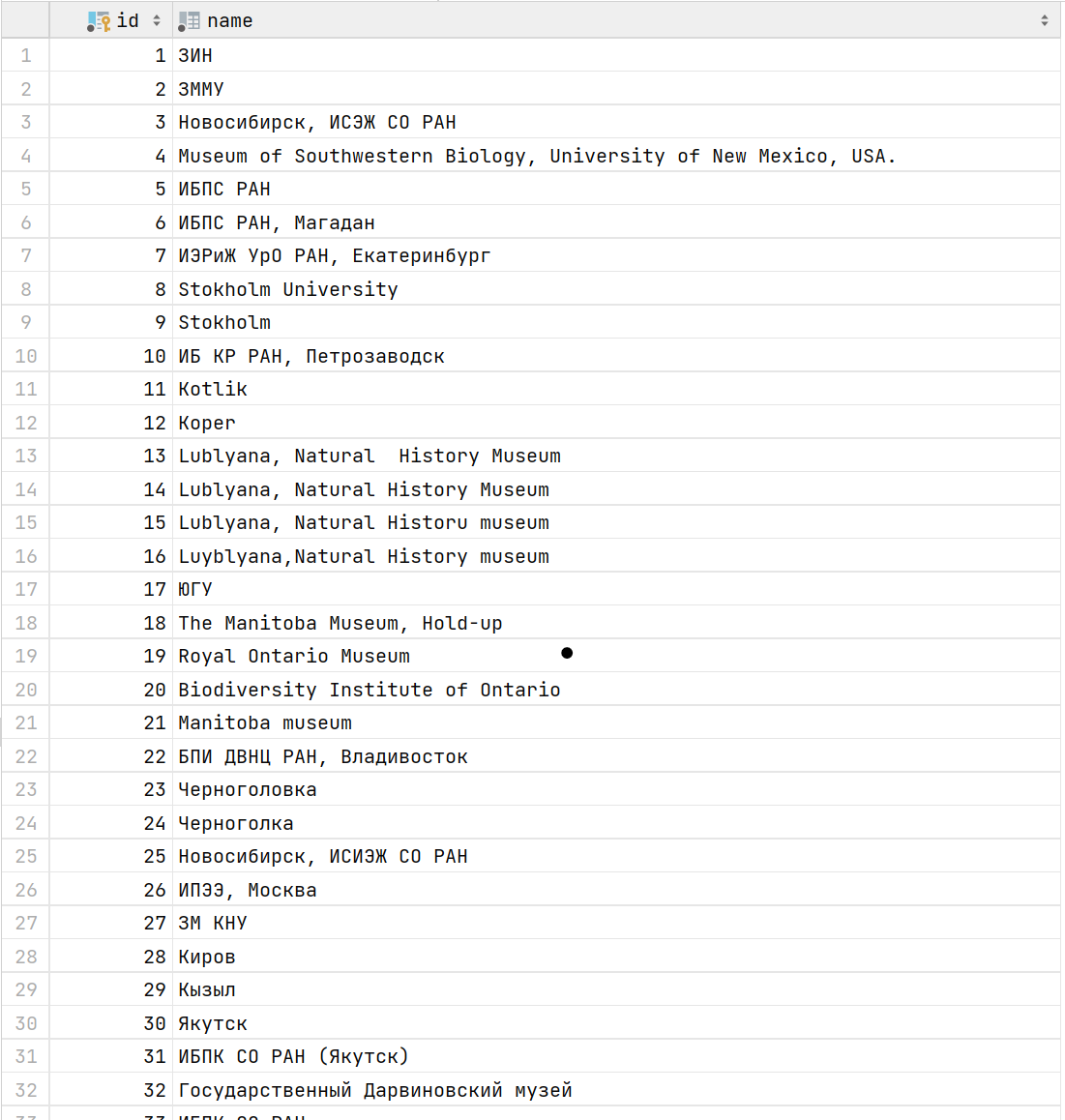


Рисунок A.2 – Таблица voucher\_institute

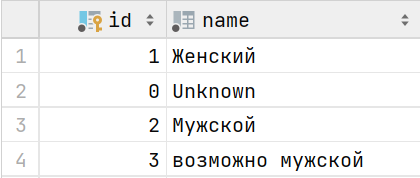


Рисунок A.3 – Таблица sex

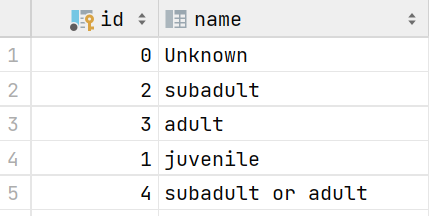


Рисунок A.4 – Таблица age

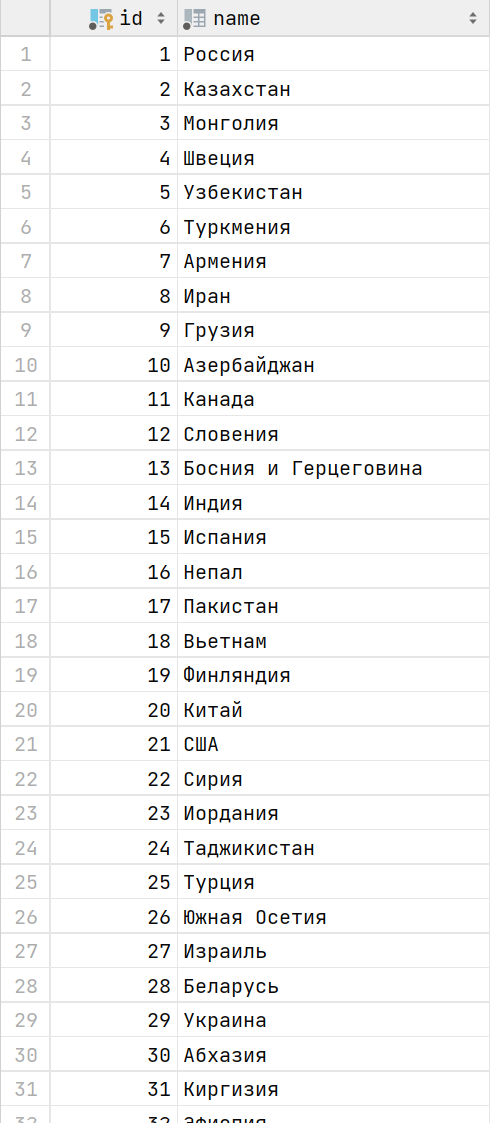


Рисунок A.5 – Таблица country

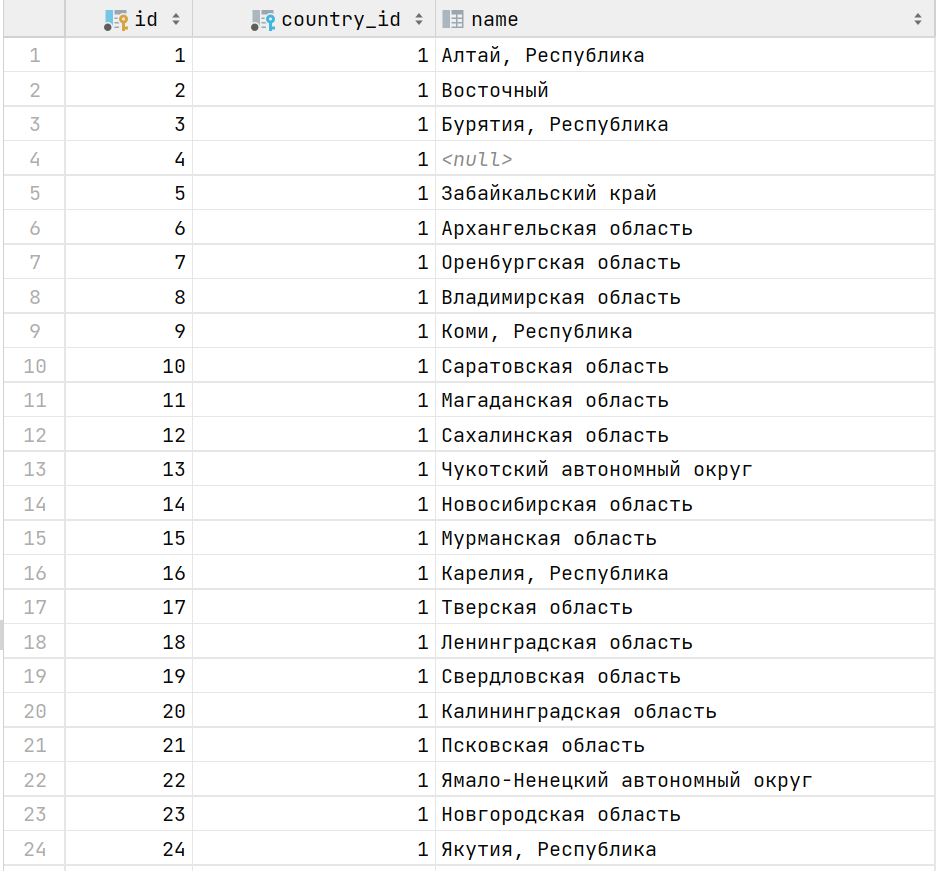


Рисунок A.6 – Таблица region

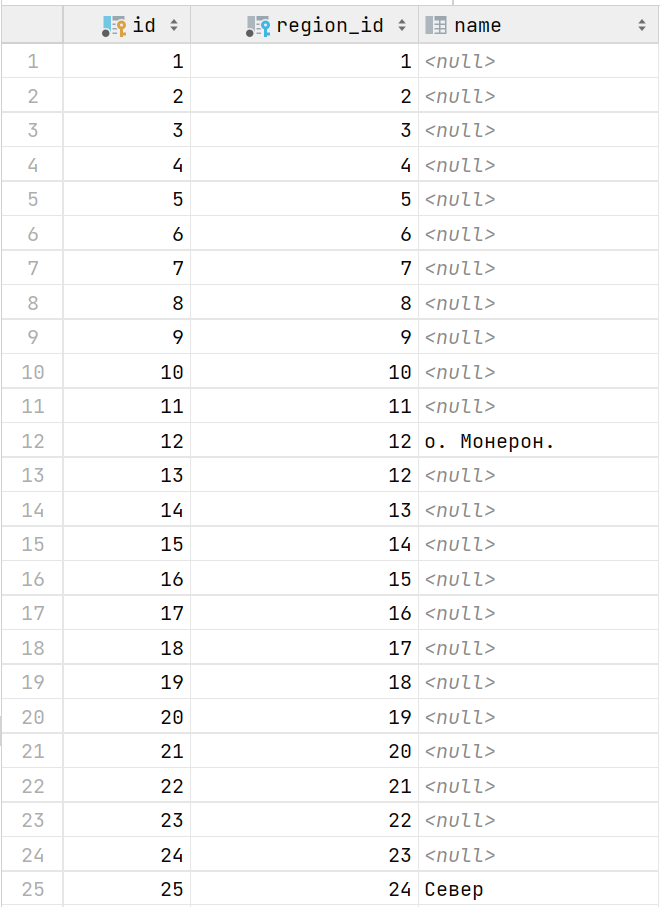


Рисунок A.7 – Таблица subregion



Рисунок A.8 – Таблица order

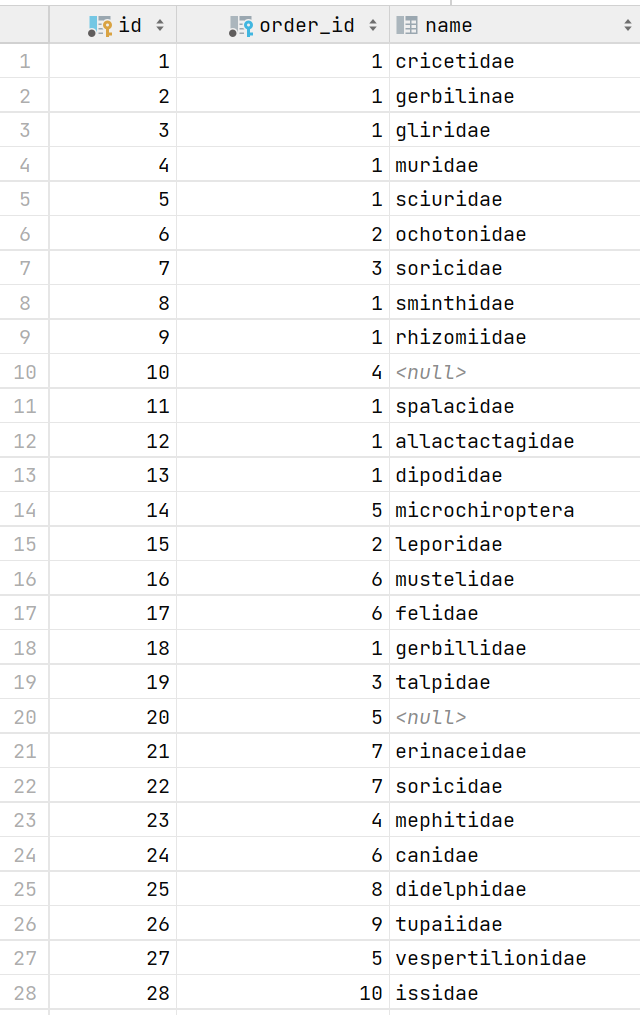


Рисунок A.9 – Таблица family

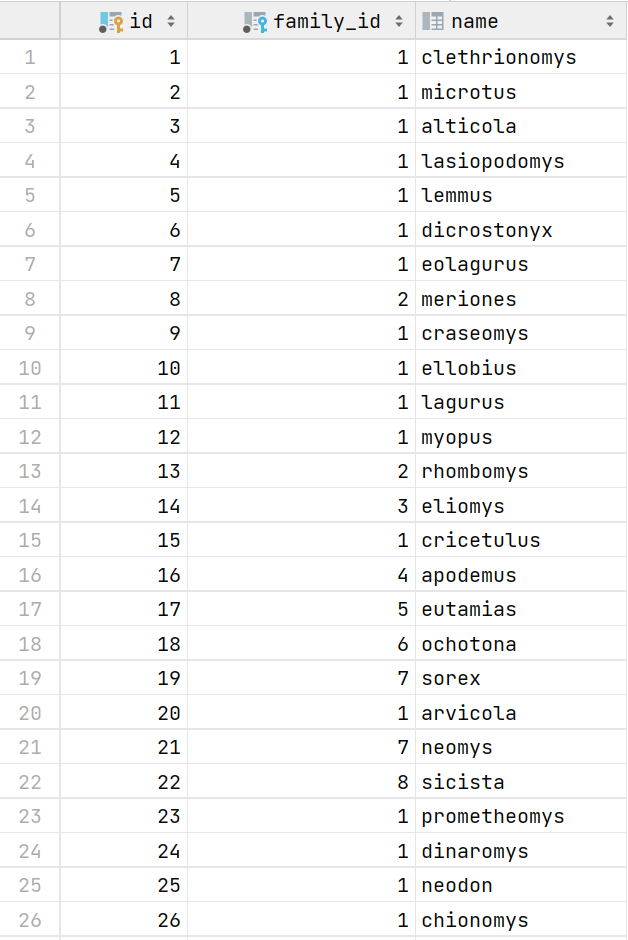


Рисунок A.10 – Таблица genus

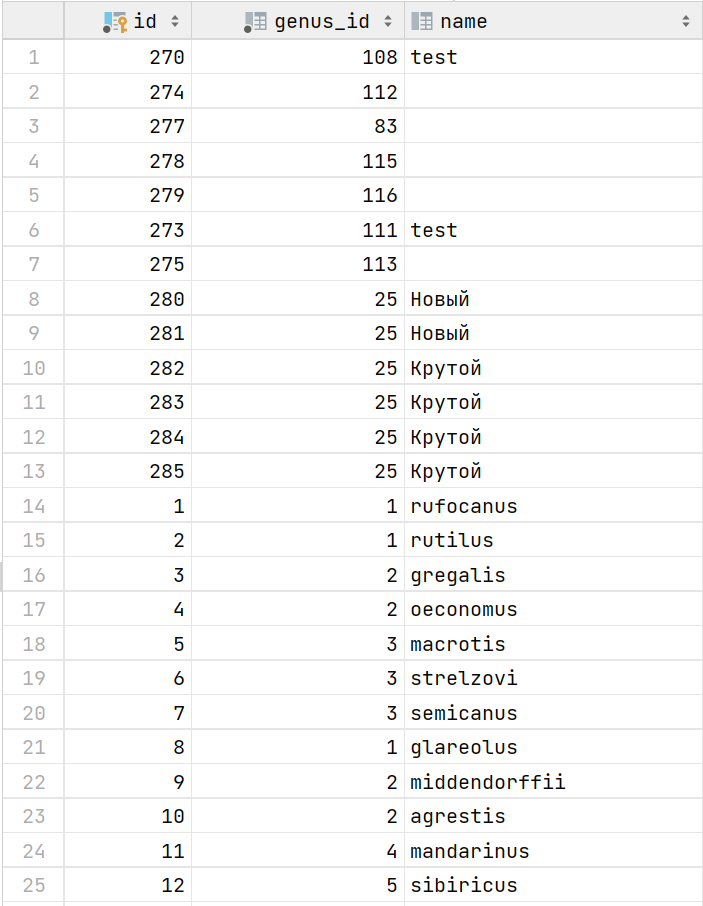


Рисунок A.11 – Таблица kind

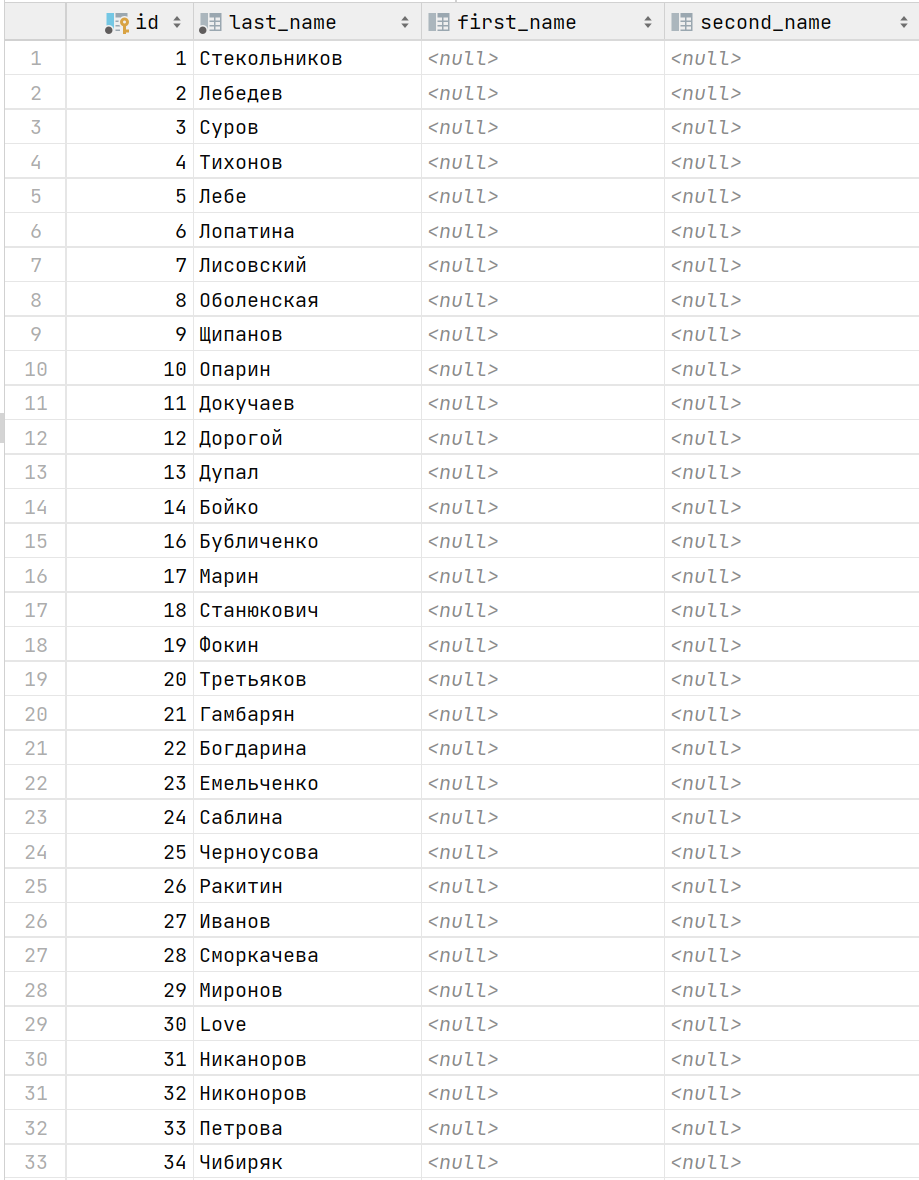


Рисунок A.12 – Таблица collector

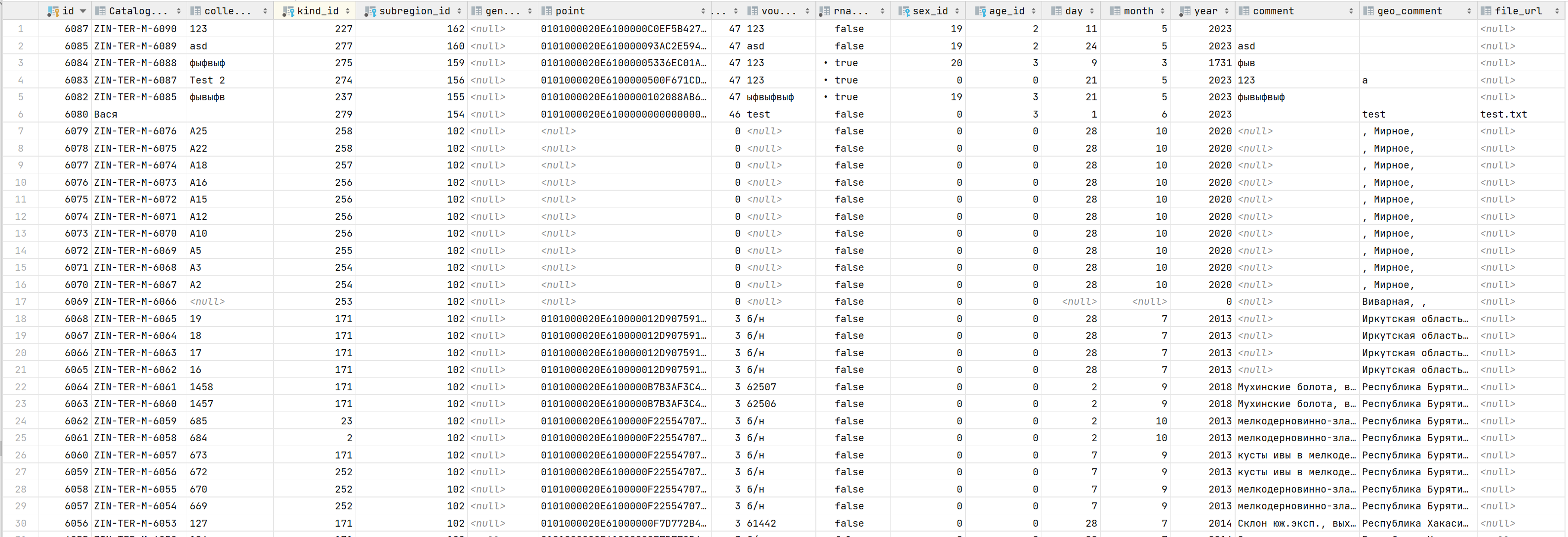


