МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «системы отслеживания ошибок»

**Исполнитель**

студент 2 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. С. Степаненко

подпись, дата

**Руководитель**

Преподаватель-стажёр \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

должность, ученая степень, ученое звание подпись, дата

Допущена к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

подпись дата инициалы и фамилия

Минск 2024

Оглавление

[Введение 4](#_Toc166882174)

[1 Анализ требований к программному средств 5](#_Toc166882175)

[1.1 Анализ аналогов 5](#_Toc166882176)

[1.1.1 Аналог Stack Overflow 5](#_Toc166882177)

[1.1.2 Аналог Quora 6](#_Toc166882178)

[1.2 Разработка функциональных требований, определение вариантов использования. 7](#_Toc166882179)

[1.3 Диаграмма UML, взаимосвязь компонентов. 8](#_Toc166882180)

[1.4 Описание информационных объектов 9](#_Toc166882181)

[1.5 Вывод 10](#_Toc166882182)

[2. Разработка объектов базы данных 11](#_Toc166882183)

[2.1 Таблицы 11](#_Toc166882184)

[2.2 Разработка представлений базы данных 12](#_Toc166882185)

[2.3. Разработка индексов базы данных 13](#_Toc166882186)

[2.4. Разработка функций базы данных 14](#_Toc166882187)

[2.5. Разработка процедур базы данных. 14](#_Toc166882188)

[2.6. Разработка триггеров базы данных. 15](#_Toc166882189)

[2.7. Разработка пользователей 15](#_Toc166882190)

[2.8. Вывод 17](#_Toc166882191)

[3. Описание функций импорта и экспорта 18](#_Toc166882192)

[4. Тестирование производительности 19](#_Toc166882193)

[5. Описание технологии и ее применение в базе данных 21](#_Toc166882194)

[6. Руководство пользователя 23](#_Toc166882195)

[7. Краткое описание приложения для демонстрации 24](#_Toc166882196)

[7.1 Пример работы приложения от лица администратора 24](#_Toc166882197)

[Заключение 26](#_Toc166882198)

[Список используемых источников 27](#_Toc166882199)

[Приложения А – код создания таблиц 28](#_Toc166882200)

[Приложение Б – код создания функций 29](#_Toc166882201)

[Приложение В – код создания процедур 34](#_Toc166882202)

[Приложение Г – код реализации импорта и экспорта 37](#_Toc166882203)

[Приложение Д – код создание представлений 38](#_Toc166882204)

[Приложение Е – код создания триггеров 39](#_Toc166882205)

[Приложение Ё – код создания пользователей и ролей 39](#_Toc166882206)

[Приложение Ж – код создания индексов. 42](#_Toc166882207)

# Введение

Цель данной работы заключается в создании реляционной базы данных для платформы по отслеживанию ошибок, которая обеспечивает пользователя доступом к имеющимся на ней разносторонним ошибкам для поиска решения.   
 База данных - это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Postgres SQL, поскольку эта система обладает высокой надежностью и производительностью, что позволяет обеспечить эффективное хранение, обработку и управление данными.

Основные требования к базе данных:

* Реализация ролей администратора и пользователя;
* Поиск отчетов по статусу, типу проблемы или состоянию;
* Загрузка отчетов на платформу пользователем;
* Взаимодействие с базой данных при помощи хранимых процедур и функций.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта.

# Анализ требований к программному средств

## 1.1 Анализ аналогов

Система отслеживания ошибок – это важный инструмент для разработчиков и команд, занимающихся программированием. Она позволяет обнаруживать, регистрировать и анализировать ошибки, которые возникают в программном обеспечении. В нашем проекте мы сосредоточимся на создании системы, которая поможет эффективно управлять ошибками в коде.

### Аналог Stack Overflow

Stack Overflow – это публичная и частная платформа, где разработчики могут находить и предоставлять ответы на технические вопросы, а компании могут зафиксировать и делиться своими знаниями. Stack Overflow помогает разработчикам, командам и организациям повышать производительность, сокращать время выхода на рынок и обмениваться знаниями.

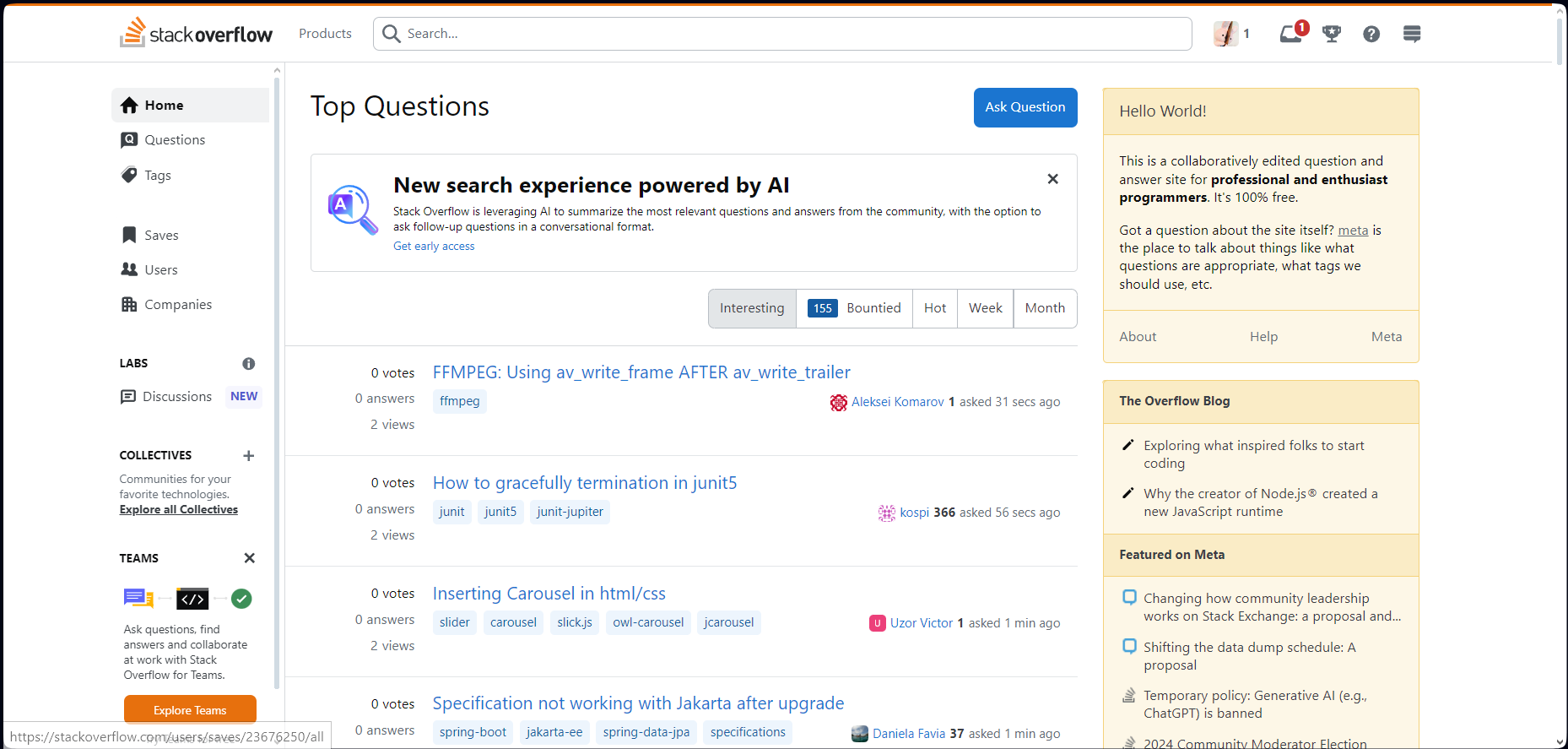


Рисунок 1.1.1 – Аналог Stack Overflow

Одной из основных функций Stack Overflow является поиск и разбор ошибок связанных с разными аспектами информационных технологий, будь то ошибки в ПО, в коде, или ошибки в программах.  
 Stack Overflow предоставляет возможности публиковать свои ошибки, если таковых ранее не встречалось в истории сайта, либо смотреть решения этих, либо ошибок, опубликованных другими пользователями.

Ещё одна функция Stack Overflow – передача собственного опыта другим разработчикам. Коммуникация в данном случае играет важную роль.

Stack Overflow использует собственную базу данных для хранения данных. Эта база данных содержит информацию о вопросах, ответах, комментариях, пользователях и других аспектах, связанных с программированием и разработкой. Она предоставляет возможность эффективно организовывать и обрабатывать огромное количество контента, созданного сообществом разработчиков.

### Аналог Quora

Quora - это социальный веб-сервис вопросов и ответов, а также онлайн-рынок знаний.

