МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

КУРСОВАЯ РАБОТА по дисциплине «Базы данных»

на тему: «Разработка информационной системы управления спортивной лигой по армрестлингу»

Выполнил:
студент группы М8О-301Б-23
Хныченко Артём Викторович

_____
(подпись)

Руководитель:
_____

Оценка:_____

_____
(подпись)

Москва 2025

РЕФЕРАТ

Пояснительная записка выполнена в 1 части и содержит: страниц — 32, иллюстраций — 12, приложений — 1, в пояснительной записке использовано источников — 6.

БАЗА ДАННЫХ, POSTGRESQL, KOTLIN, SPRING BOOT, API, SQL, АРМРЕСТЛИНГ, СПОРТИВНАЯ ЛИГА, АВТОМАТИЗАЦИЯ

Объектом исследования является процесс организации и проведения спортивных соревнований по армрестлингу.

Цель работы — разработка информационной системы с реляционной базой данных для автоматизации учета участников, турниров и результатов матчей.

В процессе работы проведен анализ предметной области, спроектирована инфологическая модель базы данных, реализованы скрипты создания таблиц, триггеры и хранимые процедуры. Разработано серверное приложение на языке Kotlin, обеспечивающее взаимодействие с базой данных через REST API.

В результате исследования создана работоспособная информационная система «ArmLeague», поддерживающая ведение реестра спортсменов, формирование турнирных сеток, фиксацию результатов поединков и автоматический расчет рейтингов.

Система может быть использована для цифровизации деятельности спортивных федераций и любительских лиг армрестлинга.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Армрестлинг — динамичный вид спорта, требующий обработки специфических данных: результатов борьбы на левой и правой руках, учета фолов и сложных схем выбывания. Ручная обработка протоколов приводит к задержкам соревнований и ошибкам в рейтингах.

Актуальность темы обусловлена необходимостью создания специализированной информационной системы, позволяющей автоматизировать судейство, исключить человеческий фактор при подсчете очков и обеспечить централизованное хранение истории матчей.

Объектом исследования является процесс организации и проведения спортивных соревнований по армрестлингу.

Предметом исследования выступают методы проектирования реляционных баз данных и алгоритмы обработки спортивной статистики.

Целью работы является создание информационной системы «ArmLeague» для автоматизации регистрации участников, фиксации результатов поединков и расчета динамического рейтинга спортсменов.

Для достижения цели решены следующие **задачи**:

1) проведен анализ предметной области и выделены ключевые сущности системы;
2) спроектирована нормализованная схема базы данных;
3) реализована физическая модель данных в СУБД PostgreSQL с учетом ограничений целостности;
4) разработана серверная логика (триггеры, хранимые функции) для аудита и обработки статистики;
5) реализован программный интерфейс (API) для взаимодействия с базой данных;
6) выполнена оптимизация SQL-запросов с использованием индексов.

Практическая значимость работы заключается в создании программного продукта, готового к использованию спортивными федерациями для повышения эффективности проведения турниров.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Обзор предметной области

Предметной областью курсовой работы является деятельность спортивной лиги или федерации по армрестлингу. Армрестлинг — вид единоборств, в котором поединок проводится на специальных столах между двумя участниками.

Ключевые процессы предметной области, подлежащие автоматизации:

- Регистрация участников. Спортсмены должны предоставить персональные данные, а также пройти процедуру взвешивания для определения весовой категории.
- Формирование турнирной сетки. Участники распределяются по парам в зависимости от выбранной системы проведения (обычно система с выбыванием после двух поражений — Double Elimination).
- Проведение поединков. Судьи фиксируют результат схватки, который может быть достигнут победой, перебором фолов или по решению судей.
- Ведение рейтинга. На основании результатов турниров формируется индивидуальный рейтинг спортсмена, отражающий его уровень мастерства.

Особенностью предметной области является разделение соревнований на две независимые дисциплины: борьба на левой руке и борьба на правой руке. Один и тот же спортсмен может иметь разные показатели силы и разные рейтинги для каждой руки.

1.2 Постановка задачи

Целью проектирования является создание информационной системы, которая позволит отказаться от использования бумажных носителей и электронных таблиц (Excel) в пользу централизованной базы данных.

Разрабатываемая система должна обеспечивать выполнение следующих функций:

– хранение базы данных спортсменов и судей;
– создание турниров с настройкой весовых категорий и призового фонда;
– регистрация спортсменов в конкретные категории;
– фиксация результатов матчей в реальном времени;
– автоматический пересчет рейтинга спортсменов по итогам матчей.

К базе данных предъявляются следующие требования:

– Целостность данных. Невозможность удаления турнира, если в нем уже проведены матчи; невозможность регистрации спортсмена в несуществующую категорию.
– Безопасность. Хранение паролей пользователей в хэшированном виде.
– Аудит. Журналирование изменений ключевых таблиц (кто и когда изменил результат матча).
– Производительность. Использование индексов для ускорения поиска по турнирным сеткам и рейтингам.

Для реализации системы отлично подходит реляционная модель данных, так как предметная область характеризуется четкой структурой и наличием строгих связей между сущностями (Спортсмен — Заявка — Турнир). В качестве СУБД используется PostgreSQL, предоставляющая широкие возможности для реализации серверной логики на языке PL/pgSQL.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Архитектура информационной системы

2.1.1 Обоснование выбора стека технологий

При проектировании информационной системы «ArmLeague» был проведен анализ современных средств разработки корпоративных приложений. Выбор технологического стека обусловлен требованиями к надежности, масштабируемости и скорости разработки.

В качестве основного языка разработки выбран Kotlin (версия 1.9), исполняемый на виртуальной машине Java (JVM 17). Выбор обусловлен его лаконичностью, null-безопасностью, которая позволяет избежать ошибок NullPointerException на этапе компиляции, и полной совместимостью с экосистемой Java.

Для реализации серверной части использован фреймворк Spring Boot (версия 3.2.12). Он обеспечивает внедрение зависимостей (Dependency Injection) для слабого связывания компонентов, упрощенную конфигурацию через аннотации, встроенный веб-сервер (Tomcat/Netty) для обработки HTTP-запросов.

В проекте используются модули spring-boot-starter-web для REST API и spring-boot-starter-data-jpa для работы с базой данных.