Рисунок A.13 – Таблица collection

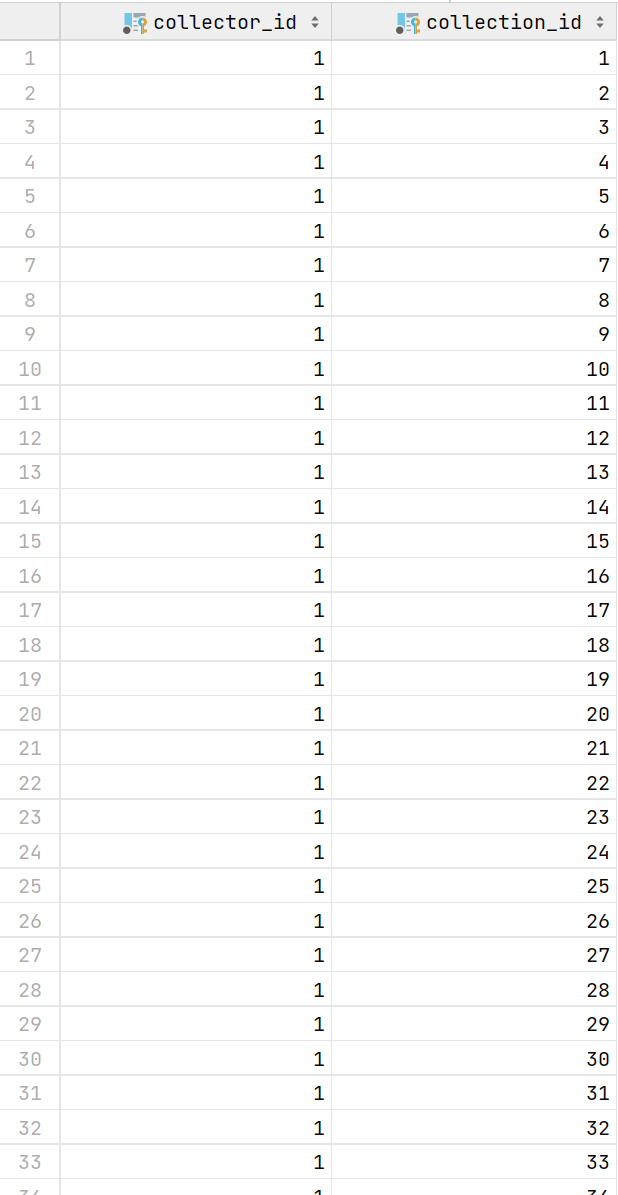


Рисунок A.14 – Таблица collector\_to\_collection

# ПРИЛОЖЕНИЕ Б Разработанные хранимые процедуры и триггеры

CREATE OR REPLACE FUNCTION get\_age\_id(age\_name varchar(20))

RETURNS integer

AS

$$

DECLARE

age\_id integer;

BEGIN

SELECT id INTO age\_id FROM age WHERE $1 = age.name;

IF age\_id IS NULL THEN

INSERT INTO age(name) VALUES ($1);

SELECT id INTO age\_id FROM age WHERE $1 = age.name;

END IF;

RETURN age\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_sex\_id(sex\_name varchar(40))

RETURNS integer

AS

$$

DECLARE

sex\_id integer;

BEGIN

SELECT id INTO sex\_id FROM sex WHERE $1 = sex.name;

IF sex\_id IS NULL THEN

INSERT INTO sex(name) VALUES ($1);