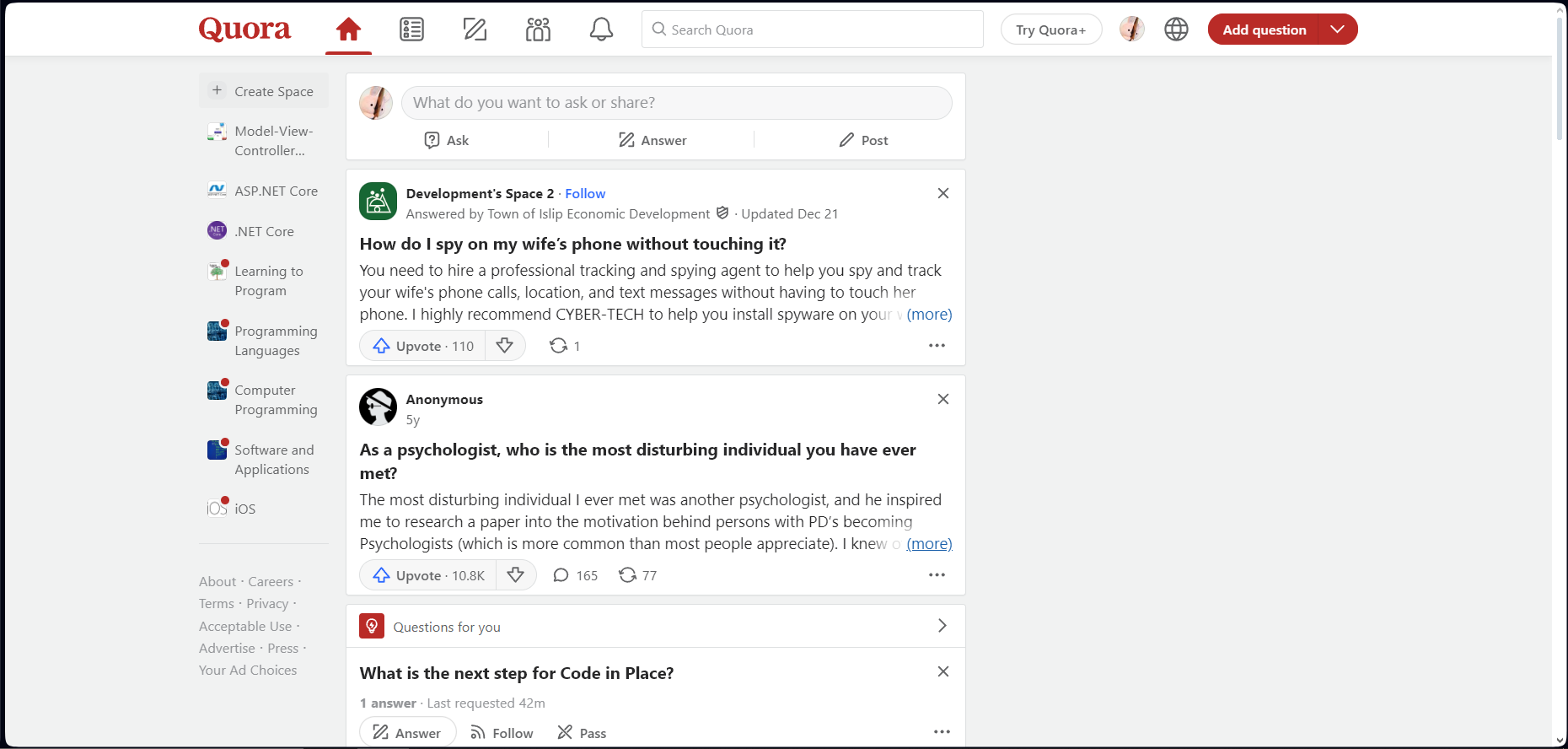


Рисунок 1.1.2 – Аналог Quora.

**Основная функция Quora**заключается в предоставлении ответов на вопросы. Опубликованные вопросы приводят к обсуждениям, которые могут состоять из одного ответа или включать дополнительные комментарии.

Quora предоставляет возможности публиковать свои ошибки, если таковых ранее не встречалось в истории сайта, либо смотреть решения этих, либо ошибок, опубликованных другими пользователями.

Quora использует MySQL в качестве базы данных для хранения данных. Однако интересно, что Quora управляет данными без жесткой схемы. Вместо этого они используют подход, который называется схемой на основе сущностей (Entity-Attribute-Value, EAV).

## Разработка функциональных требований, определение вариантов использования.

Функциональные требования базы данных определяют, как база данных должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Кроме того, функциональные требования могут определять интеграцию базы данных с другими системами и программами. Например, для системы отлова ошибок функциональные требования могут включать в себя функции для хранения информации об ошибках, комментария и пользователях, поиска ошибок по категориям и критериям, создание и управление новых отчетов, а также функции для оценки и просмотра отчетов

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

Роли пользователя — это набор прав, которые пользователь может получить в системе. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* Unathorized User.
* User.
* Admin.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.3.

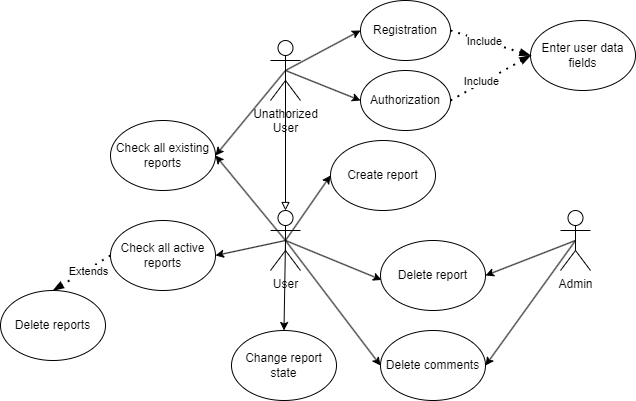


Рисунок 1.2 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет доступна только просмотр всей информации без возможности оставлять комментарии, создавать новые записи. После регистрации пользователь становится пользователем User.

Роль User дает возможность создания собственных отчетов и взаимодействия с чужими: добавление комментариев, просмотр истории состояния проекта, манипуляции над собственными отчетами.

Роль Admin дает возможности манипуляции над чужими отчетами и комментариями в случае не надлежащего контента в них. Помимо этого администратор имеет полный доступ к профилям других пользователей.

## 1.3 Диаграмма UML, взаимосвязь компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) - это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных будет представлена на рисунке 2.1.

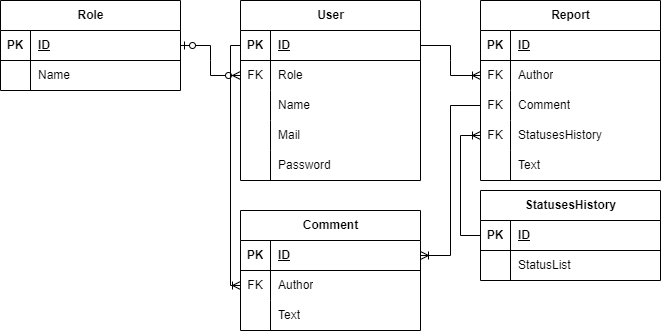


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-многим", "многие-ко-многим". Например, таблица Users связана с таблицами Report, Comment через внешние ключи ID.

## 1.4 Описание информационных объектов

Для реализации базы данных было разработано 9 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Role, Users, Report, Comment, Status, StatusHistory. Ниже будет описание про каждую из них более подробно.

Таблица Role представляет собой перечень ролей, которые могут иметь пользователи. Имеющиеся поля: ID (идентификатор роли, типа SERIAL, первичный ключ), role\_name (текстовое название роли, тип VARCHAR(255)).

Таблица Users содержит информацию о пользователях. Имеющиеся поля: ID (идентификатор пользователя, типа SERIAL, первичный ключ), Name (имя пользователя, тип VARCHAR(255)), Role (роль пользователя, тип Serial внешний ключ), Mail (адрес электронной почты пользователя, тип VARCHAR(255)), Password (пароль пользователя, тип VARCHAR(255)),

Таблица Report содержит информацию об отчетах. Имеющиеся поля: ID (идентификатор отчета, типа SERIAL, первичный ключ), Author (идентификатор автора, тип Serial внешний ключ), Comment (комментарии к отчету, тип Serial Array, внешний ключ), StatusesHistory (история статусов отчета, тип serial array, внешний ключ), Status (активный статус отчета, тип Serial внешний ключ), Text (содержимое отчета, тип VARCHAR(255)).