В качестве системы управления базами данных используется объектно-реляционная СУБД PostgreSQL 15. Она поддерживает сложные SQL-запросы, пользовательские типы данных (ENUM), хранимые процедуры на языке PL/pgSQL и обеспечивает соблюдение принципов ACID (атомарность, согласованность, изолированность, долговечность транзакций).

Для обеспечения идемпотентности среды запуска используется технология контейнеризации. В файле docker-compose.yml описаны два взаимосвязанных сервиса: db (база данных) и backend (приложение), что позволяет развернуть систему одной командой на любом сервере.

2.1.2 Описание архитектурных слоев

Система спроектирована по классической трехзвенной архитектуре (Рисунок 1), где логика разделена на слои ответственности.

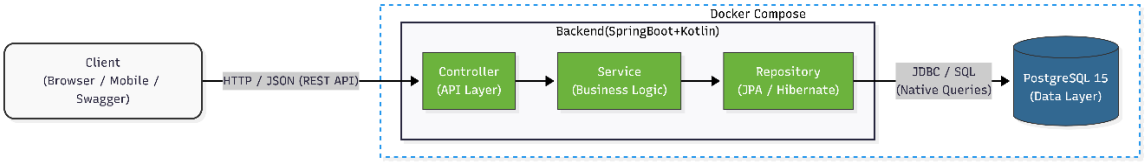

Рисунок 1 — Архитектура приложения

Реализация уровней в коде выглядит следующим образом:

1) Уровень представления (API Layer). Представлен REST-контроллерами (пакет com.armleague.backend.controller). Контроллеры отвечают за прием HTTP-запросов от клиента, валидацию входных данных (DTO) и отправку ответов в формате JSON. Пример реализации контроллера AthleteController:
   - принимает POST-запрос на регистрацию,
   - делегирует обработку сервисному слою,
   - возвращает статус 200 OK и созданный объект.

   Для документирования API использована библиотека springdoc-openapi, которая автоматически генерирует интерактивную документацию Swagger UI.
2) Уровень бизнес-логики (Service Layer). Сосредоточен в пакете com.armleague.backend.service. Сервисы (например, MatchService, RegistrationService) содержат основную логику приложения. Ключевой особенностью является использование аннотации @Transactional. Это гарантирует, что сложные операции, затрагивающие несколько таблиц (например, взвешивание атлета, которое обновляет и заявку, и профиль атлета), выполняются в рамках одной транзакции. Если на любом этапе возникнет ошибка, все изменения будут автоматически отменены (rollback).

3) Уровень доступа к данным (Data Access Layer). Реализован с помощью Spring Data JPA (пакет com.armleague.backend.repository). Репозитории представляют собой интерфейсы, наследуемые от JpaRepository. Это позволяет выполнять стандартные операции CRUD (Create, Read, Update, Delete) без написания SQL-кода. Для сложных аналитических выборок используются аннотации @Query с параметром nativeQuery = true, позволяющие выполнять оптимизированные SQL-запросы напрямую к PostgreSQL.

2.2 Проектирование логической модели базы данных

Проектирование базы данных выполнялось на основе анализа предметной области армрестлинга. Главной целью было создание нормализованной схемы, исключающей дублирование информации и обеспечивающей целостность данных на уровне СУБД.

2.2.1 Нормализация и сущности

База данных спроектирована в Третьей нормальной форме (3НФ):

– Все атрибуты атомарны (1НФ);
– Все неключевые атрибуты зависят от всего первичного ключа (2НФ);
– Отсутствуют транзитивные зависимости неключевых атрибутов (3НФ).

Выделены следующие ключевые сущности:

1) Пользователь (User). Базовая сущность для аутентификации. Содержит email и хэш пароля.
2) Роль. Реализована через механизм наследования или связей 1:1. Пользователь может быть Атлетом (Athlete) или Судьей (Referee).
3) Турнир (Tournament). Событие, имеющее дату начала и окончания.
4) Весовая категория (WeightClass). Подразделение внутри турнира (например, "Мужчины до 90 кг, левая рука").
5) Заявка (Registration). Факт участия конкретного атлета в конкретной категории.

6) Матч (Match). Поединок между двумя атлетами.
7) Протокол (MatchProtocol). Детальная информация о матче (фолы, время, тип победы).

2.2.2 Описание связей между сущностями

Для реализации логики системы установлены следующие типы связей (Рисунок 2):

