

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

Факультет Информационных технологий  
Кафедра Информационных систем и технологий  
Специальность 6-05-0611-01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»  
Тема «База данных новостного канала»

**Исполнитель**

студента 2 курса 1 группы

\_\_\_\_\_  
подпись, дата

К.С. Гацевич

**Руководитель**

Ассистент

должность, учен. степень, ученое звание

\_\_\_\_\_  
подпись, дата

М. Г. Савельева

Допущена к защите \_\_\_\_\_

\_\_\_\_\_  
дата, подпись

Курсовой проект защищен с оценкой \_\_\_\_\_

Руководитель \_\_\_\_\_  
подпись

\_\_\_\_\_  
дата

М. Г. Савельева  
инициалы и фамилия



## Содержание

Введение .....	4
1 Постановка задачи .....	5
1.1 Обзор аналогичных решений .....	5
1.2 Требования к проекту .....	9
1.3 Вывод по разделу .....	10
2 Проектирование и разработка базы данных .....	11
2.1 Определение вариантов использования .....	11
2.2 Диаграммы UML, взаимодействие всех компонентов .....	15
2.3 Вывод по разделу .....	15
3 Разработка объектов базы данных .....	16
3.1 Разработка таблиц базы данных .....	16
3.2 Разработка схем базы данных .....	17
3.3 Разработка процедур и функций базы данных .....	17
3.4 Разработка функций базы данных .....	18
3.5 Разработка представлений базы данных .....	18
3.6 Разработка триггеров базы данных .....	18
3.7 Роли и пользователи .....	18
3.8 Вывод по разделу .....	18
4 Описание процедур экспорта и импорта данных .....	20
4.1 Процедура импорта данных из JSON-файла .....	20
4.2 Процедура экспорта данных из JSON-файла .....	21
4.3 Вывод по разделу .....	22
5 Тестирование производительности .....	23
5.1 Заполнение таблицы .....	23
5.2 Тестирование производительности базы данных .....	23
5.3 Вывод по разделу .....	24
6 Описание технологии и её применение в базе данных .....	25
6.1 Технология отказоустойчивый кластер PostgreSQL .....	25
6.2 Вывод по разделу .....	25
Заключение .....	26
Список используемых источников .....	27
Приложение А .....	28
Приложение Б .....	32

## Введение

Современные онлайн-платформы сталкиваются с необходимостью обработки растущих объемов контента и пользовательских взаимодействий. Разрабатываемая база данных для платформы публикации статей призвана решить эту проблему, обеспечив надежное хранение информации и быстрый доступ к ней. В основе проекта лежит тщательный анализ требований к системе, включающий три ключевые роли пользователей с разным уровнем доступа и функционалом:

- Администратор: управление базой данных, назначение и контроль модераторов;
- Модератор: обработка жалоб пользователей, контроль контента площадки;
- Пользователь: просмотр и поиск контента, взаимодействие через комментарии и оценки, создание и управление публикациями, оставление жалоб на статьи и пользователей.

Цель курсового проекта – разработка базы данных PostgreSQL.

К задачам курсового проекта относится: изучение требований; определение вариантов использования; анализ и проектирование модели данных; описание информационных объектов и ограничений целостности; создание необходимых объектов; импорт и экспорт данных; реализация технологии отказоустойчивого кластера; тестирование производительности; формирование вывода по каждому разделу; заключение, включающее вывод по проделанной работе.

Система управления базами данных (СУБД) играет ключевую роль в хранении и обработке информации, являясь основным связующим звеном между пользователями и сервисом. Она обеспечивает структурированное хранение данных, удобный доступ к ним, эффективное управление и поддерживает функции мониторинга, настройки и резервного копирования.

Выбор PostgreSQL в качестве СУБД обусловлен его надежностью, богатым функционалом и возможностями тонкой настройки под конкретные задачи платформы. Основное внимание уделено проектированию оптимальной структуры данных, учитывающей взаимосвязи между статьями, комментариями, оценками и пользователями.

## 1 Постановка задачи

### 1.1 Обзор аналогичных решений

#### 1.1.1 Аналог «Habr»

Habr представляет собой платформу для публикации IT-статей с системой тегов, рейтингов и комментариев, что делает его близким по структуре к тематическому блогу [1].

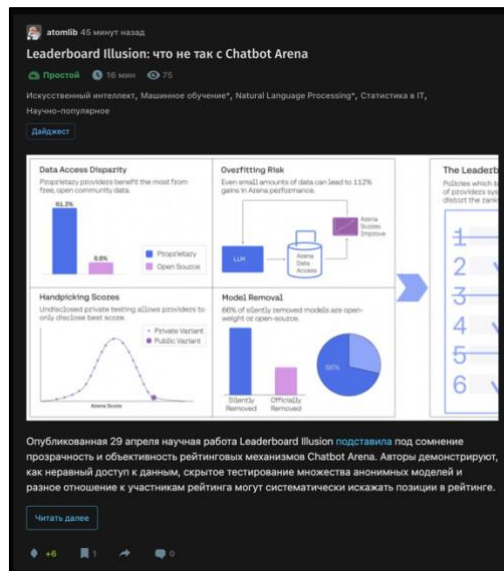


Рисунок 1.1 – Статья в каталоге на сайте «Habr»

Также каждый зарегистрированный пользователь имеет профиль, который содержит его личную информацию.

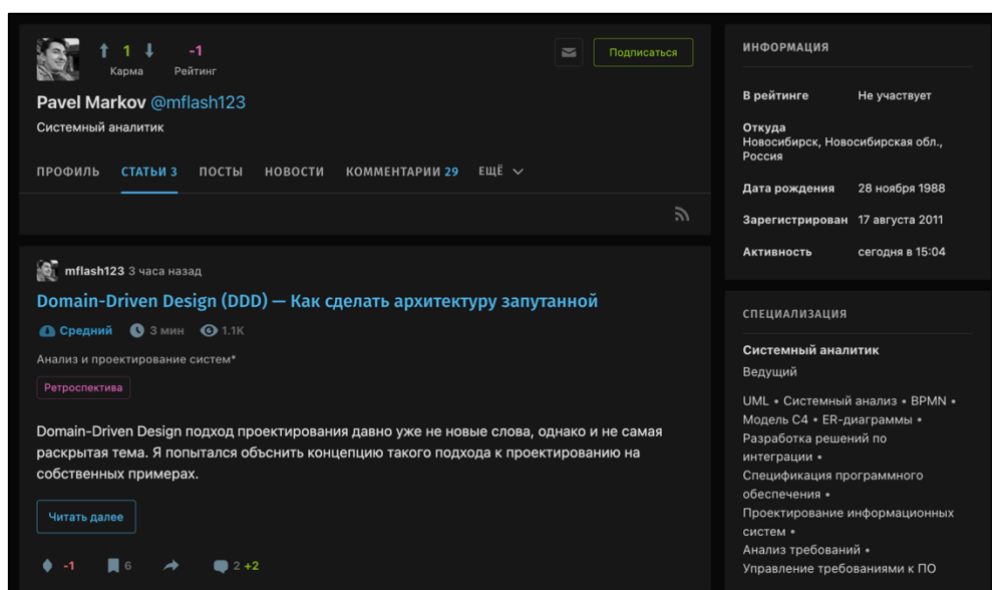


Рисунок 1.2 – Профиль пользователя на сайте «Habr»

Исходя из функционала данного аналога, были определены следующие таблицы базы данных:

Таблица «Статьи»: содержит информацию о публикациях на сайте. Включает уникальный идентификатор статьи (ID) ссылку на автора (ID пользователя), заголовок, краткое описание, полный текст материала, принадлежность к категории (ID категории), статус публикации, а также метки времени создания и последнего обновления.

Таблица «Теги»: содержит перечень ключевых слов для маркировки и классификации статей. Включает уникальный идентификатор тега (ID) и его название (name).

Таблица «Пользователи»: хранит данные зарегистрированных на портале лиц. Включает уникальный идентификатор (ID), имя пользователя, электронную почту, зашифрованный пароль, дату и время последнего входа (last\_login) и дату регистрации (created\_at).

Таблица «Категории»: содержит список разделов для классификации статей. Включает уникальный идентификатор категории (ID) и её название (name).

Таблица «Подписки»: хранит сведения о том, какие пользователи следят за другими.

