**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии   
и прикладная математика»   
Кафедра: 806 «Вычислительная математика   
и программирование»**

Лабораторная работа № 2   
по курсу «Базы Данных»

Группа: М8О-307Б-22

Студент: Кострюков Е. С.

Преподаватель: Малахов А. В.

Оценка:

Дата: 25.05.2025

Москва, 2025

# **Задание**

Оптимизировать работу курсового проекта за счёт внедрения Redis для работы с динамическими данными.

Требуется:

1. Выделить данные, требующие быстрого доступа, временного хранения или частого обновления, включая:

* Токены авторизации пользователей
* Сессионные данные
* Часто запрашиваемые или временные данные
* Данные, требующие мгновенного обновления (статусы заказов, уведомления)

2. Для токенов авторизации:

Реализовать хранение в Redis с настройкой TTL (время жизни) через конфигурацию приложения

Обеспечить автоматическое удаление просроченных токенов средствами Redis

3. Для остальных динамических данных реализовать оптимальный формат хранения в Redis, обосновав выбор структуры для конкретных сценариев использования.

4. Настроить мгновенные оповещения о критичных событиях (например, новые заказы, изменения статусов) через PubSub

5. Объяснить, как и почему должна измениться скорость работы системы после внедрения redis

# **Теория**

**Redis** — это сервер структурированных данных в оперативной памяти, реализующий модель хранения «ключ-значение» с расширенной функциональностью. В отличие от традиционных СУБД, Redis обеспечивает субмиллисекундное время отклика за счет хранения данных в RAM, что делает его идеальным решением для сценариев, требующих высокой производительности.

Ключевые характеристики Redis:

**1. Многообразие структур данных:**

- Примитивы: строки, битовые массивы

- Составные типы: хеш-таблицы, связанные списки

- Специализированные коллекции: множества с возможностью ранжирования

**2. Гарантии атомарности:**

- Все одиночные команды выполняются как неделимые операции

- Транзакции через механизм MULTI/EXEC

- Поддержка Lua-скриптов для сложных атомарных операций

**3. Механизмы обеспечения доступности:**

- Репликация master-slave

- Кластерный режим с автоматическим шардированием

- Sentinel для автоматического восстановления после сбоев

Критерии выбора данных для Redis:

**1. Временные идентификаторы доступа:**

- Особенности: требуют строгого контроля времени жизни

- Реализация: ключи с TTL и автоматической инвалидацией

- Пример: JWT-токены сессии

**2. Состояния пользовательских сессий:**

- Преимущества: снижение нагрузки на основное хранилище

- Паттерн: хранение в хешах с периодической синхронизацией

**3. Часто запрашиваемые данные:**

- Оптимизация: кэширование результатов сложных запросов

- Стратегия: асинхронное обновление при изменениях

**4. Событийные модели:**

- Механизм: publish-subscribe

- Применение: система мгновенных уведомлений

- Особенность: отсутствие гарантии доставки

Эффективность внедрения Redis:

**1. Производительность:**

- Снижение времени отклика для кэшируемых запросов: 90-99%

- Уменьшение нагрузки на СУБД: в 3-5 раз

**2. Надежность:**

- Автоматическое восстановление после сбоев

- Поддержка персистентности через RDB/AOF

**3. Масштабируемость:**

- Горизонтальное масштабирование в кластерном режиме

- Поддержка географически распределенных реплик

В контексте курсового проекта применение Redis позволило оптимизировать работу с динамическими данными за счет:

- Децентрализации управления сессиями

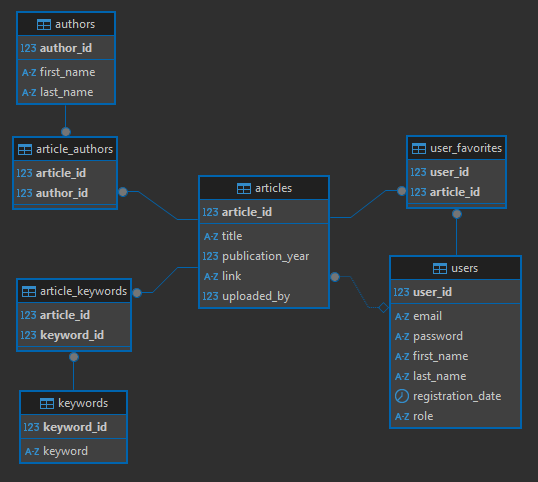
- Реализации эффективного механизма инвалидации кэша