Таблица Comment содержит информацию о комментариях. Имеющиеся поля: ID (идентификатор комментария, тип Serial, первичный ключ), Author (идентификатор автора, тип Serial внешний ключ), Text (содержимое комментария, тип VARCHAR(255)) .

Таблица Status содержит информацию о комментариях. Имеющиеся поля: ID (идентификатор статуса, тип Serial, первичный ключ), Title (именование статуса, тип VARCHAR(255)) .

Таблица StatusHistory содержит информацию о комментариях. Имеющиеся поля: ID (идентификатор истории статусов, тип Serial, первичный ключ), StatusList(прошлые статусы, тип serial array) .

## 1.5 Вывод

Разработка архитектуры проекта необходима для определения структуры и функциональности базы данных.

Описание информационных объектов является важной частью архитектуры проекта, так как это помогает понять, какие данные будут использоваться в приложении, и как они будут храниться и обрабатываться. В данном проекте были описаны объекты, такие как пользователи, роли, отчеты, статусы и комментарии.

В целом, разработка архитектуры проекта является важным шагом в процессе разработки базы данных, так как это позволяет определить необходимые компоненты и информационные объекты, которые помогут создать функциональную и эффективную базу данных.

# 2. Разработка объектов базы данных

## 2.1 Таблицы

Для реализации базы данных «Магазин автомобилей» было разработано 7 таблиц: Report, Comment, Role, Users, StatusHistory.

Таблица Report представляет список отчетов состоит из столбцов (таблица 2.1):

Таблица 2.1 – Столбцы таблицы Reviews

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID | Хранит уникальный идентификатор | Int |
| Author | Хранит id пользователя | Int |
| Text | Хранит текст отзыва | text |
| Title | Хранит заголовок объявления | text |
|  |  |  |

Таблица Comment представляет комментарии к посту, состоит из столбцов (таблица 2.2):

Таблица 2.2 – Столбцы таблицы Comment

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | номер заказа, тип, первичный ключ | int |
| Author | идентификатор автора, тип, внешний ключ для связи с таблицей users | int |
| Report | Идентификатор отчета, внешний ключ для связи с таблицей report | int |
| Text | Содержимое комментария | Text |

Таблица Role представляет список ролей, которые будут присвоены пользователям. Данная таблица состоит из столбцов представленных в таблице (таблица 2.3):

Таблица 2.3 – Столбцы таблицы Role

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | идентификатор автомобиля, тип, первичный ключ | int |
| model id | идентификатор модели, тип, внешний ключ для связи с таблицей Models | int |

Таблица Users отображает существующих пользователей из столбцов (таблица 2.4):

Таблица 2.4 – Столбцы таблицы Users

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Уникальный индентификатор | int |
| role | роль | int |
| name | имя | text |
| mail | почта | text |
| password | Хэшированный пароль | text |

Таблица StatusHistory представляет список статусов к посту(таблица 2.5):

Таблица 2.5 – Столбцы таблицы StatusHistory

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор истории, первичный ключ | int |
| rerpor | Отчет к которому привязана история | int |
| statuses | Список статусов, тип | Text[] |

## 2.2 Разработка представлений базы данных

Представление в базе данных – это виртуальная таблица, которая формируется на основе выполнения запроса к одной или нескольким таблицам в базе данных. Представления позволяют объединять данные из нескольких таблиц в одну, не изменяя исходных таблиц и их структуры. Представления используются для обеспечения удобства доступа и управления данными в базе данных, а также для улучшения производительности запросов. В данном проекте были созданы три представления:

* UsersWithRole, представление для отображения информации о пользователях (Users) вместе с их ролью (Role);
* AuthorizationUserData , представление для отображения информации о юзере необходимой только для авторизации;
* ReportsWithAuthors, для отображения информации об отчете и его авторе;
* StatusHistoryWithReportInfo, для отображение отчета и его истории статусов.

Реализация нескольких из представлений представлено в листинге 3.2. Остальные представления будут аналогичны, работая с другими данными.

|  |
| --- |
| -- Представлеие отображающее информацию о пользователе и его роли  CREATE OR REPLACE VIEW UsersWithRole AS  SELECT public.role.name AS roleName, public.users.name AS userName  from users inner join role on role.id = users.role  SELECT \* FROM UsersWithRole;  DROP VIEW UsersWithRole;  -- Представление отображающее информацию о пользователе для авторизации  CREATE OR REPLACE VIEW AuthorizationUserData AS  SELECT name, mail, password FROM USERS;  SELECT \* FROM AuthorizationUserData;  DROP VIEW AuthorizationUserData; |

Листинг 2.2 – Представление UsersWithRole и AuthorizationUserData.

## 2.3. Разработка индексов базы данных

Индекс в базе данных представляет собой объект, который используется для ускорения поиска данных. Если таблица содержит большое количество строк, то последовательный поиск данных может занимать много времени. Индекс создается на основе значений одного или нескольких столбцов таблицы и указывает на соответствующие строки таблицы. Использование индексов помогает улучшить производительность базы данных, поскольку они имеют оптимизированную структуру для поиска, например, сбалансированное дерево. Один из наиболее распространенных алгоритмов индексации – это B-дерево (B-tree). B-дерево – это сбалансированное дерево поиска, которое обеспечивает эффективный поиск элементов, используя ключи для сортировки данных в индексе.

Например, для того, чтобы быстро находить отчеты по названию или содержанию, пользователей по имени, комментарии по их содержанию, было создано четыре индекса. Создание индексов представлено в листинге 3.3.

|  |
| --- |
| CREATE INDEX idx\_text ON Report (text);  DROP INDEX idx\_text;  CREATE INDEX idx\_title ON Report (title);  DROP INDEX idx\_title;  CREATE INDEX idx\_name ON Users (name);  DROP INDEX idx\_name;  CREATE INDEX idx\_comment\_text ON Comment (text); |

Листинг 2.3 – Индексы таблиц пользователей, автомобилей и отзывов.

Применение индексов в базе данных значительно повышает скорость выполнения операций поиска, сортировки и фильтрации данных, особенно в случае больших объемов информации. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах. Поэтому необходимо балансировать количество и типы индексов, чтобы обеспечить оптимальную производительность базы данных.

## 2.4. Разработка функций базы данных

Для управления данными через приложение все пользователи использует функции. Функция – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Стоит отметить, что функции не могут содержать группы операторов DML и DDL. В моём приложении реализовано 12 функций.

GetAllReports () - Возвращает все записи из таблицы "Report".

GetAllReportsByAuthor(authorId integer) - Возвращает запись из таблицы "Report" по указанному идентификатору автора.

GetReportByID(reportId integer) - Возвращает все записи из таблицы "Report", по уникальному идентификатору отчета.

GetAllUsers () - Возвращает запись из таблицы "Users".

GetAllUsersWithRole(roleId integer)- Возвращает все записи из таблицы "Users" по идентификатору роли.

GetUserByID(userId integer)- Возвращает запись из таблицы "Users" по указанному идентификатору пользователя.

GetAllCommentsToReport(reportId integer) - Возвращает запись из таблицы "Comments" по указанному идентификатору отчета.

GetComment(commentId) - Возвращает все записи из таблицы "Comment", по уникальному идентификатору комментария.

DeleteComment() – Используется в триггере для удаления всех комментариев к отчету.

GetStatusHistoryToReport(reportId)- Возвращает все записи из таблицы "StatusHistor", по уникальному идентификатору.

CreateStatusHistory()- Используется в триггере для создания новой истории статусов для нового отчета.

## 2.5. Разработка процедур базы данных.

По большому счёту, процедуры выполняют аналогичные действия с функциями, за тем исключением, что они не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Всего было разработано мною 11 процедура.

CreateNewReport - Добавляет информацию о новом отчете в базу данных.

DeleteReport – Удаляет информацию об отчете из базы данных.

UpdateReport – Обновляем информацию о новом консультанте в базу данных.

CreateStatusHistory – Добавляет информацию об истории статусов в базу данных.

UpdateStatusHistory – Обновляет историю статусов.

CreateNewUser– Создает пользователя в системе.

UpdateUser– Обновляет пользователя в базе данных.

DeleteUser – Удаляет пользователя из базы данных.

CreateComment – Добавляет новый комментарий в базу данных.

DeleteComment– Удаляет комментарий из базы данных.

## 2.6. Разработка триггеров базы данных.

Триггер в базе данных – это объект, который автоматически выполняет определенные действия при возникновении определенных событий в таблице или представлении базы данных. Триггер может быть настроен на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для поддержки целостности данных, контроля доступа к данным и автоматической обработки данных при выполнении определенных операций в таблице.

Всего было создано 3 триггера.

CreateStatusHistoryTrigger – Создание новой истории статусов для нового отчета.