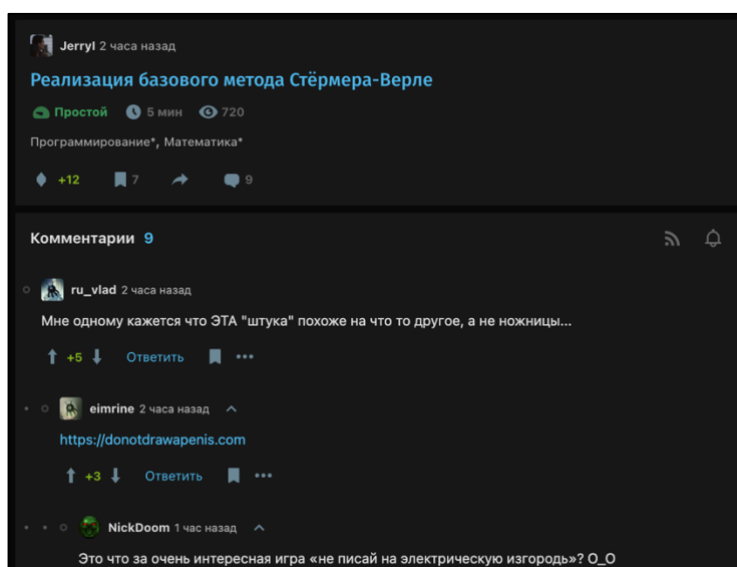


Рисунок 1.3 – Комментарии под статьёй «Habr»

Таблица «Комментарии»: хранит информацию об отзывах пользователей к статьям: каждая запись имеет уникальный идентификатор (id), ссылается на статью (article\_id) и пользователя-автора комментария (user\_id), содержит текст комментария (content) и дату и время его создания (created\_at, по умолчанию текущее время).

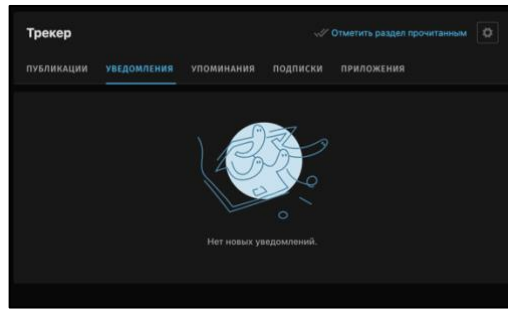


Рисунок 1.4 – Трекер уведомлений «Habr»

Таблица «Уведомления»: хранит данные о системных сообщениях для пользователей: каждая запись имеет уникальный идентификатор (id), ссылку на получателя уведомления, текст уведомления, флаг прочтения и отметку времени создания.

### 1.1.2 Аналог «StackOverFlow»

Stack Overflow — международная платформа вопросов и ответов для IT-специалистов и разработчиков, где пользователи публикуют технические вопросы и получают решения, подкреплённые примерами кода и ссылками на документацию. Контент структурируется метками, а система голосов, репутации и бейджей стимулирует сообщество создавать качественные материалы и модерировать базу знаний[2].

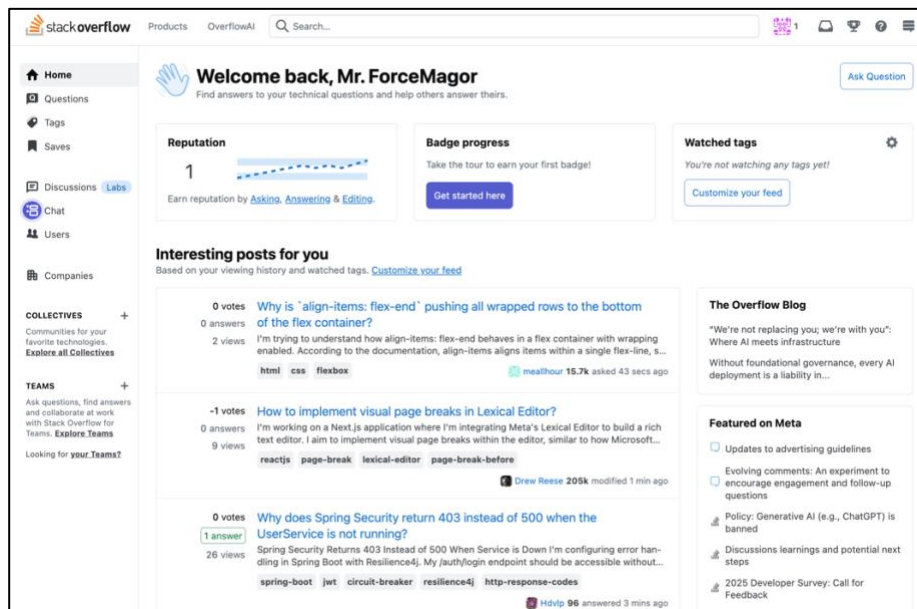


Рисунок 1.5 – Главная страница сайта «Stack Overflow»

Основная идея сервиса — обеспечить оперативный и качественный обмен техническими знаниями между разработчиками через платформу вопросов и ответов, где лучшие решения выделяются голосами сообщества.

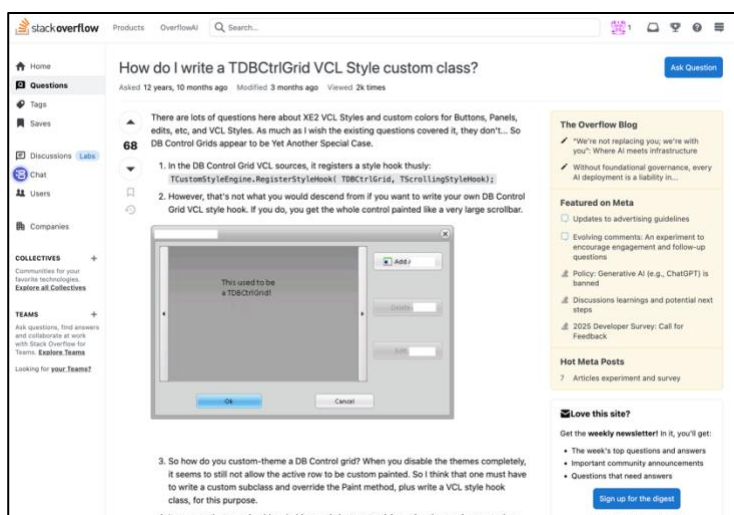


Рисунок 1.6 – Вопрос на сайте «Stack Overflow»

Stack Overflow обеспечивает быстрый и прозрачный обмен техническими знаниями: система меток позволяет точно классифицировать контент по языкам и технологиям, а встроенные голосование и комментарии помогают сообществу выявлять лучшие решения и оперативно уточнять детали.

База данных данного аналога практически идентична предыдущему аналогу. Так же, как и у прошлого аналога, здесь присутствуют таблицы «Вопросы» (аналог таблицы «Статьи»), «Теги», «Пользователи», «Комментарии», «Отзывы», «Жалобы», и «Уведомления».

### 1.1.3 Аналог «Techcrunch»

TechCrunch — это международная медиа-платформа. Основная идея платформы заключается в предоставлении пользователям актуальной информации о развитии IT-индустрии, инновациях, продуктах крупных технологических компаний и деятельности новых игроков рынка [3].

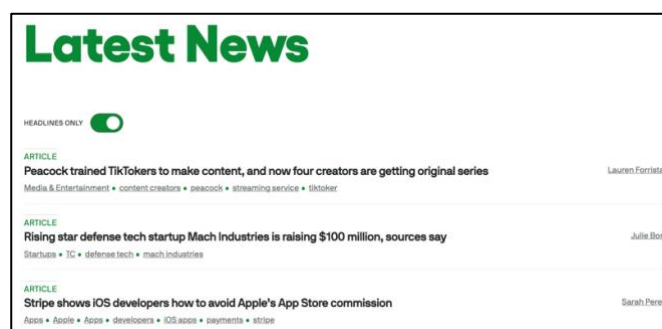


Рисунок 1.7 – Каталог новостей на сайте «TechCrunch»

Пользователь может просматривать статьи, фильтруя их по тегам и категориям.



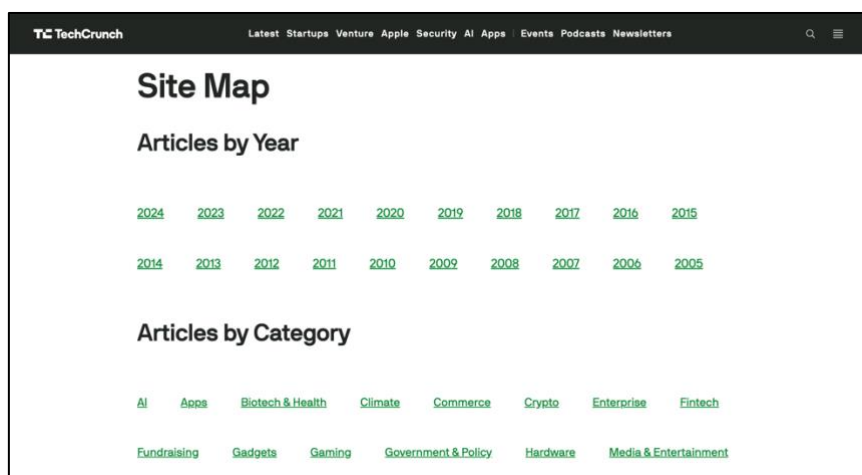


Рисунок 1.8 – Карта сайта «TechCrunch»

Данный аналог имеет базу данных аналогично описанной ранее структуре, за исключением одного важного отличия: в данной системе отсутствуют таблицы «Комментарии» и «Уведомления».