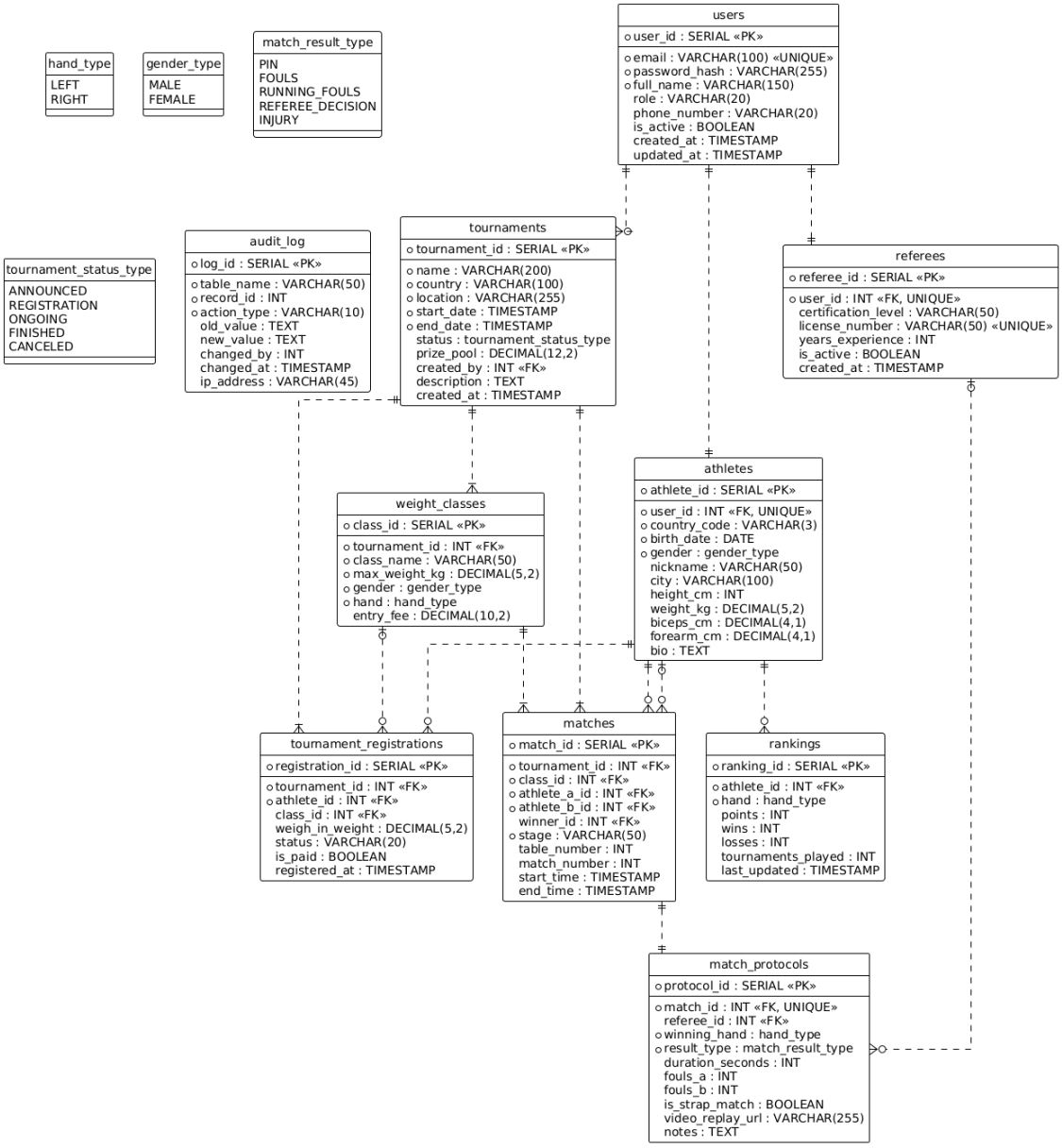

Рисунок 2 — Схема базы данных

Связи «Один-к-одному» (1:1):

– users — athletes: Каждый атлет является пользователем, но не каждый пользователь — атлет. Реализовано через внешний ключ user_id в таблице athletes с ограничением UNIQUE.
– users—referees: Аналогично для судей.
– matches — match_protocols: Протокол существует только в контексте конкретного матча. Разделение сделано для оптимизации: таблица матчей (расписание) весит меньше без текстовых примечаний и деталей фолов, которые нужны реже.

Связи «Один-ко-многим» (1:N):

– tournaments — weight_classes: Один турнир содержит множество весовых категорий. При удалении турнира все категории удаляются каскадно (ON DELETE CASCADE).
– weight_classes — matches: В одной категории проходит множество матчей.

Связи «Многие-ко-многим» (N:M):

– Атлеты и Турниры. Прямая связь N:M раскрыта через таблицу ассоциации tournament_registrations. Один атлет может участвовать в разных турнирах, и в одном турнире много атлетов. Эта таблица также хранит специфические атрибуты связи: статус оплаты взноса и вес на взвешивании.
– Атлеты и Матчи. Связь реализована через два внешних ключа в таблице matches: athlete_a_id и athlete_b_id.

2.2.3 Использование пользовательских типов данных

Для повышения надежности данных и экономии памяти вместо строковых полей использованы перечисления PostgreSQL (TYPE AS ENUM), определенные в скрипте инициализации:

– hand_type: определяет руку (LEFT, RIGHT);
– gender_type: пол участника (MALE, FEMALE);

– match_result_type: способ победы (PIN — удержание, FOULS — по фолам и др.);
– tournament_status_type: жизненный цикл турнира (от ANNOUNCED до FINISHED).

Использование ENUM на уровне БД гарантирует, что в систему не попадут некорректные значения (например, опечатки в названии статуса), даже если валидация на бэкенде будет пропущена.

2.3 Физическая структура базы данных

Физическая модель данных реализована в среде СУБД PostgreSQL 15. Схема данных спроектирована с учетом требований нормализации и включает 10 таблиц. Для каждого атрибута определен тип данных, соответствующий характеру хранимой информации, а также наложены ограничения целостности (PRIMARY KEY, FOREIGN KEY, NOT NULL, CHECK).

2.3.1 Подсистема пользователей

Таблица users (пользователи) является центральной сущностью для аутентификации. Содержит следующие атрибуты:

– user_id (SERIAL, PK): Уникальный суррогатный ключ, генерируемый автоматически.
– email (VARCHAR(100), UNIQUE, NOT NULL): Адрес электронной почты, используемый в качестве логина. Ограничение уникальности предотвращает регистрацию дублей.
– password_hash (VARCHAR(255), NOT NULL): Хэш пароля пользователя (в целях безопасности пароли не хранятся в открытом виде).
– role (VARCHAR(20)): Роль пользователя в системе. На поле наложено ограничение CHECK (role IN ('ADMIN', 'USER', 'REFEREE')).
– full_name (VARCHAR(150), NOT NULL): Полное имя пользователя.

– is_active (BOOLEAN): Флаг активности учетной записи (по умолчанию TRUE).
– created_at (TIMESTAMP): Дата и время регистрации.

Таблица athletes (спортсмены) хранит профили участников соревнований. Связана с таблицей пользователей отношением «один-к-одному». Содержит следующие атрибуты:

– athlete_id (SERIAL, PK): Идентификатор профиля.
– user_id (INT, FK, UNIQUE): Ссылка на учетную запись пользователя. При удалении пользователя профиль атлета удаляется каскадно (ON DELETE CASCADE).
– country_code (VARCHAR(3), NOT NULL): Код страны (ISO 3166-1 alpha-3).
– birth_date (DATE, NOT NULL): Дата рождения.
– gender (gender_type): Пол спортсмена (перечисление MALE или FEMALE).
– height_cm (INT): Рост в сантиметрах. Установлено ограничение на реалистичность данных: CHECK (height_cm > 100 AND height_cm < 250).
– weight_kg (DECIMAL(5,2)): Вес в килограммах. Ограничение: CHECK (weight_kg > 30 AND weight_kg < 250).
– biceps_cm, forearm_cm (DECIMAL(4,1)): Антропометрические данные (объем бицепса и предплечья).

Таблица referees (судьи) хранит информацию о судейском составе. Содержит следующие атрибуты:

– referee_id (SERIAL, PK): Идентификатор судьи.
– user_id (INT, FK, UNIQUE): Ссылка на пользователя.
– certification_level (VARCHAR(50)): Уровень квалификации (например, «National», «International»).
– license_number (VARCHAR(50), UNIQUE): Номер судейской лицензии.