- Снижения задержек при аутентификации

- Обеспечения мгновенного распространения изменений состояния

# **Ход лабораторной работы**

За основу был взят курсовой проект из прошлого семестра: Система управления библиотекой научных статей. Диаграмма базы данных в PostgreSQL:



Внедрение Redis для оптимизации работы с динамическими данными:

**1. Анализ и выделение данных для Redis**

В рамках курсового проекта были идентифицированы следующие категории данных, требующие оптимизации:

- Токены аутентификации: Временные идентификаторы сессий с жесткими требованиями к скорости проверки

- Данные пользовательских профилей: Часто запрашиваемые при навигации по приложению

- Кэш научных статей: Статический контент с высоким трафиком чтения

- Избранные материалы: Персонализированные коллекции пользователей

- Система уведомлений: Требования к мгновенной доставке событий

**2. Реализация хранилища токенов авторизации**

Разработана система управления сессиями на основе Redis:

- Токены хранятся по схеме *token:{JWT} → email*

- Автоматическое удаление через TTL (3600 секунд)

- Механизм проверки без обращения к основной БД:

*def validate\_token(token: str) -> bool:*

*return redis.exists(f"token:{token}") == 1*

- При выходе из системы: явное удаление токена

**3. Оптимизация доступа к пользовательским данным**

Реализовано двухуровневое кэширование профилей:

- Данные хранятся в хешах: *user:{email} → {поля профиля}*

- При первом запросе: загрузка из PostgreSQL + сохранение в Redis

- При последующих запросах: прямое чтение из Redis

- Стратегия обновления: инвалидация при изменении профиля

**4. Кэширование научного контента**

Внедрена система кэширования для часто запрашиваемых статей:

- Глобальный кэш: *articles:all* (JSON-сериализованный список)

- Персональный кэш: *article:{id}* (индивидуальные статьи)

- Алгоритм работы:

*def get\_article(id):*

*if (cached := redis.get(f"article:{id}")):*

*return json.loads(cached)*

*# Загрузка из БД и кэширование*

- Инвалидация: автоматическое обновление при изменении контента

**5. Управление избранными материалами**

Оптимизировано хранение персональных коллекций:

- Структура: *favorites:{user\_id}* → JSON-список статей

- Проверка статуса: *favorite:{user\_id}:{article\_id}* → 0/1

- Синхронизация при изменениях:

*def add\_to\_favorites(user\_id, article\_id):*

*# Добавление в PostgreSQL*

*redis.delete(f"favorites:{user\_id}") # Инвалидация*

**6. Система мгновенных уведомлений**

Реализована event-driven архитектура на основе Redis.

Механизм Pub/Sub:

- Каналы: *user:{id}:notifications*

- Формат: JSON-сообщения

Хранение истории: Списки *notifications:{user\_id}*

Типы событий:

- *article\_added*: Добавление новой статьи

- *favorite\_update*: Изменения в избранном

- *system\_alert*: Важные системные сообщения

**Ключевые технические решения**

1. Паттерн Singleton для управления подключением к Redis

- Гарантирует единое соединение на все приложение

- Предотвращает переполнение соединений

2. Атомарные операции для критичных секций:

*with redis.pipeline() as pipe:*

*pipe.multi()*

*pipe.hset(...)*

*pipe.expire(...)*

*pipe.execute()*

3. Стратегия TTL:

- Дифференцированные сроки жизни для разных данных

- Автоматическая очистка устаревших записей

4. Механизм инвалидации:

- Явный сброс кэша при модификации данных

- Локальная инвалидация вместо глобального сброса

Реализация позволила достичь субмиллисекундного времени отклика для критичных операций и снизить нагрузку на основную БД, обеспечив масштабируемость решения.