## 1.2 Требования к проекту

В базе данных должны быть реализованы следующие требования:

- Хранение и управление пользователями – регистрация, авторизация, назначение ролей (пользователь, модератор, администратор), хранение персональных данных и контроль доступа к функционалу платформы;
- Работа с контентом – добавление, редактирование, удаление и фильтрация статей и блогов по категориям, тегам и дате публикации;
- Система оценок и подписок – возможность оценивания статей, подписка на авторов и категории, отображение персонализированной ленты;
- Комментарии и обратная связь – хранение комментариев к публикациям, их редактирование и модерация, механизм ответов и обсуждений;
- Жалобы и модерация – обработка жалоб на пользователей, статьи и комментарии, механизм блокировки нарушителей, фиксация действий модераторов;
- Уведомления и оповещения – хранение и отправка уведомлений о новых комментариях, ответах, публикациях и системных событиях;
- Аналитика и статистика – сбор и обработка данных об активности пользователей, просмотрах, рейтингах и популярных публикациях;
- Безопасность и резервное копирование – защита персональных данных пользователей, шифрование паролей, контроль доступа, регулярные резервные копии;
- Масштабируемость и производительность – обеспечение устойчивости платформы при высокой нагрузке, оптимизация структуры базы и запросов, поддержка роста аудитории.

### 1.3 Вывод по разделу

Проведенный анализ IT-платформ (Habr, Stack Overflow, TechCrunch) дал некое понимание того, как реализовать структуру базы данных и различных ее объектов. Для реализации проекта была выбрана СУБД PostgreSQL, которая идеально соответствует требованиям проекта благодаря своей надежности, производительности и богатому функционалу. PostgreSQL обеспечивает: высокую доступность данных за счет механизмов репликации, эффективную обработку сложных запросов к контенту, масштабируемость под растущую нагрузку, встроенную поддержку полнотекстового поиска и гибкую систему разграничения прав доступа. Эти характеристики делают PostgreSQL оптимальным выбором для платформы публикаций, где критически важны стабильность работы, скорость доступа к данным и возможность обработки большого объема пользовательских взаимодействий. Дополнительным преимуществом является открытая лицензия, что снижает эксплуатационные расходы при масштабировании проекта.

## **2 Проектирование и разработка базы данных**

### **2.1 Определение вариантов использования**

Наша платформа специально спроектирована таким образом, чтобы охватить самые разные сценарии взаимодействия с IT-контентом: от простого чтения и ознакомления до полного управления ресурсами. Для этого предусмотрено четыре базовых роли — гость, пользователь, модератор и администратор — каждая из которых наделена своим набором привилегий и ограничений. Гость может беспрепятственно просматривать доступные разделы, но не вмешиваться в жизнь сообщества; зарегистрированный пользователь получает возможность активно участвовать в обсуждениях, оценивать и сохранять материалы; модератор служит гарантом порядка и качества контента, оперативно реагируя на нарушения; а администратор, обладая наивысшим уровнем доступа, отвечает за стратегическое развитие платформы и техническую стабильность. Такое разграничение ролей позволяет обеспечить гибкое управление платформой, сохранять баланс между свободой действий и безопасностью, а также поддерживать высокий уровень взаимодействия внутри сообщества.

Гость — это любой посетитель, не прошедший регистрацию или авторизацию; он может свободно просматривать публичные разделы платформы — главную страницу, список категорий, профили авторов и опубликованные статьи, но не имеет возможности оценивать контент или сохранять его для последующего просмотра.

Пользователь — это зарегистрированный участник, получивший расширенный функционал: он может создавать статьи, выставлять оценки материалам, оставлять комментарии, подписываться на любимых авторов и интересующие категории, а также сохранять привлекательные статьи в «Избранное» для быстрого доступа в будущем.

Модератор — назначается администратором и отвечает за поддержание порядка на платформе: он проверяет публикации на соответствие правилам, удаляет спам и неподобающий контент, взаимодействует с системой уведомлений о нарушениях и по необходимости передаёт администратору информацию о технических неполадках или проблемах с безопасностью.

Администратор — высшая роль, закреплённая за владельцами проекта; он осуществляет полное управление системой: назначает и отзывает права доступа, контролирует безопасность и регулярное резервное копирование данных, следит за стабильностью серверов и участвует в стратегическом развитии платформы, внедряя новые функции и улучшения работы и взаимодействие с системой резервного копирования.

Ниже представлены диаграммы вариантов использования для всех ролей базы данных проекта.

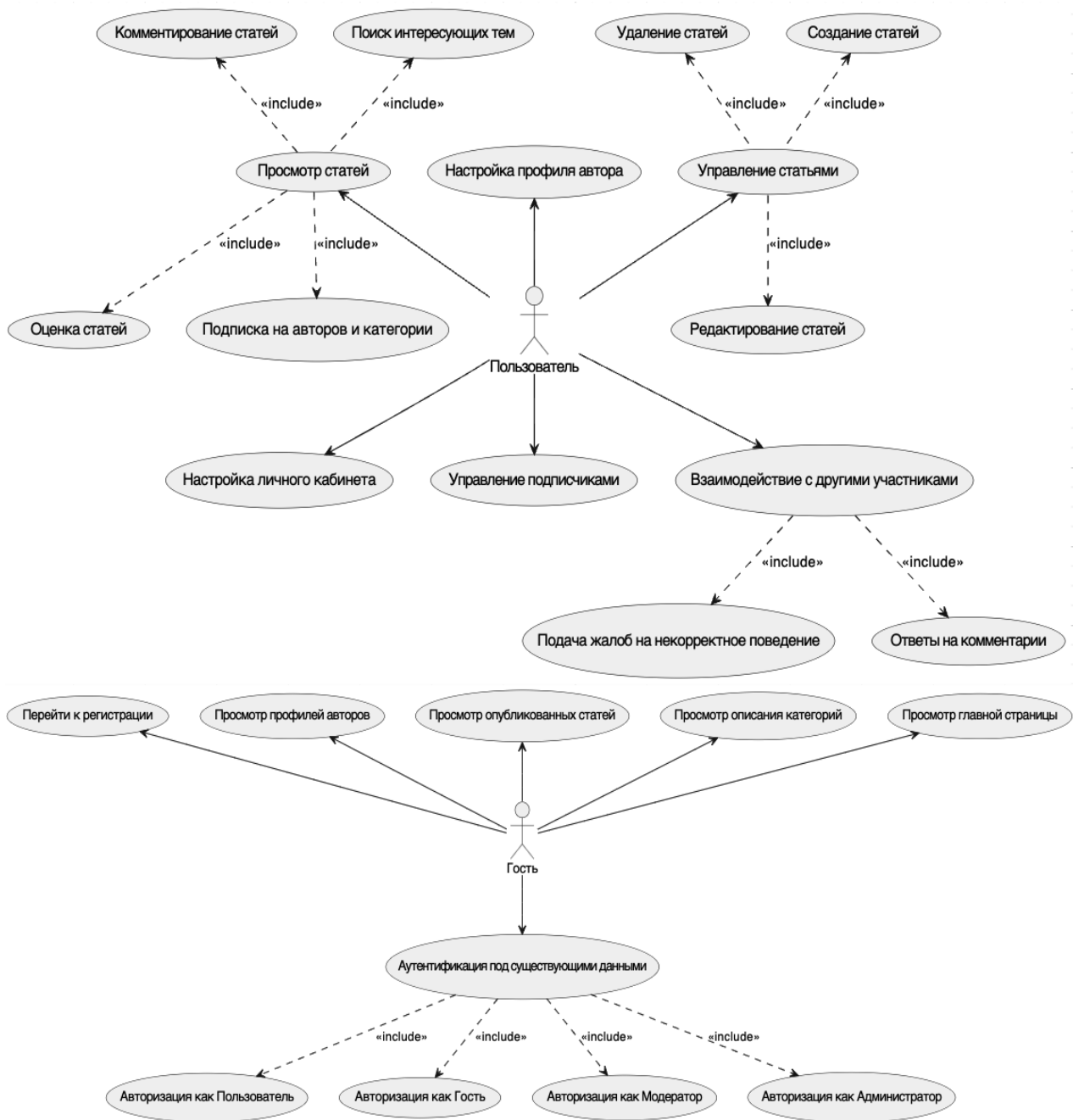


Рисунок 2.1 – Диаграмма вариантов для ролей Гость и Пользователь

Диаграммы вариантов использования наглядно показывают, как роли «Гость» и «Пользователь» взаимодействуют с системой, демонстрируя, какие действия и функциональные прецеденты доступны каждому участнику. Они позволяют чётко разграничить ответственность, визуализировать основные возможности платформы и выделить ключевые сценарии работы: от просмотра, поиска и оценки статей до комментирования, подписки на авторов и категории, управления профилем и подачи жалоб. Благодаря таким схемам легче увидеть полный набор процессов публикации, взаимодействия пользователей и обеспечения безопасности контента.