SELECT id INTO sex\_id FROM sex WHERE $1 = sex.name;

END IF;

RETURN sex\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_order\_id(order\_name varchar(80))

RETURNS integer

AS

$$

DECLARE

order\_id integer;

BEGIN

SELECT id INTO order\_id FROM "order" WHERE ($1 = "order".name OR ($1 is NULL AND "order".name is null));

IF order\_id IS NULL THEN

INSERT INTO "order"(name) VALUES ($1);

SELECT get\_order\_id($1) INTO order\_id;

END IF;

RETURN order\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_family\_id(name varchar(80), order\_id integer)

RETURNS integer

AS

$$

DECLARE

family\_id integer;

BEGIN

SELECT id

INTO family\_id

FROM family

WHERE ($1 = family.name OR ($1 is NULL AND family.name is null))

AND $2 = family.order\_id;

IF family\_id IS NULL THEN

INSERT INTO family(name, order\_id) VALUES ($1, $2);

SELECT get\_family\_id($1, $2) INTO family\_id;

END IF;

RETURN family\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_genus\_id(name varchar(80), family\_id integer)

RETURNS integer

AS

$$

DECLARE

genus\_id integer;

BEGIN

SELECT id

INTO genus\_id

FROM genus

WHERE ($1 = genus.name OR ($1 is NULL AND genus.name is null))

AND $2 = genus.family\_id;

IF genus\_id IS NULL THEN

INSERT INTO genus(name, family\_id) VALUES ($1, $2);

SELECT get\_genus\_id($1, $2) INTO genus\_id;

END IF;

RETURN genus\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_kind\_id(name varchar(80), genus\_id integer)

RETURNS integer

AS

$$

DECLARE

kind\_id integer;

BEGIN

SELECT id

INTO kind\_id

FROM kind

WHERE ($1 = kind.name OR ($1 is NULL AND kind.name is null))