DeleteStatusHistoryTrigger – Удаление истории статусов при удалении родительского отчета.

DeleteCommentsTrigger – Удаление всех комментариев под отчетом при удалении родительского отчета.

## 2.7. Разработка пользователей

Пользователь базы данных — это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

При проектировании базы данных было создано 4 типа пользователей: клиент, консультант, менеджер и директор.

Создание пользователей представлено в листинге 3.7.

|  |
| --- |
| CREATE USER baseUser WITH PASSWORD '12345';  ALTER ROLE Director SET ROLE baseUserRole;  CREATE USER adminUser WITH PASSWORD '1234';  ALTER ROLE Client SET ROLE adminUserRole; |

Листинг 2.7 – Создание пользователей

Далее для созданных пользователей базы данных были выданы необходимые привилегии необходимые для функционала пользователей.

|  |
| --- |
| GRANT EXECUTE ON FUNCTION GetAllReports() TO base\_user;  GRANT EXECUTE ON FUNCTION GetAllCommentsToReport(reportId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetAllReportsByAuthor(authorId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetAllUsers() TO base\_user;  GRANT EXECUTE ON FUNCTION GetComment(commentId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetUserById(userId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetStatusHistoryToReport(reportId integer) TO base\_user;  GRANT EXECUTE ON PROCEDURE CreateComment(userId integer, reportId integer, commentText text) TO base\_user;  GRANT EXECUTE ON PROCEDURE CreateNewReport(authorId integer, title text, text text) TO base\_user;  GRANT EXECUTE ON PROCEDURE DeleteComment(commentId integer) TO base\_user;  GRANT EXECUTE ON PROCEDURE DeleteReport(reportId integer) TO base\_user; |

Листинг 2.8 – Роль BaseUser

Данный набор запросов GRANT предоставляет клиенту с именем "BaseUser" определенные права доступа к таблицам и функциям в базе данных. Клиенту предоставлены права SELECT, INSERT и UPDATE на таблицы "users" и "reports", право SELECT на таблицы "reports", "users" и "comments". Он также имеет право выполнения различных функций, таких как получение информации о пользователях, машинах, оборудовании, заказах и отзывах. Эти запросы GRANT обеспечивают пользователю возможность выполнять различные операции с данными и получать информацию из базы данных в соответствии с предоставленными правами доступа.

|  |
| --- |
| GRANT EXECUTE ON FUNCTION GetAllReports() TO user\_admin;  GRANT EXECUTE ON FUNCTION GetAllCommentsToReport(reportId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetAllReportsByAuthor(authorId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetAllUsers() TO user\_admin;  GRANT EXECUTE ON FUNCTION GetComment(commentId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetUserById(userId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetStatusHistoryToReport(reportId integer) TO user\_admin;  GRANT EXECUTE ON PROCEDURE CreateComment(userId integer, reportId integer, commentText text) TO user\_admin;  GRANT EXECUTE ON PROCEDURE CreateNewReport(authorId integer, title text, text text) TO user\_admin;  GRANT EXECUTE ON PROCEDURE DeleteComment(commentId integer) TO user\_admin;  GRANT EXECUTE ON PROCEDURE DeleteReport(reportId integer) TO user\_admin;  GRANT EXECUTE ON PROCEDURE DeleteUser(userId integer) TO user\_admin; |

Листинг 2.9 – Роль Manager

Приведенный набор запросов GRANT предоставляет менеджеру с именем "user\_admin" права SELECT, INSERT и UPDATE на таблицы "users", "report", "comment", "statushistory" . Он также имеет право выполнения различных функций, включая получение информации о пользователях, отчетах, комментариях, статусах. Эти запросы GRANT обеспечивают менеджеру полный доступ к управлению данными в базе данных.

## 2.8. Вывод

В данном разделе была рассмотрена разработка объектов базы данных для магазина автомобилей. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря хорошо спроектированным объектам базы данных система способна обеспечивать надежность, своевременное быстродействие и целостность данных.

# 3. Описание функций импорта и экспорта

Для таблицы Report в базе данных доступна функция экспорта и импорта данных в формате JSON. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в JSON, была разработана функция, результатом которой стало создание JSON файла, с записанными строками из определенной таблицы в JSON формат. Скрипт запроса представлен в листинге 3.1.

|  |
| --- |
| CREATE OR REPLACE FUNCTION ExportReportsToJson(filePath text)  RETURNS VOID  LANGUAGE plpgsql  AS $$  DECLARE  JSON\_DATA JSON;  BEGIN  BEGIN  SELECT JSON\_AGG(ROW\_TO\_JSON(REPORT)) INTO JSON\_DATA FROM REPORT;  PERFORM PG\_FILE\_WRITE(FILE\_PATH, JSON\_DATA::TEXT, true);  EXCEPTION WHEN OTHERS THEN  RAISE 'Произошла ошибка: %', SQLERRM;  END;  END;  $$ |

Листинг 3.1 – Функция ExportReportsToJson

Для обратного экспорта данные в JSON формат из файла, реализован запросом, который обращается к JSON файлу, и выбирает данные в обычные строки, это можно реализовать с помощью функции, либо по мере надобности записать данные во временную таблицу, либо таблицу в другой базе данных. Скрипт запроса представлен в [приложении Г](file:///C:\Users\User\Downloads\Telegram%20Desktop\Пояснительная_записка_Савчук_А_Ю_.docx#_Приложение_Г).

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах JSON, что делает управление базой данных более удобным и эффективным.

# 4. Тестирование производительности

Одной из ключевых задач в процессе разработки является тестирование производительности базы данных. Такое тестирование позволяет оценить способность базы данных обрабатывать запросы и возвращать результаты с высокой скоростью. Оценка производительности важна для определения эффективности базы данных и выявления возможных узких мест, которые могут вызывать задержки или проблемы в работе приложения.

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для данной задачи мы создали анонимный блок, и так как работаем именно с действующей базой данных, то можем использовать готовую функцию для вставки данных в таблицу. Разработанный анонимный блок позволяет добавить большое количество строк за одно выполнение (листинг 4.1).

|  |
| --- |
| CREATE OR REPLACE FUNCTION InsertValuesIntoReport()  RETURNS VOID  LANGUAGE plpgsql  AS $$  DECLARE  i integer := 1;  BEGIN  WHILE i <= 100000 LOOP  CALL createnewreport(nextval('userid')::integer, 'title is: ' || CAST(i AS text), 'text is: ' || CAST(i AS text));  i := i + 1;  END LOOP;  END;  $$ |

Листинг 4.1 – Заполнение таблицы 100000 строк

Для получения выборки данных использовался запрос, который представлен на листинге 4.2.

|  |
| --- |
| EXPLAIN ANALYZE SELECT text FROM REPORT WHERE text LIKE '%title is: 1%' |

Листинг 4.2 – Запрос к таблице Report

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составило 24.058 мс, а время планирования – 0.106 мс. Результаты запроса будут представлены на рисунке 5.1.

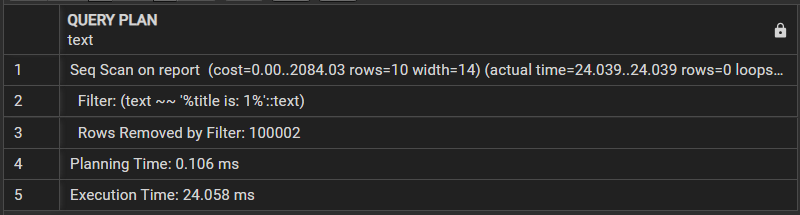


Рисунок 4.1 – Результат выполнения запроса

Для ускорения данного процесса создадим индекс на поле text так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 4.2.

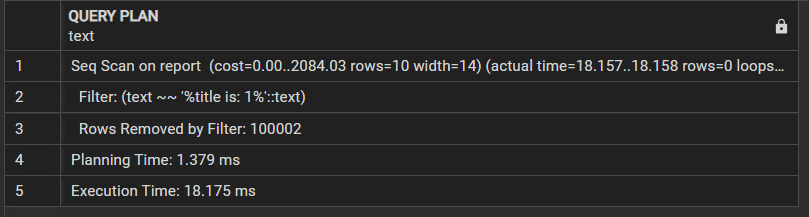


Рисунок 4.2 – Результат выполнения запроса

После создания индексов видно, что автомобили стали генерировать намного быстрее. Время планирования заняло 1.379 мс, а время выполнения 18.175 мс, что более чем в 1.5 раза быстрее.