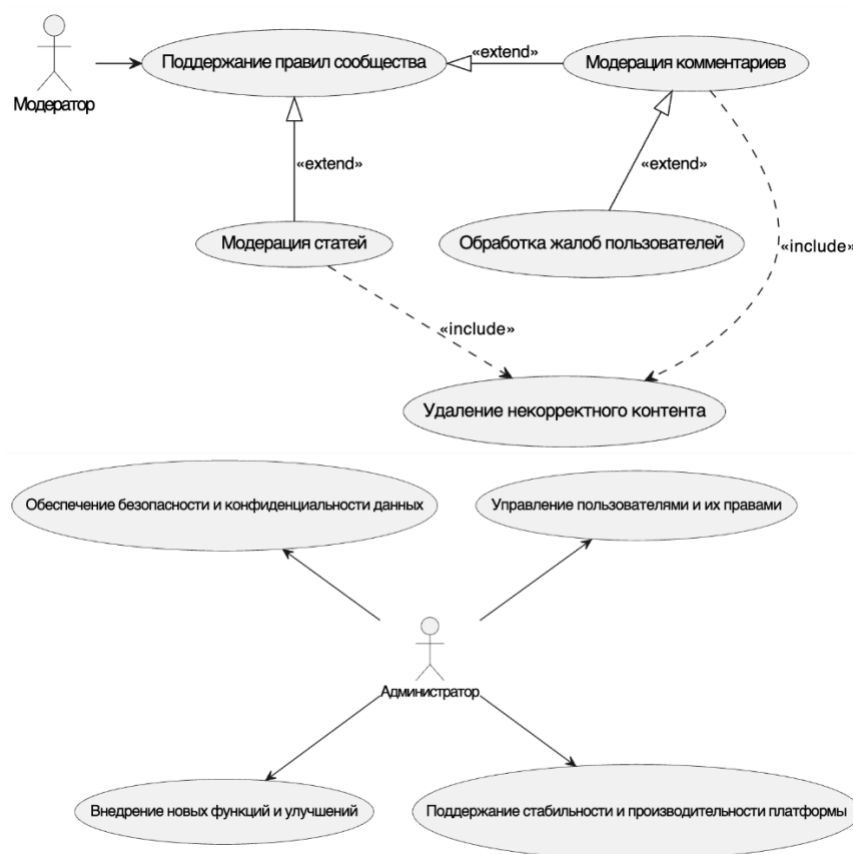


Рисунок 2.2 – Диаграмма вариантов для ролей Модератор и Администратор

Диаграммы вариантов использования наглядно демонстрируют, как роли «Модератор» и «Администратор» поддерживают жизнеспособность платформы: модератор отвечает за оперативное соблюдение правил сообщества, обрабатывает жалобы, модерирует статьи и комментарии и при необходимости удаляет некорректный контент, в то время как администратор управляет пользователями и их правами, следит за безопасностью и конфиденциальностью данных, обеспечивает стабильность и производительность системы и инициирует внедрение новых функций и улучшений; в совокупности схема чётко визуализирует разграничение ответственности между оперативным контролем качества контента и стратегическим управлением платформой.

Далее будут подробно описаны все прецеденты для каждой роли базы данных.

Общие для всех ролей:

- Просмотр публичного контента. Независимо от роли доступен просмотр основных разделов: главной страницы, описания категорий, списка опубликованных статей и профилей авторов.

Описание прецедентов для роли «Гость»:

- Просмотр главной страницы. Открытие и навигация по стартовому экрану платформы.

- Просмотр описания категорий. Изучение тематики и кратких описаний разделов.

- Просмотр опубликованных статей. Чтение доступных материалов.
- Просмотр профилей авторов. Знакомство с информацией об авторах.
- Перейти к регистрации. Переход на форму создания учётной записи.
- Аутентификация под существующими данными. Включает в себя варианты входа как Пользователь, Модератор или Администратор.
- Описание прецедентов для роли «Пользователь»:
- Просмотр статей (включает «Комментирование статей», «Оценка статей», «Подписка на авторов и категории» и «Поиск интересующих тем»).
- Настройка личного кабинета. Управление персональными данными и смена пароля.
- Управление подписчиками. Просмотр и администрирование своей аудитории.
- Настройка профиля автора. Редактирование информации о себе в качестве контент-креатора.
- Управление статьями (включает «Создание статей», «Редактирование статей» и «Удаление статей»).
- Взаимодействие с другими участниками (включает «Подачу жалоб на некорректное поведение» и «Ответы на комментарии»).
- Описание прецедентов для роли «Модератор»:
- Поддержание правил сообщества. Базовый прецедент контроля порядка.
- Модерация комментариев (расширяет «Поддержание правил сообщества»; включает «Обработку жалоб пользователей»).
- Модерация статей (расширяет «Поддержание правил сообщества»).
- Удаление некорректного контента. Включается и при модерации комментариев, и при модерации статей.
- Описание прецедентов для роли «Администратор»:
- Управление пользователями и их правами. Создание, изменение и отзыв любых учётных записей и ролей.
- Обеспечение безопасности и конфиденциальности данных. Настройка политик доступа, резервное копирование и восстановление.
- Внедрение новых функций и улучшений. Координация со службой разработки и стратегическое планирование.
- Поддержание стабильности и производительности платформы. Мониторинг серверов, балансировка нагрузки и устранение сбоев.

## 2.2 Диаграммы UML, взаимодействие всех компонентов



Рисунок 2.3 – Схема базы данных

Были разработаны следующие таблицы: Roles для хранения ролей пользователей, Users для хранения данных о пользователях, Categories для хранения категорий статей, Articles для хранения самих статей, Tags для хранения тегов, Article\_tags для связи «многие-ко-многим» между статьями и тегами, Comments для комментариев к статьям, Favourites для избранного пользователей, Ratings для оценок статей, Notifications для уведомлений пользователей, Subscriptions для управления подписками между пользователями и Reports для регистрации жалоб на пользователей.

## 2.3 Вывод по разделу

В ходе анализа были чётко выделены ключевые роли системы и их зоны ответственности, подробно описаны сценарии взаимодействия с платформой — от просмотра и создания контента до модерации и администрирования — и визуализированы в UML-диаграммах. На основе полученных прецедентов был спроектирован логический каркас базы данных, обеспечивающий хранение пользователей, ролей, контента, взаимосвязей и механизмов уведомлений и жалоб. Такое сочетание бизнес-процессов и продуманной структуры данных гарантирует прозрачное разграничение прав, надёжность операций и готовность платформы к дальнейшему развитию и масштабированию.

## 3 Разработка объектов базы данных

### 3.1 Разработка таблиц базы данных

Проектирование структуры базы данных является ключевым этапом разработки, обеспечивающим надёжное хранение, целостность и доступность данных. В данном разделе представлена разработка таблиц, отражающих основные сущности платформы публикации и взаимодействия с контентом, а также их взаимосвязи.

Все таблицы базы данных:

- Roles — содержит информацию о ролях пользователей.
- Users — хранит данные о пользователях (аватар, логин, email, статус и т. д.).

- Categories — категории статей.
- Articles — статьи, публикуемые пользователями.
- Comments — комментарии к статьям.
- Tags — теги, используемые для классификации статей.
- Article\_tags — связь «многие ко многим» между статьями и тегами.
- Favourites — избранные статьи пользователей.
- Ratings — оценки статей пользователями.
- Notifications — уведомления для пользователей.
- Subscriptions — подписки пользователей друг на друга.
- Reports — жалобы пользователей на других пользователей или контент.

Далее идет подробное описание каждой таблицы.

– Таблица Roles хранит информацию о ролях пользователей и включает следующие поля: уникальный идентификатор роли (id) и название роли (name).

– Таблица Users предназначена для хранения данных о пользователях. Она содержит следующие поля: уникальный идентификатор пользователя (id), идентификатор роли (role\_id), имя пользователя (username), адрес электронной почты (email), хеш пароля (password\_hash), ссылка на аватар (avatar\_url, по умолчанию — noimage.png), статус пользователя (status), дата и время последнего входа (last\_login) и дата регистрации (created\_at).