AND $2 = kind.genus\_id;

IF kind\_id IS NULL THEN

INSERT INTO kind(name, genus\_id) VALUES ($1, $2);

SELECT get\_kind\_id($1, $2) INTO kind\_id;

END IF;

RETURN kind\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_country\_id(name text)

RETURNS integer

AS

$$

DECLARE

country\_id integer;

BEGIN

SELECT id INTO country\_id FROM country WHERE $1 = country.name;

IF country\_id IS NULL THEN

INSERT INTO country(name) VALUES ($1);

SELECT get\_country\_id($1) INTO country\_id;

END IF;

RETURN country\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_region\_id(name text, country integer)

RETURNS integer

AS

$$

DECLARE

region\_id integer;

BEGIN

SELECT id

INTO region\_id

FROM region

WHERE ($1 = region.name OR ($1 is NULL AND region.name is null))

AND $2 = region.country\_id;

IF region\_id IS NULL THEN

INSERT INTO region(country\_id, name) VALUES ($2, $1);

SELECT get\_region\_id($1, $2) INTO region\_id;

END IF;

RETURN region\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_subregion\_id(name text, region\_id integer)

RETURNS integer

AS

$$

DECLARE

subregion\_id integer;

BEGIN

SELECT id

INTO subregion\_id

FROM subregion

WHERE ($1 = subregion.name OR ($1 is NULL AND subregion.name is null))

AND $2 = subregion.region\_id;

IF subregion\_id IS NULL THEN

INSERT INTO subregion(region\_id, name) VALUES ($2, $1);

SELECT get\_subregion\_id($1, $2) INTO subregion\_id;

END IF;

RETURN subregion\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_vouch\_inst\_id(name text)

RETURNS integer

AS

$$

DECLARE

vouch\_inst\_id integer;