Результаты тестирования говорят о том, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

# 5. Описание технологии и ее применение в базе данных

Технология email уведомлений при изменении базы данных (БД) обычно используется для автоматической отправки уведомлений на электронную почту при определенных событиях или изменениях данных в БД. Ниже представлен общий обзор этой технологии:

1. Механизм триггеров: В БД создаются триггеры, которые реагируют на определенные события, такие как вставка, обновление или удаление данных. Когда событие происходит, триггер активируется и запускает отправку уведомления.
2. Шаблоны уведомлений: Создаются шаблоны электронных писем, которые определяют структуру и содержание уведомления. В шаблонах можно использовать переменные для вставки конкретных данных из БД, таких как имена, значения полей и т. д.
3. Интеграция с почтовым сервером: Необходимо настроить соединение с почтовым сервером для отправки электронных писем. Это может быть SMTP-сервер (Simple Mail Transfer Protocol) или API-интеграция с почтовым провайдером.
4. Конфигурация правил уведомлений: Определяются правила, когда и какие уведомления должны быть отправлены. Например, можно настроить уведомление при добавлении новой записи, при изменении определенного поля или при определенном условии.
5. Обработка ошибок и безопасность: При разработке такой системы необходимо учесть обработку ошибок, чтобы предотвратить потерю уведомлений или повторную отправку. Также важно обеспечить безопасность данных, чтобы предотвратить несанкционированный доступ к информации в уведомлениях.
6. Мониторинг и отчетность: Желательно иметь возможность отслеживать отправленные уведомления и получать отчеты о доставке. Это помогает контролировать процесс и обнаруживать возможные проблемы или ошибки.
7. Для реализации технологии email уведомлений при изменении базы данных через язык С# можно использовать следующий подход:

|  |
| --- |
| public static class MailSender  {  private static readonly int \_smtpPort = 587;  private static readonly string \_smtpServer = "smtp.gmail.com";  private static readonly string \_smtpName = "artem9steponenko@gmail.com";  private static readonly string \_smtpPassword = "yutczhqunopqikqc";  public static void SendTableChangedMesssage(string tableName, string operation)  {  using (SmtpClient smtpClient = new SmtpClient(\_smtpServer, \_smtpPort))  {  smtpClient.Credentials = new NetworkCredential(\_smtpName, \_smtpPassword);  smtpClient.EnableSsl = true;  using (MailMessage mailMessage = new MailMessage())  {  mailMessage.From = new MailAddress(\_smtpName);  mailMessage.To.Add("reportandsolvenotificator@gmail.com");  mailMessage.Subject = "DB changed";  mailMessage.Body = $"Table {tableName} was {operation}";  try  {  smtpClient.Send(mailMessage);  }  catch (Exception ex)  {  }  }  }  }  } |

Листинг 5.1 – Email уведомление на C#

Код выполняет следующие действия:

1. Устанавливает соединение с базой данных PostgreSQL, используя модуль Entity framework c#. Для этого код указывает параметры подключения, такие как хост, имя базы данных, имя пользователя и пароль.
2. Определяет функцию `CreateUser `, которая добавляет нового пользователя в таблицу "Users" базы данных. Внутри функции выполняется SQL-запрос для вставки новой записи в таблицу "Users" с указанными значениями.
3. После добавления отзыва в базу данных, код создает объект `mailMessage ` для формирования электронного письма. Задаются заголовок, отправитель, получатель и тема письма.
4. Создается объект `SMTP` для установления соединения с сервером. Затем отправитель и получатель указываются в методе `SendTableChangedMessage`, и письмо преобразуется в строку и отправляется.
5. После отправки письма, соединение с SMTP-сервером закрывается.
6. Если добавление отзыва в базу данных не удалось, выводится сообщение об ошибке.

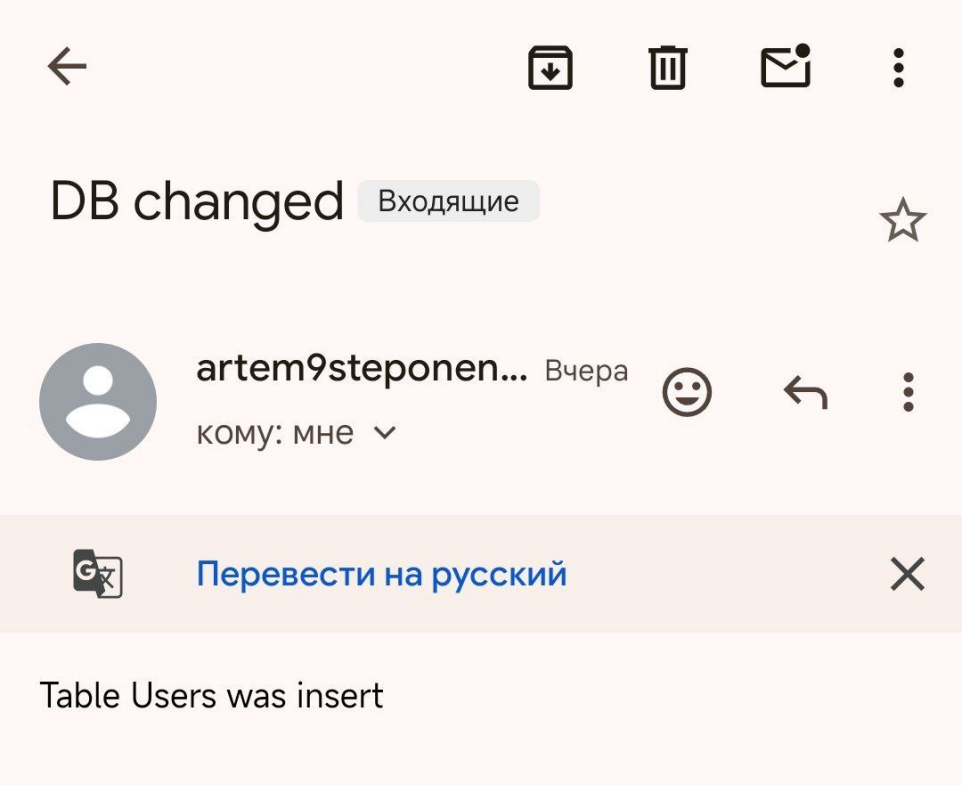


Рисунок 5.1 – новое письмо на почте.

# 6. Руководство пользователя

При входе в приложение пользователь будет иметь статус «Пользователь», следовательно – мы можем лишь посмотреть существующие отчеты, никак с ними не взаимодействуя, посмотреть профиль существующих пользователей и на этом функционал не авторизованного пользователя заканчивается.

* Права на выполнение операций SELECT на таблицу USERS.
* Права на выполнение операции SELECT, INSERT, UPDATE, DELETE на таблицу REPORT.
* Права на выполнение операций SELECT, UPDATE на таблицу STATUSHISTROY.
* Права на выполнение операций SELECT, INSERT, UPDATE, DELETE на таблицу COMMENT.

Если вход выполнен имени администратора, то тут к возможностям пользователя добавляется возможность редактирования и удаление чужих отчетов, удаление чужих комментариев, удаление пользователей.

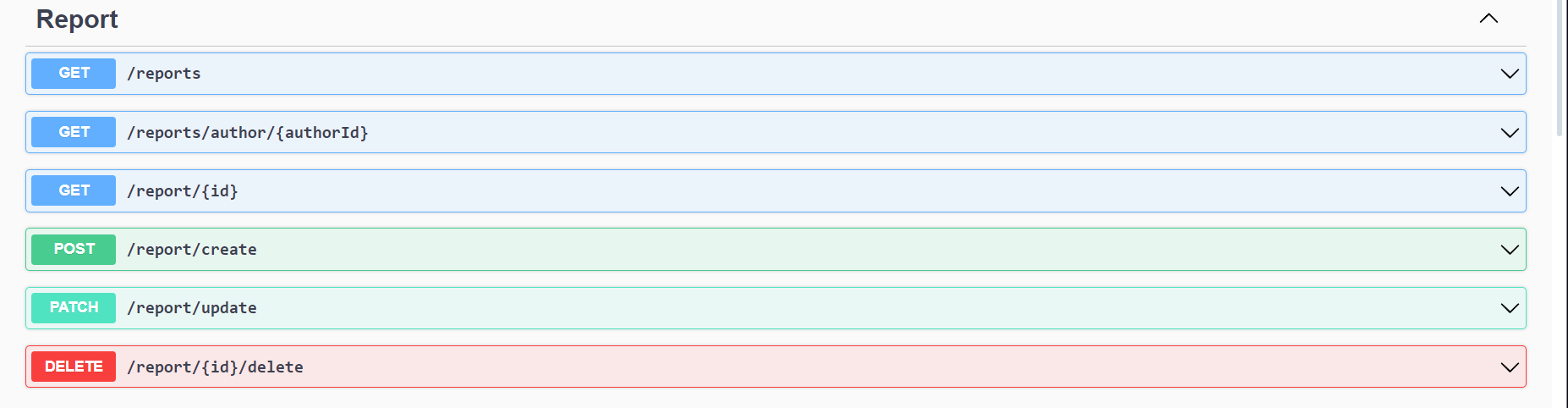
* Права на выполнение операций SELECT, UPDATE, DELETE на таблицу USERS.
* Права на выполнение операций SELECT, UPDATE, DELETE, INSERT на таблицу REPORT.
* Права на выполнение операций SELECT, UPDATE, DELETE, INSERT на таблицу COMMENT.
* Права на выполнение операций SELECT, UPDATE на таблицу STATUSHISTORY.