– Таблица Categories описывает категории статей и состоит из полей: уникальный идентификатор категории (id) и уникальное название категории (name).

– Таблица Articles предназначена для хранения публикаций пользователей. В неё входят следующие поля: уникальный идентификатор статьи (id), уникальный текстовый идентификатор (slug), идентификатор пользователя-автора (user\_id), заголовок статьи (title), краткое описание (short\_description), содержимое статьи (content), путь к изображению (image, по умолчанию — noimage.png), идентификатор категории (category\_id), статус статьи (status), дата создания (created\_at) и дата последнего обновления (updated\_at).

– Таблица Comments хранит комментарии пользователей к статьям. В таблицу входят: уникальный идентификатор комментария (id), идентификатор



статьи (`article_id`), идентификатор пользователя (`user_id`), текст комментария (`content`) и дата создания комментария (`created_at`).

- Таблица `Tags` описывает теги, которые используются для классификации статей. Она содержит поля: уникальный идентификатор тега (`id`) и название тега (`name`).

- Таблица `Article_tags` реализует связь «многие ко многим» между статьями и тегами и состоит из следующих полей: уникальный идентификатор связи (`id`), идентификатор тега (`tag_id`) и идентификатор статьи (`article_id`).

- Таблица `Favourites` содержит сведения об избранных статьях пользователей. Она включает: уникальный идентификатор записи (`id`), идентификатор пользователя (`user_id`) и идентификатор статьи (`article_id`).

- Таблица `Ratings` предназначена для хранения оценок, выставленных пользователями статьям. Она содержит поля: уникальный идентификатор оценки (`id`), идентификатор пользователя (`user_id`), идентификатор статьи (`article_id`) и значение оценки от 1 до 5 (`value`).

- Таблица `Notifications` служит для хранения пользовательских уведомлений. В таблице представлены: уникальный идентификатор уведомления (`id`), идентификатор пользователя (`user_id`), текст уведомления (`text`), статус прочтения (`is_read`) и дата создания (`created_at`).

- Таблица `Subscriptions` содержит данные о подписках пользователей друг на друга и включает: уникальный идентификатор подписки (`id`), идентификатор подписчика (`follower_id`), идентификатор пользователя, на которого оформлена подписка (`followed_id`), и флаг уведомлений (`notices`).

- Таблица `Reports` предназначена для хранения жалоб, отправленных пользователями. В неё входят поля: уникальный идентификатор жалобы (`id`), идентификатор пользователя, отправившего жалобу (`reporter_id`), идентификатор пользователя, на которого она подана (`target_id`), текст жалобы (`content`), статус жалобы (`status`) и дата создания жалобы (`date`).

## 3.2 Разработка схем базы данных

Была разработана 1 схема базы данных:

- `wonks_ru` – в этой схеме хранятся все процедуры, для работы с базой данных.

## 3.3 Разработка процедур и функций базы данных

Для данного проекта было разработано 14 процедур, основные из которых:

- `register_user` – регистрация нового пользователя.
- `authenticate_user` – аутентификация пользователя.
- `create_article` – создание новой статьи.
- `export_schema_to_json_file` – экспорт данных в json.
- `import_schema_to_json_file` – импорт данных из json.
- `subscribe_to_user` – подписка одного пользователя на другого с возможностью включения уведомлений.

### 3.4 Разработка функций базы данных

Для данного проекта было разработано 53 функций, основные из которых:

- get\_article\_details – получение всех данных статьи.
- get\_tranding\_articles – получение популярных статей.
- get\_user\_followers – получение списка подписчиков.
- get\_articles\_paginated\_filtered – получение списка статей с пагинацией и множеством фильтров.
- get\_user\_profile – получения данных профиля пользователя.
- get\_user\_favourite\_articles\_paginated\_filtered – получение списка понравившихся статей с пагинацией и множеством фильтров.

### 3.5 Разработка представлений базы данных

Для данного проекта было разработано 6 представлений:

- view\_article\_comments – представление комментариев к статьям.
- view\_user\_favourite\_articles – представление избранных статей пользователей.
- view\_user\_notification – представление уведомлений пользователей.
- view\_user\_subscription – представление подписок пользователей.
- view\_reports – представление всех жалоб.
- view\_articles – агрегированное представление статей.

### 3.6 Разработка триггеров базы данных

Для данного проекта было разработано 4 триггеров, основные из которых:

- trg\_article\_updated\_at – триггер, обновляющий поле updated\_at у статьи при каждом её изменении.
- trg\_prevent\_self\_subscription – триггер, предотвращающий возможность подписки пользователя на самого себя.
- trg\_notify\_comment – триггер, уведомляющий автора статьи о новом комментарии.
- trg\_notify\_followers\_on\_publish – триггер, рассылающий уведомления подписчикам пользователя при публикации новой статьи.

### 3.7 Роли и пользователи

В разделе 2.1 подробно описаны все роли, предусмотренные в данном проекте. При регистрации пользователю присваивается роль «Пользователь», изменить ее может только администратор

### 3.8 Вывод по разделу

В ходе разработки объектов базы данных для платформы публикаций и взаимодействия с контентом была спроектирована структура, включающая ключевые таблицы: пользователи, статьи, комментарии, оценки, подписки, уведомления, жалобы и другие. Особое внимание уделено связям между таблицами, обеспечивающим целостность данных через внешние ключи и

ограничения. Реализованы роли с разграничением прав (гость, пользователь, модератор, администратор), предусмотрен профиль пользователя с отображением личной информации и активности.

Кроме таблиц, в систему включены процедуры, функции, представления и триггеры, автоматизирующие логику работы и повышающие удобство доступа к данным. Созданы механизмы регистрации, аутентификации, фильтрации контента, подписок и уведомлений. Представления упрощают работу с агрегированными данными, а триггеры обеспечивают реакцию системы на ключевые события. Все эти решения сформировали надёжную и масштабируемую архитектуру базы данных, готовую к дальнейшему развитию платформы.

## 4 Описание процедур экспорта и импорта данных

### 4.1 Процедура импорта данных из JSON-файла

Ниже в листинге 4.1 представлена функция для импорта данных из JSON-файла.

```
CREATE OR REPLACE FUNCTION import_schema_from_jsonb(p_file TEXT,p_mode
TEXT DEFAULT 'TRUNCATE',p_schema TEXT DEFAULT 'wonks_ru')
RETURNS TABLE(status TEXT,message TEXT)
LANGUAGE plpgsql SECURITY DEFINER AS $$
DECLARE
_data JSONB;_tbl TEXT;_arr JSONB;_cols TEXT;_sql TEXT;_first BOOLEAN:=TRUE;
BEGIN
IF p_file IS NULL OR p_file="" OR p_file~'[/\\]' THEN RETURN QUERY SELECT
'ERROR','Invalid filename'; END IF;
CREATE TEMP TABLE _tmp(data JSONB) ON COMMIT DROP;
EXECUTE format('COPY _tmp FROM PROGRAM %L',format('sh -c %L','cat
/var/lib/postgresql/io/"||p_file));
SELECT data INTO _data FROM _tmp LIMIT 1;
IF _data IS NULL THEN RETURN QUERY SELECT 'ERROR','No JSON data'; END IF;
FOR _tbl,_arr IN SELECT key,value FROM jsonb_each(_data) LOOP
IF _first AND upper(p_mode)='TRUNCATE' THEN
EXECUTE format('TRUNCATE TABLE %I.%I RESTART IDENTITY
CASCADE',p_schema,_tbl);
END IF;
_first:=FALSE;
IF jsonb_typeof(_arr)='array' AND jsonb_array_length(_arr)>0 AND jsonb_typeof(_arr-
>0)='object' THEN
SELECT string_agg(quote_ident(k),',') INTO _cols FROM jsonb_object_keys(_arr->0)
AS keys(k);
_sql:=format(
'INSERT INTO %I.%I(%s)SELECT %s FROM
jsonb_populate_recordset(NULL::%I.%I,%L)ON CONFLICT(id)DO %s',
p_schema,_tbl,_cols,_cols,p_schema,_tbl,_arr,
CASE WHEN upper(p_mode)='UPSERT' THEN
'UPDATE SET '||(SELECT string_agg(format('%I=EXCLUDED.%I',k,k),',') FROM
jsonb_object_keys(_arr->0) AS keys(k) WHERE k<>'id')
ELSE 'NOTHING' END
);
EXECUTE _sql;
END IF;
END LOOP;
RETURN QUERY SELECT 'OK','Import complete';
EXCEPTION WHEN OTHERS THEN RETURN QUERY SELECT
'ERROR',SQLERRM;
END;$$;
```

Листинг 4.1 – Функция импорта из JSON-файла

Функция `import_schema` позволяет гибко восстановить данные из JSON-файла на сервере: она принимает путь к файлу, режим (TRUNCATE или

UPSERT) и схему (по умолчанию `wonks_ru`), загружает весь JSON в временную таблицу, после чего для каждой таблицы из корня JSON автоматически очищает её (при первом проходе и режиме TRUNCATE) и вставляет записи через `jsonb_populate_recordset`, динамически формируя список колонок. При конфликте по ключу `id` она либо игнорирует дубли (DO NOTHING), либо обновляет существующие строки (DO UPDATE SET ...), обеспечивая корректную и бездублирующую импортировку данных в базу.