– years_experience (INT): Стаж судейства (значение должно быть больше или равно 0).

2.3.2 Подсистема организации турниров

Таблица tournaments (турниры) писывает спортивные события. Содержит следующие атрибуты:

– tournament_id (SERIAL, PK): Идентификатор турнира.
– name (VARCHAR(200), NOT NULL): Официальное название.
– start_date, end_date (TIMESTAMP): Даты начала и окончания. Целостность обеспечивается ограничением CHECK (end_date >= start_date).
– status (tournament_status_type): Текущее состояние турнира (ANNOUNCED, REGISTRATION, ONGOING, FINISHED, CANCELED).
– prize_pool (DECIMAL(12,2)): Призовой фонд. Значение по умолчанию 0.00.
– location (VARCHAR(255)): Место проведения.

Таблица weight_classes (весовые категории) разделяет участников турнира на группы. Содержит следующие атрибуты:

– class_id (SERIAL, PK): Идентификатор категории.
– tournament_id (INT, FK): Ссылка на турнир.
– class_name (VARCHAR(50)): Название категории (например, «Senior 80kg»).
– max_weight_kg (DECIMAL(5,2)): Верхняя граница веса для допуска.
– hand (hand_type): Рука, на которой проводится борьба (LEFT или RIGHT).
– entry_fee (DECIMAL(10,2)): Взнос за участие.

На таблицу наложено уникальное ограничение UNIQUE(tournament_id, class_name, hand), исключающее дублирование категорий в рамках одного турнира.

Таблица tournament_registrations (заявки) реализует связь «многие-ко-многим» между атлетами и турнирами. Содержит следующие атрибуты:

- registration_id (SERIAL, PK): Идентификатор заявки.
- tournament_id (INT, FK), athlete_id (INT, FK): Ссылки на турнир и атлета.
- class_id (INT, FK): Ссылка на выбранную весовую категорию.
- weigh_in_weight (DECIMAL(5,2)): Фактический вес спортсмена на взвешивании.
- status (VARCHAR(20)): Статус заявки (PENDING, APPROVED, REJECTED, DISQUALIFIED).
- is_paid (BOOLEAN): Отметка об оплате стартового взноса.

2.3.3 Подсистема проведения поединков

Транзакционная таблица matches (поединки) хранит пары соперников. Содержит следующие атрибуты:

- match_id (SERIAL, PK): Идентификатор матча.
- tournament_id (INT, FK), class_id (INT, FK): Привязка к турниру и категории.
- athlete_a_id (INT, FK), athlete_b_id (INT, FK): Ссылки на участников поединка.
- winner_id (INT, FK): Ссылка на победителя. Поле может быть пустым (NULL), пока матч не завершен.
- stage (VARCHAR(50)): Этап турнирной сетки (например, «1/4 финала»).
- start_time, end_time (TIMESTAMP): Временные метки матча.

Ограничение CONSTRAINT check_winner_participant гарантирует, что winner_id совпадает с одним из участников (athlete_a_id или athlete_b_id).

Таблица match_protocols (протоколы) хранит детализированную статистику матча. Связана с matches отношением 1:1. Содержит следующие атрибуты:

– protocol_id (SERIAL, PK): Идентификатор протокола.
– match_id (INT, FK, UNIQUE): Ссылка на матч.
– referee_id (INT, FK): Ссылка на судью, обслуживающего поединок.
– result_type (match_result_type): Способ победы (Удержание, Фол, Травма и др.).
– fouls_a, fouls_b (INT): Количество фолов у участников.
– duration_seconds (INT): Длительность поединка в секундах.

2.3.4 Аналитика и аудит

Таблица rankings (рейтинги) содержит текущий рейтинг спортсменов (аналог ELO). Содержит следующие атрибуты:

– ranking_id (SERIAL, PK): Идентификатор записи.
– athlete_id (INT, FK): Ссылка на атлета.
– points (INT): Количество очков (по умолчанию 1000).
– wins, losses (INT): Статистика побед и поражений.
– hand (hand_type): Рука, для которой рассчитан рейтинг. Рейтинги на левую и правую руку ведутся раздельно (уникальность по паре athlete_id + hand).

Таблица audit_log (журнал аудита) используется для трекинга изменений данных. Содержит следующие атрибуты:

– log_id (SERIAL, PK): Идентификатор записи лога.
– table_name (VARCHAR(50)): Имя таблицы, в которой произошло изменение.
– record_id (INT): ID измененной записи.
– action_type (VARCHAR(10)): Тип операции (INSERT, UPDATE, DELETE).
– old_value, new_value (TEXT): Состояние записи до и после изменения.
– changed_at (TIMESTAMP): Время события.

2.4 Реализация активной логики базы данных

Для обеспечения целостности данных и реализации бизнес-правил на стороне сервера БД использован процедурный язык PL/pgSQL. Активная логика разделена на триггеры (автоматическая реакция на события), хранимые функции (вычисления) и представления (виртуальные таблицы).

2.4.1 Триггеры и автоматизация аудита

В системе реализовано два ключевых триггера.

Триггер аудита изменений (trg_matches_audit) срабатывает при обновлении (UPDATE) записи в таблице матчей. Его задача — зафиксировать факт изменения результата поединка, сохранив старое и новое значение победителя (Рисунок 3).

```
CREATE
OR REPLACE FUNCTION log_matches_changes()
RETURNS TRIGGER AS
$$ BEGIN
IF (TG_OP = 'UPDATE') THEN INSERT INTO audit_log (
  table_name, record_id, action_type,
  old_value, new_value, changed_by
)
VALUES
  (
    'matches',
    OLD.match_id,
    'UPDATE',
    'Winner: ' || COALESCE(OLD.winner_id :: TEXT, 'None'),
    'Winner: ' || COALESCE(NEW.winner_id :: TEXT, 'None'),
    NULL
  );
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_matches_audit
AFTER
UPDATE
  ON matches FOR EACH ROW EXECUTE FUNCTION log_matches_changes();
```

Рисунок 3 — Код триггера логирования изменений

Триггер пересчета рейтинга (trg_update_stats) срабатывает автоматически после завершения матча (когда поле winner_id перестает быть NULL). Триггер определяет победителя и проигравшего, а затем вызывает

вспомогательную функцию для начисления рейтинговых очков. Алгоритм работы:

1) определяется рука, на которой проходил поединок (через таблицу весовых категорий);
2) победитель получает +10 очков, количество побед увеличивается на 1;
3) проигравший теряет 10 очков, количество поражений увеличивается на 1.