BEGIN

SELECT id INTO vouch\_inst\_id FROM voucher\_institute WHERE $1 = voucher\_institute.name;

IF vouch\_inst\_id IS NULL THEN

INSERT INTO voucher\_institute(name) VALUES ($1);

SELECT get\_vouch\_inst\_id($1) INTO vouch\_inst\_id;

END IF;

RETURN vouch\_inst\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_collector\_id(last\_name varchar(100), first\_name varchar(100) DEFAULT NULL,

second\_name varchar(100) DEFAULT NULL)

RETURNS integer AS

$$

DECLARE

collector\_id integer;

BEGIN

SELECT id

INTO collector\_id

FROM collector

WHERE collector.last\_name = $1

AND (collector.first\_name = $2 or (collector.first\_name is NULL and $2 is NULL))

AND (collector.second\_name = $3 or (collector.second\_name is NULL and $3 is NULL));

IF collector\_id IS NULL THEN

INSERT INTO collector(last\_name, first\_name, second\_name) VALUES ($1, $2, $3);

SELECT get\_collector\_id($1, $2, $3) INTO collector\_id;

END IF;

RETURN collector\_id;

END;

$$ LANGUAGE plpgsql;

-- Добавление в коллекции

CREATE OR REPLACE FUNCTION add\_collection(collect\_id text default null, "order" varchar(80) default null,

family varchar(80) default null, genus varchar(80) default null,

kind varchar(80) default null, age varchar(20) DEFAULT 'Unknown', sex text DEFAULT 'Unknown', vauch\_inst text DEFAULT NULL, vauch\_id text default null,

point geography(point, 4326) default null, country text default null, region text default null, subregion text default null,

geocomment text default null, date\_collect date default null, comment text default null,

collectors text[][3] DEFAULT '{}', rna bool default false)

RETURNS void

LANGUAGE plpgsql

AS

$$

DECLARE

kind\_id\_ integer;

subregion\_id\_ integer DEFAULT 4;

collection\_id\_ integer;

collector text[];

collector\_id\_ integer;

BEGIN

kind\_id\_ := get\_kind\_id($5, (get\_genus\_id($4, (get\_family\_id($3, (get\_order\_id($2)))))));

IF ($11 IS NOT NULL) THEN

subregion\_id\_ := get\_subregion\_id($13, (get\_region\_id($12, (get\_country\_id($11)))));

END IF;

collection\_id\_ := (SELECT id FROM collection ORDER BY id DESC LIMIT 1);

INSERT INTO collection("CatalogueNumber", collect\_id, kind\_id, subregion\_id, point, vouch\_inst\_id, vouch\_id, sex\_id,

age\_id, day, month, year, comment, geo\_comment, rna)

VALUES (concat('ZIN-TER-M-', collection\_id\_), $1, kind\_id\_, subregion\_id\_, $10, get\_vouch\_inst\_id($8), $9,

get\_sex\_id($7), get\_age\_id($6),

extract(day from date\_collect), extract(month from date\_collect), extract(year from date\_collect), $16, $14,

$18);

collection\_id\_ := collection\_id\_ + 1;

IF (array\_length($17, 1) IS NULL) THEN RETURN;

END IF;

FOREACH collector SLICE 1 IN ARRAY $17

LOOP

collector\_id\_ := get\_collector\_id(collector[1]);

INSERT INTO collector\_to\_collection(collector\_id, collection\_id) VALUES (collector\_id\_, collection\_id\_);

END LOOP;

END

$$;

;

-- Удаление из коллекции по ID

CREATE OR REPLACE FUNCTION public.remove\_collection\_by\_id(col\_id int)

RETURNS void

LANGUAGE plpgsql AS

$$

BEGIN

DELETE FROM collection WHERE col\_id = id;

END;

$$;

-- Обновление записи

CREATE OR REPLACE FUNCTION update\_collection\_by\_id(col\_id int, collect\_id text DEFAULT null,

"order" varchar(80) DEFAULT null,

family varchar(80) DEFAULT null, genus varchar(80) DEFAULT null,

kind varchar(80) DEFAULT null, age varchar(20) DEFAULT 'Unknown',

sex text DEFAULT 'Unknown', vauch\_inst text DEFAULT null,

vauch\_id text DEFAULT null,

point geography(point, 4326) DEFAULT null, country text DEFAULT null,

region text DEFAULT null,

subregion text DEFAULT null,

geocomment text DEFAULT null, date\_collect date DEFAULT null,

comment text DEFAULT null,

collectors text[][3] DEFAULT '{}', rna bool default false)

RETURNS void

LANGUAGE plpgsql AS

$$

DECLARE

kind\_id\_ integer;

subregion\_id\_ integer DEFAULT 4;

collector text[];

collector\_id\_ int;

BEGIN

kind\_id\_ := get\_kind\_id($6, (get\_genus\_id($5, (get\_family\_id($4, (get\_order\_id($3)))))));

IF ($12 IS NOT NULL) THEN

subregion\_id\_ := get\_subregion\_id($14, (get\_region\_id($13, (get\_country\_id($12)))));

END IF;