## 4.2 Процедура экспорта данных из JSON-файла

Ниже в листинге 4.2 представлены 2 процедуры для экспорта данных в JSON-файл.

```
CREATE OR REPLACE FUNCTION export_schema_to_file(
    _file TEXT,
    _schema TEXT DEFAULT 'wonks_ru'
)
RETURNS TABLE(status TEXT, message TEXT) LANGUAGE plpgsql
SECURITY DEFINER AS $$ DECLARE
    _base_dir TEXT := '/var/lib/postgresql/io/';
    _path TEXT;
    _cmd TEXT;
BEGIN
    IF _file IS NULL OR _file = '' OR _file ~ '[/\]' THEN
        RETURN QUERY SELECT 'ERROR', 'Неверное имя файла';
        RETURN;
    END IF;
    _path := _base_dir || _file; _cmd := format('sh -c %L', 'cat > ' || _path);
    EXECUTE format(
        'COPY (SELECT jsonb_object_agg(table_name, data) FROM (SELECT
            table_name,
            COALESCE(
                (SELECT jsonb_agg(row_to_json(t)) FROM %I.%I t),
                ''[]'::jsonb
            ) AS data
        FROM information_schema.tables
        WHERE table_schema = %L
        ) sub
        ) TO PROGRAM %L',
        _schema, table_name := ",
        _schema,
        _cmd
    );

    RETURN QUERY SELECT 'OK', 'Экспорт в ' || _file || ' завершён';
EXCEPTION
    WHEN OTHERS THEN
        RETURN QUERY SELECT 'ERROR', SQLERRM;
END;
$$;
```

Листинг 4.2 – Процедура экспорта данных из JSON-файла

Функция `export_schema_to_file` принимает имя файла и опционально схему (по умолчанию `wonks_ru`), проверяет корректность имени (не пустое и без символов `"\"`), формирует полный путь поддиректории `/var/lib/postgresql/io/`, затем в одной команде `COPY ... TO PROGRAM 'sh -c "cat > /полный/путь"'` собирает данные всех таблиц указанной схемы в единый JSON-объект (через `jsonb_object_agg` и `jsonb_agg(row_to_json)`) и записывает его непосредственно в файл на сервере, после чего возвращает статус и сообщение об успешном завершении или описание ошибки.

### 4.3 Вывод по разделу

В этом разделе описаны две ключевые функции для резервного копирования и восстановления данных в схеме `wonks_ru`. Функция `export_schema_to_file(_file, _schema)` принимает имя файла и опционально имя схемы, проверяет корректность пути, собирает данные всех таблиц указанной схемы в единый JSON-объект (через `jsonb_object_agg` и `jsonb_agg(row_to_json)`), а затем в одной команде `COPY ... TO PROGRAM 'sh -c "cat > /path/..."` сохраняет получившийся JSON прямо в файл на сервере. Функция `import_schema(p_file, p_mode, p_schema)` наоборот загружает этот файл в временную таблицу, извлекает из него JSON, и для каждой таблицы из корня JSON сначала при режиме `TRUNCATE` очищает её и сбрасывает идентичности, затем вставляет записи через `jsonb_populate_recordset`, динамически формируя список колонок, и при конфликтах по ключу `id` либо игнорирует дубли (`DO NOTHING`), либо обновляет существующие строки (`DO UPDATE SET ...`). Вместе они обеспечивают надёжное экспортирование данных в JSON-файл и восстановление полного состояния базы без дублирования.

## 5 Тестирование производительности

### 5.1 Заполнение таблицы

Для оценки производительности базы данных и проверки устойчивости системы при обработке больших объёмов данных была разработана функция `insert_articles`, предназначенная для массового заполнения таблицы `Articles` тестовыми записями. Она позволяет быстро сгенерировать значительное количество статей с различными статусами, имитируя поведение реального новостного портала.

```
CREATE OR REPLACE FUNCTION insert_articles() RETURNS VOID
AS $$
    DECLARE
        i INTEGER;
        s ARTICLE_STATUS;
    BEGIN
        FOR i IN 30..100000 LOOP
            s:=(ARRAY['moderated'::ARTICLE_STATUS,
'published'::ARTICLE_STATUS,
'rejected'::ARTICLE_STATUS])[floor(random() * 3 + 1)];
            INSERT INTO articles (slug, user_id, content,
short_description, image, category_id, status, title)
            VALUES ('slug_' || i, 1, 'content_' || i,
'short_description_' || i, DEFAULT, 1, s, 'title_' || i);
        END LOOP;
    END;
$$ LANGUAGE plpgsql;
```

Листинг 5.1 – Процедура для генерации строк

Данная процедура вставляет 100000 строк в таблицу `Articles`. Для генерации последовательности используется функция `generate_series`, что обеспечивает высокую производительность при массовом создании тестовых данных и позволяет гибко управлять объёмом вставляемой информации в зависимости от целей тестирования.

### 5.2 Тестирование производительности базы данных

Для оценки влияния объёма данных и индексирования на производительность были проведены тесты выполнения выборок из таблицы `Articles`.

На первом этапе было выполнено простая выборка статьи с фильтром по полю `title`:

```
SELECT title FROM Articles WHERE content =
'content_1024';
```

Листинг 5.2 – Select-запрос таблицы `Articles`

Результат выполнения запроса представлен на рисунке 5.1.

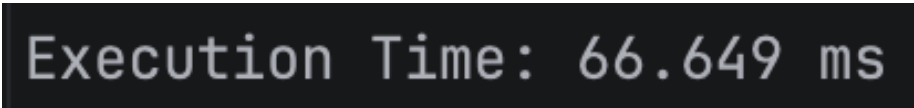
A dark rectangular box with a light gray border containing the text "Execution Time: 66.649 ms" in a light gray, monospaced font.

Рисунок 5.1 – Результат выполнения select-запроса

Анализ результатов показал, что выполнение запроса без индекса требует полного перебора всех строк таблицы, что приводит к увеличению времени выполнения при большом количестве данных.

Для оптимизации выборки был создан B-tree индекс:

```
CREATE INDEX idx_articles_content ON Articles (content);
```

Листинг 5.3 – Индекс к таблице Articles

Он создаёт B-tree индекс по столбцу content, который позволяет ускорить операции сравнения по точному совпадению (=, <>) или диапазонам (<, >, BETWEEN), однако неэффективен для полнотекстового поиска или шаблонов с подстановкой (LIKE '%...%'). Такой индекс уместен, если предполагается фильтрация по полному содержимому поля content, а не по вхождению подстрок.

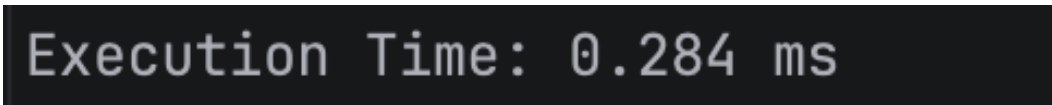
A dark rectangular box with a light gray border containing the text "Execution Time: 0.284 ms" in a light gray, monospaced font.

Рисунок 5.2 – Результат выполнения select-запроса с индексом

Результаты повторного тестирования после создания индекса представлены на рисунке 5.2. Время выполнения запроса значительно сократилось: выборка 100 000 строк заняла менее 0,2 миллисекунды. Это демонстрирует высокую эффективность применения индексирования даже к вычисляемым полям.

### 5.3 Вывод по разделу

В рамках раздела была разработана и протестирована функция `insert_articles`, позволяющая сгенерировать 100 000 строк в таблице `Articles` с уникальными значениями и случайными статусами, что позволило смоделировать работу платформы под нагрузкой. Для ускорения выборки данных по полю `content` был создан B-tree индекс `idx_articles_content`, который значительно повысил производительность: время выполнения запроса сократилось до менее 0,2 миллисекунды. Это подтвердило эффективность сочетания массовой генерации данных с индексированием при тестировании производительности базы данных.