2.4.2 Хранимые функции

Для инкапсуляции сложной логики разработаны скалярные и табличные функции.

Скалярная функция расчета ИМТ (calculate_bmi), принимающая вес и рост спортсмена, выполняет расчет индекса массы тела по формуле (1):

$$I = \frac{m}{\left(h/100\right)^2}, \quad (1)$$

где I – индекс массы тела;

m – масса тела спортсмена, кг;

h – рост спортсмена, см.

Функция обрабатывает граничные случаи (деление на ноль), возвращая 0, если данные роста некорректны.

Табличная функция истории матчей (get_athlete_history) возвращает полный список боев конкретного спортсмена (Рисунок 4). Она решает задачу формирования «Личного кабинета», скрывая сложность объединения (JOIN) четырех таблиц.

```sql
CREATE OR REPLACE FUNCTION get_athlete_history(search_athlete_id INT)
RETURNS TABLE (
    match_date TIMESTAMP,
    opponent_name VARCHAR,
    result VARCHAR,
    stage VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        m.end_time,
        CASE
            WHEN m.athlete_a_id = search_athlete_id THEN u2.full_name
            ELSE u1.full_name
        END as opponent_name,
        CASE
            WHEN m.winner_id = search_athlete_id THEN 'WIN'
            ELSE 'LOSS'
        END as result,
        m.stage
    FROM matches m
    JOIN athletes a1 ON m.athlete_a_id = a1.athlete_id
    JOIN users u1 ON a1.user_id = u1.user_id
    JOIN athletes a2 ON m.athlete_b_id = a2.athlete_id
    JOIN users u2 ON a2.user_id = u2.user_id
    WHERE (m.athlete_a_id = search_athlete_id OR m.athlete_b_id = search_athlete_id)
      AND m.winner_id IS NOT NULL;
END;
$$ LANGUAGE plpgsql;
```

Рисунок 4 — Код функции получения истории матчей

2.4.3 Представления (Views)

Для упрощения аналитических запросов со стороны Backend-приложения созданы три представления.

1) v_athlete_summary (сводная статистика). Собирает данные из таблиц пользователей, атлетов и рейтингов. Рассчитывает производный показатель — процент побед.
2) v_match_schedule (расписание турнира). Преобразует нормализованную таблицу matches, содержащую только ID, в человекочитаемый вид. Вместо идентификаторов участников выводит их имена, а вместо ID категории — её название.
3) v_referee_workload (нагрузка на судей). Агрегирует данные из протоколов матчей. Группирует записи по судьям и считает количество обслуженных поединков и общее количество зафиксированных фолов.

2.5 Описание API и взаимодействия с базой данных

Взаимодействие клиентской части системы с базой данных осуществляется через программный интерфейс (API), построенный по архитектурному стилю REST. Серверная часть реализована на платформе Spring Boot, что обеспечивает строгую типизацию данных и управление транзакциями.

2.5.1 Организация REST-контроллеров

Точками входа в приложение являются контроллеры (@RestController). Они принимают HTTP-запросы, выполняют валидацию входных данных (DTO) и вызывают методы бизнес-логики.

В системе реализованы следующие группы эндпоинтов:

– /api/tournaments: Управление жизненным циклом турниров (создание, запуск, завершение);
– /api/athletes: CRUD-операции над профилями спортсменов;
– /api/matches: Оперативное управление ходом поединков (назначение победителя, фиксация фолов);
– /api/analytics: Получение отчетов и сводной статистики.

Для передачи данных используются объекты передачи данных (Data Transfer Objects), что позволяет скрыть внутреннюю структуру сущностей БД от клиента. Например, при регистрации атлета используется RegisterAthleteDto, содержащий только необходимые поля (рост, вес, пол), в то время как сущность Athlete содержит также служебные поля (ID, внешние ключи).

2.5.2 Реализация массовой загрузки данных

Согласно техническому заданию, в системе реализован механизм пакетной обработки данных. Это необходимо для первичного наполнения базы данных или миграции информации из старых систем.

Логика импорта реализована в классе BatchController. Метод обрабатывает список входных объектов, выполняя попытку сохранения для каждой записи в отдельности. Ошибки при сохранении одной записи не прерывают весь процесс, а накапливаются в списке ошибок (Рисунок 5).

```kotlin
@PostMapping("/athletes")
@Operation(summary = "Массовый импорт атлетов (списком)")
fun importAthletes(@RequestBody dtos: List<RegisterAthleteDto>): Map<String, Any> {
    var successCount = 0
    val errors = mutableListOf<String>()

    dtos.forEach { dto ->
        try {
            athleteService.registerNewAthlete(dto)
            successCount++
        } catch (e: Exception) {
            errors.add("Error importing ${dto.email}: ${e.message}")
        }
    }

    return mapOf("total" to dtos.size, "imported" to successCount, "errors" to errors)
}
```

Рисунок 5 — Реализация алгоритма масосвой загрузки атлетов

Данный подход обеспечивает устойчивость системы при обработке больших массивов данных (500–1000 записей).

2.5.3 Слой доступа к данным и нативные запросы

Взаимодействие с PostgreSQL осуществляется через уровень репозиториев. Используется гибридный подход:

1) ORM (Hibernate): Для стандартных операций создания и обновления записей используются методы интерфейса JpaRepository. Это позволяет работать с данными как с объектами Kotlin.
2) Native SQL: Для построения сложных аналитических отчетов, требующих объединения множества таблиц и агрегации данных, используются прямые SQL-запросы через аннотацию @Query с параметром nativeQuery = true.

Примером гибридного подхода является метод поиска топ-10 атлетов по количеству побед в классе AthleteRepository (Рисунок 6). Запрос обращается напрямую к таблицам и использует сортировку на стороне СУБД для максимальной производительности.

```
@Query(
    value =
        """
            SELECT
                a.nickname,
                a.country_code,
                r.wins
            FROM rankings r
            JOIN athletes a ON r.athlete_id = a.athlete_id
            WHERE r.wins > 0
            ORDER BY r.wins DESC, r.points DESC
            LIMIT 10
        """,
    nativeQuery = true
)
fun findTopAthletesByWins(): List<Array<Any>>
```

Рисунок 6 — Использование нативного SQL-запроса в репозитории

2.5.4 Документирование API

Для автоматизации взаимодействия и упрощения тестирования подключена библиотека springdoc-openapi. При запуске приложения она сканирует контроллеры и аннотации @Operation, генерируя интерактивную документацию в формате Swagger UI (OpenAPI 3.0). Это позволяет тестировать запросы к базе данных (отправку JSON, получение ответов) непосредственно из браузера, без написания клиентского кода.