# Краткое описание приложения для демонстрации

## Пример работы приложения от лица администратора

Для администратора доступны все возможность api для взаимодействия с базой данных. При входе от лица администратора нам доступно множество методов, но мы рассмотрим лишь основные:

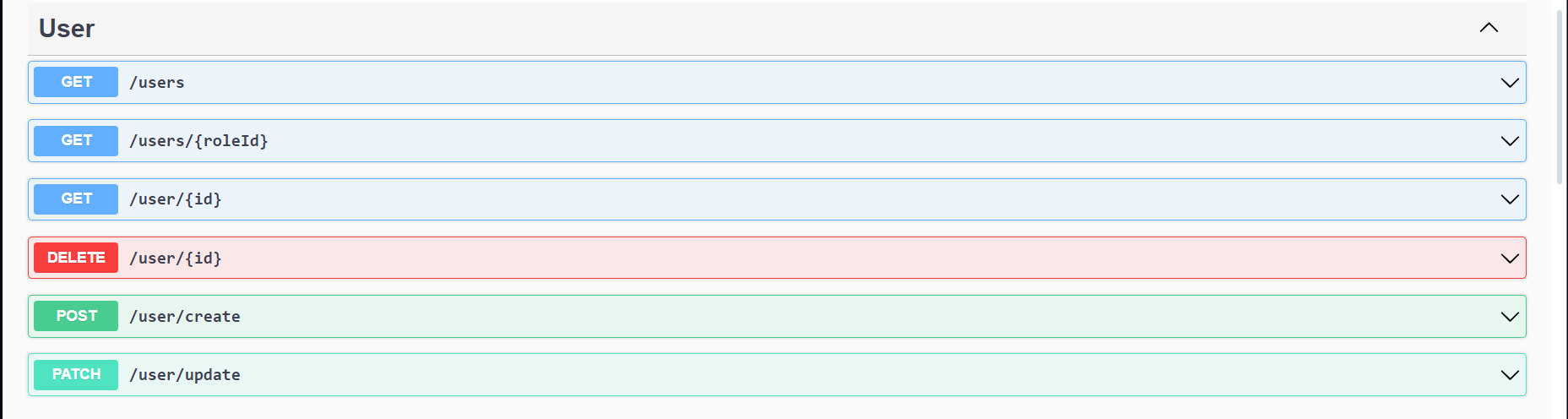
* /reports – возвращает список всех существующих отчетов.
* /report/create – создает новый отчет.
* /report/{id}/delete – удаление существующего отчета.
* /report/update – обновление существующего отчета.



Листинг 7.1.1 – демонстрация основных методов доступа к таблице Report.

Данная часть api отвечает за взаимодействие с таблицей отчетов путем вызова хранимых функций и процедур. Тут мы можем выполнять все операции CRUD, по отношению к таблице Report.

* /users – возвращает список всех существующих отчетов.
* /user/{id} – возвращает пользователя с заданным идентификатором.
* /user/create – создает нового пользователя.
* /user/delete – удаление существующего пользователя.
* /user/update – обновление существующего пользователя.



Листинг 7.1.1 – демонстрация основных методов доступа к таблице Users.

Данная часть api предназначена для взаимодействия с таблицей пользователей, тут также реализованы основные CRUD запросы, которые позволяют взаимодействовать с таблицей пользователей.

С помощью этого api в будущем возможно будет разработать приложение, которое поможет решать проблемы в реальном времени, обсуждая ошибки с другими пользователями.

Все функции были разработаны в соответствии потребностям пользователей, что делает каждый запрос интуитивно понятным, и не сложным в освоении. Помимо этого, т.к предусмотрено большое количество схем представления данных, что дает возможность не привязываться к одному типу данных, что в свою очередь делает исходный код не таким связным и более масшатбируемым.

Это небольшой пример того, что есть в api для взаимодействия с базой данных. Помимо основных запросов для взаимодействия с таблицами пользователей и отчетов, предусмотрены запросы и для работы с таблицами комментариев и истории статусов.

# Заключение

В данном проекте была поставлена задача разработки базы данных для магазина автомобилей с использованием технологии разработки базы данных с системой e-mail-уведомлений о событиях в своей базе данных.

В ходе выполнения проекта были использованы различные объекты, такие как таблицы, представления, процедуры, индексы триггеры, функции, специальные функции, чтобы обеспечить структурированное хранение данных и удобный доступ к ним. В итоге, поставленная цель была успешно достигнута, и на выходе получена готовая база данных.

В рамках тестирования базы данных был использован поток данных, и результаты тестирования оказались положительными. Также были разработаны процедуры импорта и экспорта данных в формате JSON, что значительно облегчило управление данными и повысило удобство использования базы данных.

Разработанная база данных для магазина автомобилей имеет ряд важных особенностей, которые позволяют эффективно управлять ей. Одной из главных особенностей является технологии разработки базы данных с системой e-mail-уведомлений о событиях в своей базе данных, которая обеспечивает отправку данных в случае любых изменений или ошибок.

Кроме того, база данных обладает гибкой системой привилегий, которая позволяет разграничивать доступ к различным функциям и данным в зависимости от роли пользователя. Это обеспечивает безопасность данных и предотвращает несанкционированный доступ.

Делая вывод, можно сказать, что поставленные цели были выполнены. База данных для системы отчетов об ошибках выполняет необходимую функцию хранения и управления данными.

# Список используемых источников

1. PostgreSQL Сайт с информацией [Электронный ресурс] – https://postgrespro.ru/docs/postgresql.com/ – Дата доступа 10.04.2024
2. Quora [Электронный ресурс] – https://www.quora.com– Дата доступа 27.04.2024
3. StackOverflow[Электронный ресурс] – https://stackoverflow.com/ – Дата доступа 27.04.2024
4. Технология email рассылки в PostgreSQL [Электронный ресурс] – https://postgrespro.ru/docs/postgresql/9.6/backup-dump/ – Дата доступа 13.04.2024
5. Email рассылка с помощью C# [Электронный ресурс] - <https://ru.stackoverflow.com/questions/705300/Как-с-помощью-класса-smtpclient-отправить-почту-на-яндекс> - Дата доступа 18.05.2024

# Приложения А – код создания таблиц

|  |
| --- |
| CREATE TABLE ROLE  (  ID serial PRIMARY KEY,  Name varchar(255)  )  DROP TABLE ROLE;  CREATE TABLE USERS  (  ID serial PRIMARY KEY,  Role serial REFERENCES ROLE(ID),  Name varchar(255),  Mail text UNIQUE,  Password text NOT NULL  )  CREATE TABLE COMMENT  (  ID serial PRIMARY KEY,  Author serial REFERENCES USERS(ID),  Report serial REFERENCES REPORT(ID),  Text text  )  DROP TABLE COMMENT;  CREATE TABLE STATUSHISTORY  (  ID serial PRIMARY KEY,  Report serial REFERENCES REPORT(ID),  Statuses text[]  )  DROP TABLE STATUSHISTORY;  CREATE TABLE REPORT  (  ID serial PRIMARY KEY,  Author serial REFERENCES USERS(ID),  Text text,  Titile text  )  DROP TABLE REPORT; |