## 6 Описание технологии и её применение в базе данных

### 6.1 Технология отказоустойчивый кластер PostgreSQL

В рамках обеспечения отказоустойчивости и высокой доступности базы данных в проекте была реализована кластерная архитектура на основе **Patroni**, **etcd** и **HAProxy**, развёрнутая в изолированной среде с использованием **Docker Compose**. Настройка состоит из трёх узлов **etcd** для обеспечения консенсуса и управления конфигурацией, трёх экземпляров **Patroni**, отвечающих за автоматическое управление кластером PostgreSQL, а также прокси-сервера **HAProxy**, выполняющего маршрутизацию клиентских подключений к активному узлу. Такой подход позволяет обеспечить автоматическое переключение на реплику при сбое основного сервера и устойчивую работу кластера без потери данных. Используемая технология демонстрирует готовность системы к эксплуатации в условиях высоких нагрузок и отказов, обеспечивая стабильную и безопасную работу с критически важной информацией.

stats																																
		Queue			Session rate			Sessions					Bytes		Denied		Errors		Warnings		Server											
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtme	Thrtle	
Frontend					2	2	-	2	2	100	5					865	29 761	0	0	2			OPEN									
Backend		0	0		0	0		0	0	10	0	0	0s	865	29 761	0	0	0	0	0	0	0	31m52s UP (0/0)			0/0	0	0			0	
patroni-postgresql-primary																																
		Queue			Session rate			Sessions					Bytes		Denied		Errors		Warnings		Server											
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtme	Thrtle	
Frontend					0	2	-	1	6	100	9					237 799	398 462	0	0	0			OPEN									
patroni1		0	0	-	0	2		1	3	100	9	9	8s			237 799	398 462		0		0	0	0	31m52s UP	L7OK/200 in 5ms	1/1	Y	-	0	0	0s	-
Backend		0	0		0	2		1	3	10	9	9	8s			237 799	398 462	0	0		0	0	0	31m52s UP (1/3)		1/1	1	0		0	0s	
patroni-postgresql-standbys																																
		Queue			Session rate			Sessions					Bytes		Denied		Errors		Warnings		Server											
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtme	Thrtle	
Frontend					0	0	-	0	0	100	0					0	0	0	0			OPEN										
patroni0		0	0	-	0	0		0	0	100	0	0	?	0	0	0	0		0	0	0	0	31m52s UP	L7OK/200 in 4ms	1/1	Y	-	0	0	0s	-	
patroni2		0	0	-	0	0		0	0	100	0	0	?	0	0	0	0		0	0	0	0	31m52s UP	L7OK/200 in 5ms	1/1	Y	-	0	0	0s	-	
Backend		0	0		0	0		0	0	10	0	0	?	0	0	0	0		0	0	0	0	31m52s UP (2/3)		2/2	2	0		0	0s		

Рисунок 6.1 – Мониторинг HAProxy

На изображении показан интерфейс HAProxy Stats, отражающий текущее состояние кластера PostgreSQL под управлением Patroni: `patroni1` работает как основной узел, а `patroni0` и `patroni2` — как резервные. Все узлы доступны, прошли проверку состояния, соединения обрабатываются без ошибок, что свидетельствует о корректной работе отказоустойчивого кластера.

### 6.2 Вывод по разделу

Применение отказоустойчивого кластера на основе Patroni и HAProxy обеспечивает надёжную работу базы данных, гарантируя доступность сервиса даже при сбое одного из узлов. Такая архитектура соответствует современным требованиям к устойчивости и масштабируемости систем, снижает риски простоев и обеспечивает непрерывность обработки пользовательских запросов.

## **Заключение**

В ходе выполнения курсового проекта была разработана полнофункциональная база данных для платформы публикации статей, реализованная с использованием СУБД PostgreSQL. Проект охватывает все этапы — от анализа аналогичных решений и постановки требований до построения логической модели, реализации процедур, функций, представлений и триггеров. Была обеспечена поддержка ключевых пользовательских ролей, реализованы механизмы регистрации, взаимодействия с контентом, системы уведомлений и жалоб. В рамках работы проведено тестирование производительности, подтверждающее эффективность применённых подходов к индексированию и генерации данных. Особое внимание было уделено вопросам надёжности и безопасности: реализовано хеширование паролей, а также развёрнута отказоустойчивая кластерная архитектура с Patroni и HAProxy. Результатом проекта стала гибкая, масштабируемая и безопасная база данных, готовая к использованию в реальной информационной системе.

### **Список используемых источников**

1. Аналог «Habr» [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/>. – Дата доступа: 12.03.2025.
2. Аналог «Stack Overflow» [Электронный ресурс]. – Режим доступа: <https://stackoverflow.com/>. – Дата доступа: 12.03.2025.
3. Аналог «TechCrunch» [Электронный ресурс]. – Режим доступа: <https://techcrunch.com/>. – Дата доступа: 12.03.2025.

## Приложение А

```
CREATE TABLE Users
(
    id SERIAL,
    avatar_url VARCHAR(255) NOT NULL DEFAULT
'noimage.png',
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    role_id INTEGER NOT NULL,
    status USER_STATUS NOT NULL DEFAULT
'activated',
    last_login TIMESTAMPTZ NOT NULL DEFAULT now(),
    created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
    PRIMARY KEY (id)
);
```

Листинг А.1 – Таблица Users

```
CREATE TABLE Categories
(
    id SERIAL,
    name VARCHAR(255) NOT NULL UNIQUE,
    PRIMARY KEY (id)
);
```

Листинг А.2 – Таблица Categories

```
CREATE TABLE Comments
(
    id SERIAL,
    article_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    content TEXT NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    PRIMARY KEY (id)
);
```

Листинг А.3 – Таблица Comments

```

CREATE TABLE Articles
(
    id          SERIAL,
    slug        TEXT          NOT NULL UNIQUE,
    user_id     INTEGER        NOT NULL,
    content     TEXT          NOT NULL,
    short_description TEXT NOT NULL,
    image       VARCHAR(255)   NOT NULL DEFAULT
'noimage.png',
    category_id INTEGER        NOT NULL,
    status      ARTICLE_STATUS NOT NULL,
    title       VARCHAR(255)   NOT NULL UNIQUE,
    created_at  TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at  TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    PRIMARY KEY (id)
);

```

Листинг А.4 – Таблица Articles

```

CREATE TABLE Tags
(
    id    SERIAL,
    name  VARCHAR(255) NOT NULL UNIQUE,
    PRIMARY KEY (id)
);

```

Листинг А.5 – Таблица Tags

```

CREATE TABLE Article_tags
(
    id          SERIAL,
    tag_id      INTEGER NOT NULL,
    article_id  INTEGER NOT NULL,
    PRIMARY KEY (id)
);

```

Листинг А.6 – Таблица Article\_tags

```
CREATE TABLE Favourites
(
    id          SERIAL,
    user_id     INTEGER NOT NULL,
    article_id  INTEGER NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг А.7 – Таблица Favourites

```
CREATE TABLE Ratings
(
    id          SERIAL,
    user_id     INTEGER NOT NULL,
    article_id  INTEGER NOT NULL,
    value       INTEGER NOT NULL CHECK (value BETWEEN 1
AND 5),
    PRIMARY KEY (id)
);
```

Листинг А.8 – Таблица Ratings

```
CREATE TABLE Notifications
(
    id          SERIAL,
    user_id     INTEGER NOT NULL,
    text        TEXT      NOT NULL,
    is_read     BOOLEAN NOT NULL DEFAULT false,
    created_at  TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    PRIMARY KEY (id)
);
```

Листинг А.9 – Таблица Notifications

```
CREATE TABLE Subscriptions
(
    id          SERIAL,
    follower_id INTEGER NOT NULL,
    followed_id INTEGER NOT NULL,
    notices     BOOLEAN NOT NULL DEFAULT false,
    PRIMARY KEY (id)
);
```

Листинг А.10 – Таблица Subscriptions

```
CREATE TABLE Reports
(
    id          SERIAL,
    reporter_id INTEGER          NOT NULL,
    target_id   INTEGER          NOT NULL,
    content     TEXT             NOT NULL,
    status      COMPLAINT_STATUS NOT NULL DEFAULT
'dispatched',
    date        TIMESTAMPTZ      NOT NULL DEFAULT
NOW(),
    PRIMARY KEY (id)
);
```

Листинг A.11 – Таблица Reports

```
CREATE TABLE Roles
(
    id          SERIAL,
    name        VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);
```

Листинг A.12 – Таблица Roles

## Приложение Б