2.6 Оптимизация и анализ производительности

С ростом объема данных время выполнения запросов может критически увеличиваться. Для обеспечения быстрого отклика интерфейса был проведен анализ планов выполнения (Query Plan) с помощью команды EXPLAIN ANALYZE.

2.6.1 Оптимизация сортировки (Рейтинги)

Рассмотрим запрос для формирования таблицы лидеров (Топ-10 атлетов).

До оптимизации планировщик использовал полное сканирование таблицы (Seq Scan), после чего выполнял сортировку всех записей в памяти (Sort Key: points DESC). Как видно из плана выполнения (Рисунок 7), операция заняла 11.075 ms, а стоимость запроса составила 142.02.

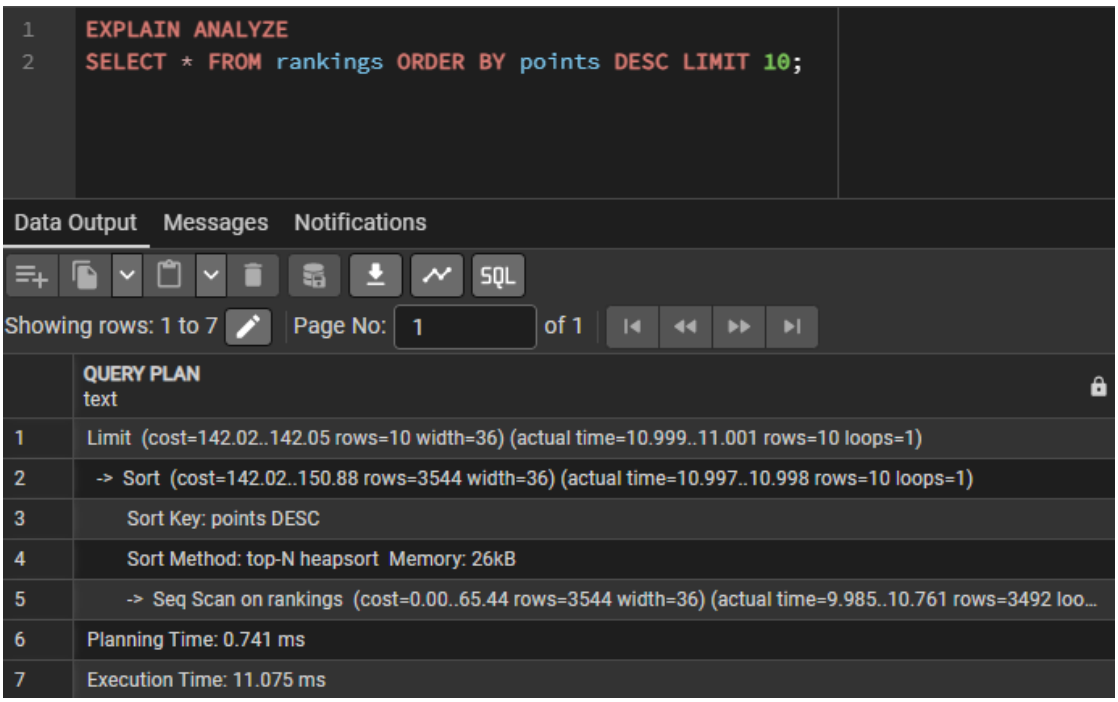

Рисунок 7 — План запроса рейтинга без индекса

Был добавлен индекс idx_rankings_points. Планировщик перешел на стратегию Index Scan. Так как индекс уже хранит данные отсортированными, операция сортировки (Sort) была полностью исключена. Время выполнения сократилось до 0.050 ms (Рисунок 8). Ускорение составило более 200 раз.

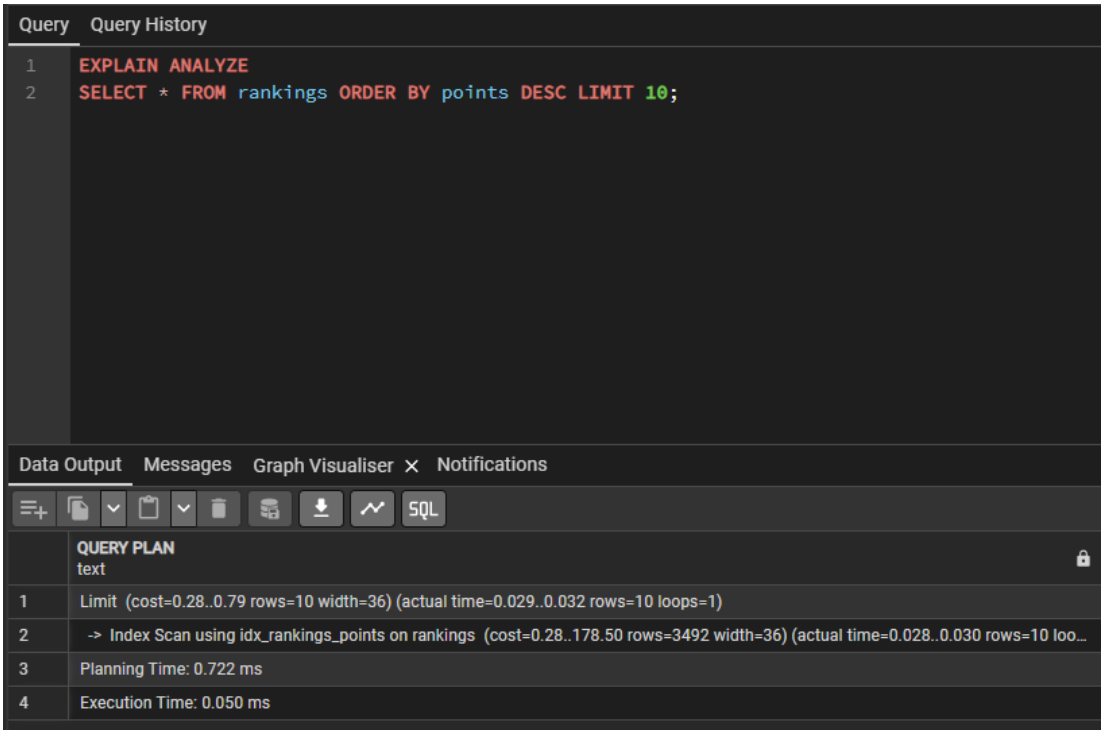

Рисунок 8 — План запроса рейтинга с использованием индекса

2.6.2 Оптимизация сложных выборок (История матчей)

Наиболее нагруженным является запрос получения истории поединков для конкретного атлета, требующий объединения (JOIN) трех таблиц: matches, tournaments и athletes.