Листинг 1 – создание таблиц

# Приложение Б – код создания функций

|  |
| --- |
| -- Reports --  CREATE OR REPLACE FUNCTION GetAllReports()  RETURNS SETOF PUBLIC.REPORT  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.REPORT;  END;  $$  DROP FUNCTION GetAllReports;  CREATE OR REPLACE FUNCTION GetAllReportsByAuthor(authorId integer)  RETURNS SETOF PUBLIC.REPORT  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.REPORT  WHERE "author" = authorId;  END;  $$  DROP FUNCTION GetAllReportsByAuthor;  CREATE OR REPLACE FUNCTION GetReportByID(reportId integer)  RETURNS SETOF PUBLIC.REPORT  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.REPORT  WHERE "id" = reportId;  END;  $$  DROP FUNCTION GetReportByID;  -- Users --  CREATE OR REPLACE FUNCTION GetAllUsers()  RETURNS SETOF PUBLIC.USERS  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.USERS;  END;  $$  DROP FUNCTION GetAllUsers;  CREATE OR REPLACE FUNCTION GetAllUsersWithRole(roleId integer)  RETURNS SETOF PUBLIC.USERS  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.USERS  WHERE "role" = roleId;  END;  $$  DROP FUNCTION GetAllUsersWithRole;  CREATE OR REPLACE FUNCTION GetUserByID(userId integer)  RETURNS SETOF PUBLIC.USERS  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.USERS  WHERE "id" = userId;  END;  $$  DROP FUNCTION GetUserByID;  -- Comments --  CREATE OR REPLACE FUNCTION GetAllCommentsToReport(reportId integer)  RETURNS SETOF PUBLIC.COMMENT  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.COMMENT  WHERE "report" = reportId;  END;  $$  DROP FUNCTION GetAllCommentsToReport;  CREATE OR REPLACE FUNCTION GetComment(commentId integer)  RETURNS SETOF PUBLIC.COMMENT  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.COMMENT  WHERE "id" = commentId;  END;  $$  DROP FUNCTION GetComment;  CREATE OR REPLACE FUNCTION DeleteComment()  RETURNS TRIGGER  LANGUAGE plpgsql  AS $$  BEGIN  DELETE FROM COMMENT WHERE Report = OLD.ID;  RETURN OLD;  END;  $$  DROP FUNCTION DeleteComment;  -- StatusHistory  CREATE OR REPLACE FUNCTION GetStatusHistoryToReport(reportId integer)  RETURNS SETOF PUBLIC.ATUSHISTORY  LANGUAGE plpgsql  AS $$  BEGIN  RETURN QUERY SELECT \* FROM PUBLIC.STATUSHISTORY  WHERE "report" = reportId;  END;  $$  DROP FUNCTION GetStatusHistoryToReport;  CREATE OR REPLACE FUNCTION CreateStatusHistory()  RETURNS TRIGGER  LANGUAGE plpgsql  AS $$  BEGIN  INSERT INTO STATUSHISTORY (Report, Statuses) VALUES (NEW.ID, '{}');  RETURN NEW;  END;  $$  DROP FUNCTION CreateStatusHistory;  CREATE OR REPLACE FUNCTION DeleteStatusHistory()  RETURNS TRIGGER  LANGUAGE plpgsql  AS $$  BEGIN  DELETE FROM STATUSHISTORY WHERE Report = OLD.ID;  RETURN OLD;  END;  $$  DROP FUNCTION DeleteStatusHistory;  CREATE OR REPLACE FUNCTION ExportReportsToJson(filePath text)  RETURNS VOID  LANGUAGE plpgsql  AS $$  DECLARE  JSON\_DATA JSON;  BEGIN  BEGIN  SELECT JSON\_AGG(ROW\_TO\_JSON(REPORT)) INTO JSON\_DATA FROM REPORT;  PERFORM PG\_FILE\_WRITE(filePath, JSON\_DATA::TEXT, true);  EXCEPTION WHEN OTHERS THEN  RAISE 'Произошла ошибка: %', SQLERRM;  END;  END;  $$  SELECT \* FROM ExportReportsToJson('D:\Labs\BDKyrsach\data.json');  DELETE FROM Report  DROP FUNCTION ExportReportsToJson;  CREATE OR REPLACE FUNCTION IMPORT\_REPORTS\_FROM\_JSON\_FILE(FILE\_PATH TEXT)  RETURNS TABLE (  ID INTEGER,  Author INTEGER,  Text TEXT,  Title TEXT  ) AS $$  DECLARE  FILE\_CONTENT TEXT;  JSON\_DATA JSON;  REPORT\_DATA JSON;  BEGIN  CREATE TEMP TABLE IF NOT EXISTS temp\_reports (  ID INTEGER,  Author INTEGER,  Text TEXT,  Title TEXT  );  DELETE FROM temp\_reports;  BEGIN  FILE\_CONTENT := pg\_read\_file(FILE\_PATH, 0, 1000000000);  EXCEPTION WHEN OTHERS THEN  RAISE 'Файл не найден: %', FILE\_PATH;  END;  BEGIN  JSON\_DATA := FILE\_CONTENT::JSON;  EXCEPTION WHEN OTHERS THEN  RAISE 'Некорректный JSON: %', SQLERRM;  END;  FOR REPORT\_DATA IN SELECT \* FROM json\_array\_elements(JSON\_DATA)  LOOP  IF NOT (REPORT\_DATA::jsonb ? 'id' AND REPORT\_DATA::jsonb ? 'author' AND REPORT\_DATA::jsonb ? 'text' AND REPORT\_DATA::jsonb ? 'title') THEN  CONTINUE;  END IF;  INSERT INTO temp\_reports (ID, Author, Text, Title)  VALUES (  (REPORT\_DATA->>'id')::INTEGER,  (REPORT\_DATA->>'author')::INTEGER,  REPORT\_DATA->>'text',  REPORT\_DATA->>'title'  );  END LOOP;  RETURN QUERY SELECT \* FROM temp\_reports;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 1 – функции, связанные с таблицами

# Приложение В – код создания процедур

|  |
| --- |
| -- Reports --  CREATE OR REPLACE PROCEDURE CreateNewReport(authorId integer, title text, text text)  LANGUAGE plpgsql  AS $$  DECLARE  lastId integer;  BEGIN  INSERT INTO PUBLIC.REPORT(Author, Title, Text) VALUES (authorId, title, text);  END;  $$  DROP PROCEDURE CreateNewReport;  CREATE OR REPLACE PROCEDURE UpdateReport(reportId integer, newTitle text, newText text)  LANGUAGE plpgsql  AS $$  BEGIN  UPDATE REPORT  SET "title" = newTitle, "text" = newText  WHERE REPORT.id = reportId;  END;  $$  DROP PROCEDURE UpdateReport;  CREATE OR REPLACE PROCEDURE DeleteReport(reportId integer)  LANGUAGE plpgsql  AS $$  BEGIN  DELETE FROM REPORT  WHERE "id" = reportId;  END;  $$  DROP PROCEDURE DeleteReport;  -- StatusHistory --  CREATE OR REPLACE PROCEDURE CreateStatusHistory(reportId integer)  LANGUAGE plpgsql  AS $$  BEGIN  INSERT INTO STATUSHISTORY(Report, Statuses) VALUES (reportId, '{}');  END;  $$  DROP PROCEDURE CreateStatusHistory;  CREATE OR REPLACE PROCEDURE UpdateStatusHistory(reportId integer, newStatus text)  LANGUAGE plpgsql  AS $$  DECLARE  previousStatuses text[];  BEGIN  SELECT statuses  INTO previousStatuses  FROM STATUSHISTORY  WHERE "report" = reportId;    previousStatuses := previousStatuses || newStatus;    UPDATE STATUSHISTORY  SET "statuses" = previousStatuses  WHERE "report" = reportId;  END;  $$  DROP PROCEDURE UpdateStatusHistory  -- Users --  CREATE OR REPLACE PROCEDURE CreateNewUser(roleId integer, name text, mail text, password text)  LANGUAGE plpgsql  AS $$  BEGIN  INSERT INTO USERS(Role, Name, Mail, Password) VALUES(roleId, name, mail, password);  END;  $$  DROP PROCEDURE CreateNewUser;  CREATE OR REPLACE PROCEDURE UpdateUser(userId integer, newRoleId integer, newName text, newMail text, newPassword text)  LANGUAGE plpgsql  AS $$  BEGIN  UPDATE USERS  SET "role" = newRoleId, "name" = newName, "mail" = newMail, "password" = newPassword  WHERE "id" = userId;  END;  $$  DROP PROCEDURE UpdateUser;  CREATE OR REPLACE PROCEDURE DeleteUser(userId integer)  LANGUAGE plpgsql  AS $$  BEGIN  DELETE FROM STATUSHISTORY  WHERE "report" IN (SELECT Id FROM REPORT WHERE "author" = userId);    DELETE FROM COMMENT  WHERE "author" = userId;  DELETE FROM REPORT  WHERE "author" = userId;  DELETE FROM Users  WHERE "id" = userId;  END;  $$  DROP PROCEDURE DeleteUser;  -- Comment --  CREATE OR REPLACE PROCEDURE CreateComment(userId integer, reportId integer, commentText text)  LANGUAGE plpgsql  AS $$  BEGIN  INSERT INTO PUBLIC."comment"(author, report, text) VALUES(userId, reportId, commentText);  END;  $$  DROP PROCEDURE CreateComment;  CREATE OR REPLACE PROCEDURE DeleteComment(commentId integer)  LANGUAGE plpgsql  AS $$  BEGIN  DELETE FROM PUBLIC.COMMENT WHERE "id" = commentId;  END;  $$  DROP PROCEDURE DeleteComment; |