```

CREATE OR REPLACE FUNCTION
wonks_ru.get_articles_paginated_filtered(
    _limit INTEGER DEFAULT 10,
    _offset INTEGER DEFAULT 0,
    filter_id INTEGER DEFAULT NULL,
    filter_slug TEXT DEFAULT NULL,
    filter_title TEXT DEFAULT NULL,
    filter_category_name TEXT DEFAULT NULL,
    filter_tags TEXT[] DEFAULT NULL,
    filter_created_at_start TIMESTAMPTZ DEFAULT
NULL,
    filter_created_at_end TIMESTAMPTZ DEFAULT NULL
)
    RETURNS TABLE
    (
        id                INTEGER,
        slug              TEXT,
        title             TEXT,
        content           TEXT,
        short_description TEXT,
        created_at        TIMESTAMPTZ,
        updated_at        TIMESTAMPTZ,
        image            TEXT,
        category_name     TEXT,
        tags              TEXT[],
        rating            NUMERIC
    )
    LANGUAGE plpgsql SECURITY DEFINER
AS
$$
DECLARE
    query          TEXT;
    where_clauses TEXT[] := '{}';
BEGIN
    query := 'SELECT id, slug, title, content,
short_description, created_at, updated_at, image,
category_name, tags, rating
                FROM wonks_ru.view_articles';

    IF filter_id IS NOT NULL THEN
        where_clauses :=
array_append(where_clauses, format('id = %L',
filter_id));
    END IF;

```



```

        IF filter_slug IS NOT NULL THEN
            where_clauses := array_append(where_clauses,
format('slug = %L', filter_slug));
        END IF;
        IF filter_title IS NOT NULL THEN
            where_clauses := array_append(where_clauses,
format('title ILIKE %L', '%' || filter_title || '%'));
        END IF;
        IF filter_category_name IS NOT NULL THEN
            where_clauses := array_append(where_clauses,
format('category_name = %L', filter_category_name));
        END IF;
        IF filter_tags IS NOT NULL AND
array_length(filter_tags, 1) > 0 THEN
            where_clauses := array_append(where_clauses,
format('tags @> %L', filter_tags));
        END IF;
        IF filter_created_at_start IS NOT NULL THEN
            where_clauses := array_append(where_clauses,
format('created_at >= %L', filter_created_at_start));
        END IF;
        IF filter_created_at_end IS NOT NULL THEN
            where_clauses := array_append(where_clauses,
format('created_at <= %L', filter_created_at_end));
        END IF;

        IF array_length(where_clauses, 1) > 0 THEN
            query := query || ' WHERE ' ||
array_to_string(where_clauses, ' AND ');
        END IF;

        query := query || format(' ORDER BY id DESC LIMIT
%L OFFSET %L', _limit, _offset);
        RAISE NOTICE 'Executing query: %', query;
        RETURN QUERY EXECUTE query;
    END;
    $$;

```

Листинг Б.1 – Функция получения списка статей с пагинацией и по фильтрам

```

CREATE OR REPLACE FUNCTION create_article(
    _user_id INTEGER,
    _title VARCHAR(255),
    _slug TEXT,
    _content TEXT,
    _short_description TEXT,
    _category_id INTEGER,
    _status ARTICLE_STATUS,
    _image VARCHAR(255) DEFAULT NULL,
    _tags TEXT[] DEFAULT NULL
)
    RETURNS TABLE (
        new_article_id INTEGER,
        status_code TEXT,
        message TEXT
    )
    LANGUAGE plpgsql SECURITY DEFINER
AS $$
DECLARE
    _inserted_article_id INTEGER := NULL;
    _tag_id INTEGER;
    _tag_name TEXT;
    _final_image VARCHAR(255);
BEGIN
    IF _user_id IS NULL OR _title IS NULL OR
TRIM(_title) = '' OR _slug IS NULL OR TRIM(_slug) = '' OR
        _content IS NULL OR TRIM(_content) = '' OR
    _short_description IS NULL OR TRIM(_short_description) =
'' OR
        _category_id IS NULL OR _status IS NULL
    THEN
        RETURN QUERY SELECT NULL::INTEGER,
'INVALID_INPUT'::TEXT, 'Обязательные поля (пользователь,
заголовок, slug, содержимое, краткое описание, категория,
статус) не могут быть пустыми.'::TEXT;
        RETURN;
    END IF;

    IF NOT EXISTS (SELECT 1 FROM wonks_ru.Users WHERE
id = _user_id) THEN
        RETURN QUERY SELECT NULL::INTEGER,
'USER_NOT_FOUND'::TEXT, 'Пользователь-автор не
найден.'::TEXT;
        RETURN;
    END IF;

```

```

        IF NOT EXISTS (SELECT 1 FROM wonks_ru.Categories
WHERE id = _category_id) THEN
            RETURN QUERY SELECT NULL::INTEGER,
'CATEGORY_NOT_FOUND'::TEXT, 'Категория не
найдена.'::TEXT;
            RETURN;
        END IF;

        IF EXISTS (SELECT 1 FROM wonks_ru.Articles WHERE
lower(title) = lower(_title)) THEN
            RETURN QUERY SELECT NULL::INTEGER,
'TITLE_EXISTS'::TEXT, 'Статья с таким заголовком уже
существует.'::TEXT;
            RETURN;
        END IF;

        IF EXISTS (SELECT 1 FROM wonks_ru.Articles WHERE
slug = _slug) THEN
            RETURN QUERY SELECT NULL::INTEGER,
'SLUG_EXISTS'::TEXT, 'Статья с таким slug уже
существует.'::TEXT;
            RETURN;
        END IF;

        _final_image := COALESCE(_image, (SELECT
column_default FROM information_schema.columns
WHERE
table_schema = 'wonks_ru' AND table_name = 'articles' AND
column_name = 'image'
LIMIT
1)::VARCHAR);
        _final_image := COALESCE(_final_image,
'noimage.png');

        INSERT INTO wonks_ru.Articles
(user_id, title, slug, content,
short_description, category_id, status, image,
created_at, updated_at)
VALUES
(_user_id, _title, _slug, _content,
_short_description, _category_id, _status, _final_image,
NOW(), NOW())
RETURNING id INTO _inserted_article_id;

        IF _inserted_article_id IS NULL THEN

```

```

        RETURN QUERY SELECT NULL::INTEGER,
'ERROR'::TEXT, 'Не удалось создать запись статьи.'::TEXT;
        RETURN;
    END IF;

    IF _tags IS NOT NULL AND array_length(_tags, 1)
> 0 THEN
        FOREACH _tag_name IN ARRAY _tags LOOP
            _tag_name := TRIM(_tag_name);
            IF _tag_name <> '' THEN
                SELECT id INTO _tag_id FROM
wonks_ru.Tags WHERE lower(name) = lower(_tag_name);
                IF NOT FOUND THEN
                    INSERT INTO wonks_ru.Tags
(name) VALUES (_tag_name)
ON CONFLICT (name) DO
NOTHING
RETURNING id INTO _tag_id;
                    IF _tag_id IS NULL THEN
                        SELECT id INTO _tag_id
FROM wonks_ru.Tags WHERE lower(name) = lower(_tag_name);
                    END IF;
                END IF;

                IF _tag_id IS NOT NULL THEN
                    INSERT INTO
wonks_ru.Article_tags (article_id, tag_id)
VALUES
(_inserted_article_id, _tag_id)
ON CONFLICT (article_id,
tag_id) DO NOTHING;
                ELSE
                    RAISE WARNING 'Не удалось
найти или создать тег: %', _tag_name;
                END IF;
            END IF;
        END LOOP;
    END IF;

    RETURN QUERY SELECT _inserted_article_id,
'OK'::TEXT, 'Статья успешно создана.'::TEXT;

EXCEPTION
    WHEN unique_violation THEN
        RAISE WARNING 'Ошибка уникальности при
создании статьи: %', SQLERRM;

```

```

        RETURN QUERY SELECT NULL::INTEGER,
        'ERROR'::TEXT, 'Не удалось создать статью из-за нарушения
        уникальности (заголовков или slug).';
        WHEN foreign_key_violation THEN
            RAISE WARNING 'Ошибка внешнего ключа при
        создании статьи: %', SQLERRM;
        RETURN QUERY SELECT NULL::INTEGER,
        'ERROR'::TEXT, 'Не удалось создать статью: неверный
        user_id или category_id.';
        WHEN invalid_text_representation THEN
            RAISE WARNING 'Неверное значение enum при
        создании статьи: %', SQLERRM;
        RETURN QUERY SELECT NULL::INTEGER,
        'ERROR'::TEXT, 'Указано недопустимое значение статуса.';
        WHEN OTHERS THEN
            RAISE WARNING 'Ошибка при создании статьи:
        %', SQLERRM;
        RETURN QUERY SELECT NULL::INTEGER,
        'ERROR'::TEXT, 'Произошла непредвиденная ошибка при
        создании статьи: ' || SQLERRM::TEXT;
    END;
    $$;

```

Листинг Б.2 – Функция для создания статьи

```

CREATE OR REPLACE FUNCTION
update_article_timestamp()
RETURNS TRIGGER AS
$$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

```

Листинг Б.3 – Триггерная функция для обновления поля updated\_at при обновлении в статье