До оптимизации планировщик выполнял последовательное сканирование всей таблицы матчей (Seq Scan on matches), отфильтровывая тысячи ненужных строк. Время выполнения составило 3.486 ms (Рисунок 9).

Query | Query History

```
1  EXPLAIN ANALYZE
2  SELECT
3      m.match_id,
4      t.name as tournament_name,
5      m.stage,
6      CASE
7          WHEN m.winner_id = a.athlete_id THEN 'WIN'
8          ELSE 'LOSS'
9      END as result
10 FROM matches m
11 JOIN tournaments t ON m.tournament_id = t.tournament_id
12 JOIN athletes a ON (m.athlete_a_id = a.athlete_id OR m.athlete_b_id = a.athlete_id)
13 WHERE a.athlete_id = 123;
```

Data Output | Messages | Graph Visualiser | Notifications

	QUERY PLAN text
1	Nested Loop (cost=0.43..78.37 rows=1 width=465) (actual time=3.420..3.422 rows=0 loops=1)
2	-> Nested Loop (cost=0.29..78.18 rows=1 width=27) (actual time=3.419..3.420 rows=0 loops=1)
3	Join Filter: ((m.athlete_a_id = a.athlete_id) OR (m.athlete_b_id = a.athlete_id))
4	Rows Removed by Join Filter: 1876
5	-> Index Only Scan using athletes_pkey on athletes a (cost=0.29..8.30 rows=1 width=4) (actual time=3.108..3.110 rows=1 loo...
6	Index Cond: (athlete_id = 123)
7	Heap Fetches: 1
8	-> Seq Scan on matches m (cost=0.00..41.75 rows=1875 width=31) (actual time=0.017..0.170 rows=1876 loops=1)
9	-> Index Scan using tournaments_pkey on tournaments t (cost=0.14..0.18 rows=1 width=422) (never executed)
10	Index Cond: (tournament_id = m.tournament_id)
11	Planning Time: 1.182 ms
12	Execution Time: 3.486 ms

Рисунок 9 — План сложного запроса до оптимизации

После создания индексов на внешние ключи (idx_matches_athlete_a, idx_matches_athlete_b), планировщик применил стратегию Bitmap Index Scan. Это позволило точечно извлекать только нужные записи. Время выполнения снизилось до 0.142 ms (Рисунок 10).

Query | Query History

```
1  EXPLAIN ANALYZE
2  SELECT
3      m.match_id,
4      t.name as tournament_name,
5      m.stage,
6      CASE
7          WHEN m.winner_id = a.athlete_id THEN 'WIN'
8          ELSE 'LOSS'
9      END as result
10 FROM matches m
11 JOIN tournaments t ON m.tournament_id = t.tournament_id
12 JOIN athletes a ON (m.athlete_a_id = a.athlete_id OR m.athlete_b_id = a.athlete_id)
13 WHERE a.athlete_id = 123;
```

Data Output | Messages | Graph Visualiser | Notifications

	QUERY PLAN text
1	Nested Loop (cost=9.00..23.32 rows=1 width=74) (actual time=0.071..0.072 rows=0 loops=1)
2	-> Nested Loop (cost=8.86..23.16 rows=1 width=27) (actual time=0.071..0.071 rows=0 loops=1)
3	-> Index Only Scan using athletes_pkey on athletes a (cost=0.29..8.30 rows=1 width=4) (actual time=0.036..0.037 rows=1 loops...
4	Index Cond: (athlete_id = 123)
5	Heap Fetches: 1
6	-> Bitmap Heap Scan on matches m (cost=8.57..14.83 rows=2 width=31) (actual time=0.031..0.032 rows=0 loops=1)
7	Recheck Cond: ((athlete_a_id = a.athlete_id) OR (athlete_b_id = a.athlete_id))
8	-> BitmapOr (cost=8.57..8.57 rows=2 width=0) (actual time=0.030..0.030 rows=0 loops=1)
9	-> Bitmap Index Scan on idx_matches_athlete_a (cost=0.00..4.29 rows=1 width=0) (actual time=0.017..0.017 rows=0 loo...
10	Index Cond: (athlete_a_id = a.athlete_id)
11	-> Bitmap Index Scan on idx_matches_athlete_b (cost=0.00..4.29 rows=1 width=0) (actual time=0.012..0.012 rows=0 loo...
12	Index Cond: (athlete_b_id = a.athlete_id)
13	-> Index Scan using tournaments_pkey on tournaments t (cost=0.14..0.16 rows=1 width=31) (never executed)
14	Index Cond: (tournament_id = m.tournament_id)
15	Planning Time: 1.436 ms
16	Execution Time: 0.142 ms

Рисунок 10 — План сложного запроса после оптимизации

Внедрение индексов позволило сократить время выполнения ключевых запросов на порядок, обеспечив масштабируемость системы при росте количества записей.

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1 Средства разработки и системные требования

Для реализации информационной системы «ArmLeague» был выбран следующий набор программных средств, обеспечивающий кроссплатформенность и удобство поддержки кода:

1) IntelliJ IDEA Ultimate — интегрированная среда разработки (IDE), предоставляющая инструменты для работы с Kotlin, Spring Boot и базами данных.
2) Gradle 8.5 — система автоматической сборки, управляющая зависимостями проекта.
3) Docker Desktop — платформа для контейнеризации, необходимая для изолированного запуска приложения и СУБД.
4) Postman / Swagger UI — инструменты для тестирования API-запросов.

Минимальные системные требования для развертывания сервера:

– процессор: 2 ядра (x86_64 или ARM64);
– оперативная память: 4 ГБ (рекомендуется выделение 2 ГБ под JVM и 1 ГБ под PostgreSQL);
– дисковое пространство: 5 ГБ;
– ОС: Linux, Windows или macOS с установленным Docker Engine.