Листинг 1 – процедуры, связанные с таблицами

# Приложение Г – код реализации импорта и экспорта

|  |
| --- |
| CREATE OR REPLACE FUNCTION ExportReportsToJson(filePath text)  RETURNS VOID  LANGUAGE plpgsql  AS $$  DECLARE  JSON\_DATA JSON;  BEGIN  BEGIN  SELECT JSON\_AGG(ROW\_TO\_JSON(REPORT)) INTO JSON\_DATA FROM REPORT;  PERFORM PG\_FILE\_WRITE(filePath, JSON\_DATA::TEXT, true);  EXCEPTION WHEN OTHERS THEN  RAISE 'Произошла ошибка: %', SQLERRM;  END;  END;  $$  CREATE OR REPLACE FUNCTION IMPORT\_REPORTS\_FROM\_JSON\_FILE(FILE\_PATH TEXT)  RETURNS TABLE (  ID INTEGER,  Author INTEGER,  Text TEXT,  Title TEXT  ) AS $$  DECLARE  FILE\_CONTENT TEXT;  JSON\_DATA JSON;  REPORT\_DATA JSON;  BEGIN  CREATE TEMP TABLE IF NOT EXISTS temp\_reports (  ID INTEGER,  Author INTEGER,  Text TEXT,  Title TEXT  );  DELETE FROM temp\_reports;  BEGIN  FILE\_CONTENT := pg\_read\_file(FILE\_PATH, 0, 1000000000);  EXCEPTION WHEN OTHERS THEN  RAISE 'Файл не найден: %', FILE\_PATH;  END;  BEGIN  JSON\_DATA := FILE\_CONTENT::JSON;  EXCEPTION WHEN OTHERS THEN  RAISE 'Некорректный JSON: %', SQLERRM;  END;  FOR REPORT\_DATA IN SELECT \* FROM json\_array\_elements(JSON\_DATA)  LOOP  IF NOT (REPORT\_DATA::jsonb ? 'ID' AND REPORT\_DATA::jsonb ? 'Author' AND REPORT\_DATA::jsonb ? 'Text' AND REPORT\_DATA::jsonb ? 'Title') THEN  CONTINUE;  END IF;  INSERT INTO temp\_reports (ID, Author, Text, Title)  VALUES (  DEFAULT,  (REPORT\_DATA->>'Author')::INTEGER,  REPORT\_DATA->>'Text',  REPORT\_DATA->>'Title'  );END LOOP;  RETURN QUERY SELECT \* FROM temp\_reports;  END;  $$ LANGUAGE PLPGSQL;  SELECT \* FROM IMPORT\_REPORTS\_FROM\_JSON\_FILE('D:\Games\test.json'); |

Листинг 1 - импорт данных из json-файла

# Приложение Д – код создание представлений

|  |
| --- |
| CREATE OR REPLACE VIEW UsersWithRole AS  SELECT public.role.name AS roleName, public.users.name AS userName  from users inner join role on role.id = users.role  SELECT \* FROM UsersWithRole;  DROP VIEW UsersWithRole;  CREATE OR REPLACE VIEW AuthorizationUserData AS  SELECT name, mail, password FROM USERS;  SELECT \* FROM AuthorizationUserData;  DROP VIEW AuthorizationUserData;  CREATE OR REPLACE VIEW ReportsWithAuthors AS  SELECT  public.report.ID AS report\_id,  public.report.Title AS report\_title,  public.report.Text AS report\_text,  public.users.Name AS author\_name  FROM REPORT  JOIN USERS ON public.report.Author = public.users.ID;  SELECT \* FROM ReportsWithAuthors;  DROP VIEW ReportsWithAuthors;  CREATE OR REPLACE VIEW StatusHistoryWithReportInfo AS  SELECT  public.statushistory.ID AS status\_history\_id,  public.report.Title AS report\_title,  public.statushistory.Statuses  FROM STATUSHISTORY  JOIN REPORT ON public.statushistory.Report = public.report.ID;  DROP VIEW StatusHistoryWithReportInfo; |

Листинг 1 – код создания представлений.

# Приложение Е – код создания триггеров

|  |
| --- |
| -- Report --  CREATE TRIGGER CreateStatusHistoryTrigger  AFTER INSERT ON REPORT  FOR EACH ROW  EXECUTE FUNCTION CreateStatusHistory();    DROP TRIGGER CreateStatusHistoryTrigger ON REPORT;    CREATE TRIGGER DeleteStatusHistoryTrigger  BEFORE DELETE ON REPORT  FOR EACH ROW  EXECUTE FUNCTION DeleteStatusHistory();    DROP TRIGGER DeleteStatusHistoryTrigger ON REPORT;    CREATE TRIGGER DeleteCommentsTrigger  BEFORE DELETE ON REPORT  FOR EACH ROW  EXECUTE FUNCTION DeleteComment();    DROP TRIGGER DeleteCommentsTrigger ON REPORT; |

Листинг 1 – код создания триггеров.

# Приложение Ё – код создания пользователей и ролей

|  |
| --- |
| CREATE ROLE BaseUser;  GRANT SELECT, INSERT, UPDATE ON REPORT TO BaseUser;  GRANT SELECT ON USERS TO BaseUser;  GRANT SELECT, UPDATE ON STATUSHISTORY TO BaseUser;  GRANT SELECT, INSERT, UPDATE, DELETE ON COMMENT TO BaseUser;  DROP ROLE BaseUser;  CREATE ROLE UserAdmin;  GRANT SELECT, INSERT, UPDATE, DELETE ON REPORT TO UserAdmin;  GRANT SELECT, UPDATE, DELETE ON USERS TO UserAdmin;  GRANT SELECT, UPDATE ON STATUSHISTORY TO UserAdmin;  GRANT SELECT, INSERT, UPDATE, DELETE ON COMMENT TO UserAdmin;  DROP ROLE UserAdmin;  CREATE USER user\_admin WITH LOGIN PASSWORD '1234';  GRANT UserAdmin TO user\_admin;  CREATE USER base\_user WITH LOGIN PASSWORD '12345';  GRANT BaseUser TO base\_user;  -- grant to base user  GRANT EXECUTE ON FUNCTION GetAllReports() TO base\_user;  GRANT EXECUTE ON FUNCTION GetAllCommentsToReport(reportId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetAllReportsByAuthor(authorId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetAllUsers() TO base\_user;  GRANT EXECUTE ON FUNCTION GetComment(commentId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetUserById(userId integer) TO base\_user;  GRANT EXECUTE ON FUNCTION GetStatusHistoryToReport(reportId integer) TO base\_user;  GRANT EXECUTE ON PROCEDURE CreateComment(userId integer, reportId integer, commentText text) TO base\_user;  GRANT EXECUTE ON PROCEDURE CreateNewReport(authorId integer, title text, text text) TO base\_user;  GRANT EXECUTE ON PROCEDURE DeleteComment(commentId integer) TO base\_user;  GRANT EXECUTE ON PROCEDURE DeleteReport(reportId integer) TO base\_user;  -- grants to admin user  GRANT EXECUTE ON FUNCTION GetAllReports() TO user\_admin;  GRANT EXECUTE ON FUNCTION GetAllCommentsToReport(reportId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetAllReportsByAuthor(authorId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetAllUsers() TO user\_admin;  GRANT EXECUTE ON FUNCTION GetComment(commentId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetUserById(userId integer) TO user\_admin;  GRANT EXECUTE ON FUNCTION GetStatusHistoryToReport(reportId integer) TO user\_admin;  GRANT EXECUTE ON PROCEDURE CreateComment(userId integer, reportId integer, commentText text) TO user\_admin;  GRANT EXECUTE ON PROCEDURE CreateNewReport(authorId integer, title text, text text) TO user\_admin;  GRANT EXECUTE ON PROCEDURE DeleteComment(commentId integer) TO user\_admin;  GRANT EXECUTE ON PROCEDURE DeleteReport(reportId integer) TO user\_admin;  GRANT EXECUTE ON PROCEDURE DeleteUser(userId integer) TO user\_admin; |

Листинг 1 – код создания пользователей и ролей.

# Приложение Ж – код создания индексов.

|  |
| --- |
| CREATE INDEX idx\_text ON Report (text);  DROP INDEX idx\_text;  CREATE INDEX idx\_title ON Report (title);  DROP INDEX idx\_title;  CREATE INDEX idx\_name ON Users (name);  DROP INDEX idx\_name;  CREATE INDEX idx\_comment\_text ON Comment (text); |

Листинг 1 – код создания индексов.