UPDATE collection

SET kind\_id = kind\_id\_,

"CatalogueNumber" = concat('ZIN-TER-M-', col\_id),

subregion\_id = subregion\_id\_,

collect\_id = $2,

age\_id = get\_age\_id(age),

sex\_id = get\_sex\_id(sex),

vouch\_inst\_id = get\_vouch\_inst\_id(vauch\_inst),

vouch\_id = $10,

point = $11,

geo\_comment = geocomment,

day = extract(day from date\_collect),

month = extract(month from date\_collect),

year = extract(year from date\_collect),

comment = $17,

rna = $19

WHERE id = col\_id;

DELETE FROM collector\_to\_collection WHERE collection\_id = col\_id;

IF (array\_length($18, 1) IS NULL) THEN

RETURN;

END IF;

FOREACH collector SLICE 1 IN ARRAY $18

LOOP

collector\_id\_ := get\_collector\_id(collector[1]);

INSERT INTO collector\_to\_collection(collector\_id, collection\_id) VALUES (collector\_id\_, col\_id);

END LOOP;

END

$$;

;

create schema auth;

-- проверка существует ли роль

create or replace function

auth.check\_role\_exists() returns trigger as

$$

begin

if not exists (select 1 from pg\_roles as r where r.rolname = new.role) then

raise foreign\_key\_violation using message =

'unknown database role: ' || new.role;

-- return null;

end if;

return new;

end

$$ language plpgsql;

drop trigger if exists ensure\_user\_role\_exists on auth.users;

create constraint trigger ensure\_user\_role\_exists

after insert or update

on auth.users

for each row

execute procedure auth.check\_role\_exists();

create extension if not exists pgcrypto;

-- триггер хеширования паролей

create or replace function

auth.encrypt\_pass() returns trigger as

$$

begin

if tg\_op = 'INSERT' or new.pass <> old.pass then

new.pass = crypt(new.pass, gen\_salt('bf'));

end if;

return new;

end

$$ language plpgsql;

drop trigger if exists encrypt\_pass on auth.users;

create trigger encrypt\_pass

before insert or update

on auth.users

for each row

execute procedure auth.encrypt\_pass();

-- получение роли(фактически авторизация)

create or replace function

auth.user\_role(login text, pass text) returns name

language plpgsql

as

$$

begin

return (select role

from auth.users

where users.login = user\_role.login

and users.pass = crypt(user\_role.pass, users.pass));

end;

$$;

-- тип token - токен для входа пользователей

CREATE TYPE auth.jwt\_token AS

(

token text

);

create extension if not exists pgjwt;

ALTER DATABASE lab\_base SET "app.jwt\_secret" TO 'Q5He86xPvYscMiZxQw29gy8YkbD7a4aMDH1hQFP';

-- функция авторизации, для анонимного пользователя нужно только выполненные функций

create or replace function

public.login(login text, pass text) returns auth.jwt\_token as

$$

declare

\_role name;

result auth.jwt\_token;

begin

-- check email and password

select auth.user\_role(login, pass) into \_role;

if \_role is null then

raise invalid\_password using message = 'invalid user or password';

end if;

-- НЕ КОМУ НЕ СООБЩАТЬ КОД, НЕ ХРАНИТЬ ЕГО В ОТКРЫТЫХ ПЕРЕМЕННЫХ

select sign(

row\_to\_json(r), current\_setting('app.jwt\_secret')

) as token

from (select \_role as role,

$1 as login,

extract(epoch from now())::integer + 60 \* 60 \* 24 as exp) r

into result;

return result;

end;

$$ language plpgsql security definer;

CREATE TYPE user\_info AS

(

login text,

avatar\_url text,

role text

);

CREATE OR REPLACE FUNCTION get\_user\_info() RETURNS user\_info

LANGUAGE plpgsql AS

$$

DECLARE login\_ text := current\_setting('request.jwt.claims', true)::json->>'login';

DECLARE role text := current\_setting('request.jwt.claims', true)::json->>'role';

DECLARE result user\_info;

BEGIN

SELECT avatar INTO result.avatar\_url FROM auth.users WHERE login\_ = login;

result.role := role;

result.login := login\_;

return result;

END

$$ SECURITY DEFINER;

CREATE OR REPLACE FUNCTION add\_topology("order" varchar(80), family varchar(80) DEFAULT null, genus varchar(80) DEFAULT null,

kind varchar(80) DEFAULT null) RETURNS text AS

$$

BEGIN

PERFORM (SELECT get\_kind\_id(kind, get\_genus\_id(genus, get\_family\_id(family, get\_order\_id("order")))));

RETURN 'ok';

END

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION check\_auth() RETURNS text AS

$$

DECLARE

login\_ text := current\_setting('request.jwt.claims', true)::json ->> 'login';

role\_ text := current\_setting('request.jwt.claims', true)::json ->> 'role';

BEGIN

IF EXISTS(SELECT \* FROM auth.users u WHERE u.role = role\_ AND u.login = login\_) THEN

RETURN 'ok';

END IF;

RAISE sqlstate 'PT403' using message = 'Вы неавторизованы!',

DETAIL = 'Обновите токен',

HINT = 'Перезайдите в аккаунт или попробуйте обновить токен';

END

$$ LANGUAGE plpgsql SECURITY DEFINER ;