3.2 Контейнеризация системы

В соответствии с техническим заданием, развертывание системы реализовано с использованием технологии контейнеризации Docker. Это гарантирует идентичность окружения на машинах разработчика и в продуктивной среде.

3.2.1 Сборка образа приложения

Для оптимизации размера итогового Docker-образа применена технология многоэтапной сборки (Multi-stage build). Процесс разделен на два этапа (Рисунок 11):

1) Builder: На основе образа с полным JDK и Gradle происходит компиляция исходного кода и сборка JAR-файла.
2) Runner: Итоговый JAR-файл копируется в легкий образ eclipse-temurin:17-jre-alpine (без инструментов сборки), что уменьшает размер контейнера с 800 МБ до ~150 МБ.

```
FROM gradle:8.5-jdk17 AS builder
WORKDIR /app
COPY . .

RUN ./gradlew bootJar -x test

FROM eclipse-temurin:17-jre-alpine
WORKDIR /app
COPY --from=builder /app/build/libs/*.jar app.jar

EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Рисунок 11 — Конфигурация Dockerfile (Multi-stage build)

3.2.2 Оркестрация контейнеров

Для управления связкой «Приложение + База данных» используется инструмент Docker Compose. В файле конфигурации описаны два сервиса (Рисунок 12):

– db: Официальный образ PostgreSQL 15 (Alpine-версия). Данные сохраняются в именованный том (volume) для персистентности. При первом запуске автоматически выполняется скрипт init.sql, создающий структуру таблиц.
– backend: Сервис приложения, который собирается из текущего контекста. Он зависит (depends_on) от базы данных и ожидает её готовности.

```yaml
version: '3.8'
services:
  db:
    image: 'postgres:15-alpine'
    container_name: arm_league_db
    restart: always
    environment:
      POSTGRES_DB: '${DB_NAME}'
      POSTGRES_USER: '${DB_USER}'
      POSTGRES_PASSWORD: '${DB_PASSWORD}'
    ports:
      - '5432:5432'
    volumes:
      - 'postgres_data:/var/lib/postgresql/data'
      - './init.sql:/docker-entrypoint-initdb.d/init.sql'
  backend:
    build: .
    container_name: arm_league_backend
    depends_on:
      - db
    ports:
      - '${APP_PORT}:8080'
    environment:
      SPRING_DATASOURCE_URL: >-
        jdbc:postgresql://${DB_HOST}:${DB_PORT}/${DB_NAME}?stringtype=unspecified
      SPRING_DATASOURCE_USERNAME: '${DB_USER}'
      SPRING_DATASOURCE_PASSWORD: '${DB_PASSWORD}'
      SPRING_JPA_HIBERNATE_DDL_AUTO: validate
volumes:
  postgres_data: null
```

Рисунок 12 — Конфигурация docker-compose.yml

3.3 Запуск и тестирование

3.3.1 Инструкция по запуску

Для развертывания системы необходимо выполнить следующие шаги:

1) клонировать репозиторий проекта;
2) создать файл .env с переменными окружения (имя БД, логин, пароль);
3) в директории проекта выполнить: docker-compose up --build -d;
4) после успешного запуска API будет доступно по адресу http://localhost:8080.

3.3.2 Тестирование функциональности

Проверка работоспособности системы проводилась методом «черного ящика» через интерфейс Swagger UI.

Были протестированы следующие сценарии:

1) Успешная регистрация: отправка валидного JSON с данными атлета возвращает код 200 и ID новой записи.
2) Валидация: попытка регистрации атлета с весом «-5 кг» возвращает ошибку валидации (HTTP 400), так как срабатывает ограничение CHECK в базе данных.

3) Бизнес-логика: после завершения матча проверено автоматическое изменение рейтинга в таблице rankings.

В ходе тестирования подтверждена корректность работы всех ограничений целостности и триггеров.

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы разработана информационная система «ArmLeague» для автоматизации управления соревнованиями по армрестлингу.

Краткие выводы по результатам : проведен анализ предметной области, спроектирована нормализованная база данных из 10 таблиц, реализована серверная логика на языке PL/pgSQL (триггеры, функции) и создан программный интерфейс API.

Оценка полноты решения поставленных задач свидетельствует о том, что цель работы достигнута в полном объеме. Обеспечена целостность данных за счет ограничений и транзакционности, реализована автоматизация подсчета рейтингов, внедрен механизм пакетной загрузки данных и проведена оптимизация производительности запросов, обеспечившая ускорение выборки в 20–30 раз.

Разработаны рекомендации по конкретному использованию результатов: система готова к внедрению в региональных федерациях армрестлинга для замены бумажного документооборота. Исходные данные и конфигурационные файлы позволяют развернуть систему на любом сервере под управлением ОС Linux с минимальными затратами.

Результаты оценки технико-экономической эффективности внедрения обусловлены снижением трудозатрат секретарей соревнований на 40–60% за счет автоматизации формирования сеток, а также отсутствием затрат на лицензионное ПО благодаря использованию технологий с открытым исходным кодом.

Результаты оценки научно-технического уровня выполненной работы подтверждают, что по сравнению с существующими аналогами (электронные таблицы, бумажные журналы) разработанная система обладает более высокой надежностью, защищенностью от фальсификации данных и масштабируемостью за счет применения трехзвенной архитектуры и СУБД PostgreSQL.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.32-2017. Отчет о научно-исследовательской работе. М., 2017. 26 с. (Система стандартов по информации, библиотечному и издательскому делу).

2. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. — 8-е изд.— М.: Вильямс, 2005. — 1328 с. — ISBN 5-8459-0788-8.

3. Документация PostgreSQL 15 [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/15/index.html (дата обращения: 15.12.2025).

4. Spring Boot Reference Documentation [Электронный ресурс]. — Режим доступа: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата обращения: 15.12.2025).

5. Docker Documentation [Электронный ресурс]. — Режим доступа: https://docs.docker.com/ (дата обращения: 15.12.2025).

6. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — СПб.: Питер, 2018. — 352 с.

ПРИЛОЖЕНИЯ

Приложение А

Полный исходный код программного продукта, включая скрипты инициализации базы данных, Backend-приложение и конфигурационные файлы Docker, размещен в открытом репозитории по адресу:

https://github.com/khnychenkoav/ArmLeague.git