МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)» (МАИ)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект

по дисциплине «Базы данных»

Тема: «разработка системы организации и проведения турниров CyberTournament»

Студент: Фокин Л.А.
Группа: М8О-301Б-23
Преподаватель: _____
Оценка: _____
Дата: _____
Подпись: _____

Москва 2025

РЕФЕРАТ

Данный курсовой проект представляет собой разработку информационной системы для организации и проведения киберспортивных турниров «CyberTournament». Система предназначена для хранения и анализа информации о профессиональных игроках, командных составах, турнирных событиях, результатах матчей и всех связанных сущностях. В качестве технологического стека используется PostgreSQL для работы с данными и backend на Go, который предоставляет REST API для взаимодействия с клиентами.

Основные задачи включают проектирование архитектуры приложения и структуры базы данных, создание логической и физической моделей, реализацию серверной логики средствами СУБД (триггеры, хранимые функции, представления) и разработку REST API с поддержкой стандартных операций над данными, аналитических выборок и пакетной загрузки справочников. В процессе работы был выполнен анализ предметной области киберспорта, сформулированы требования к системе, спроектирована БД с механизмами миграций, внедрены триггеры для журналирования изменений и динамического расчёта командных рейтингов, а также подготовлены SQL-запросы различной сложности с анализом их производительности.

Практическая ценность разработки заключается в получении рабочего прототипа, пригодного для дальнейшего развития в направлении коммерческого продукта для киберспортивных организаций и турнирных операторов. Кроме того, проект служит примером применения актуальных подходов к проектированию реляционных систем, созданию процедурных расширений SQL и построению интеграции между БД и прикладным слоем.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объект исследования — данные киберспортивных турниров: команды, игроки, матчи, результаты и статистика.

Предмет исследования — методы построения реляционных БД, серверная логика СУБД (триггеры, функции, представления) и связка базы данных с backend через REST API.

Цель работы — создать систему управления турнирной информацией, показав на практике проектирование схемы БД, работу с ограничениями целостности, написание аналитических SQL-запросов, автоматизацию через триггеры и разработку документированного API для доступа к данным.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Предметная область

В рамках проекта рассматривается ведение данных киберспортивных соревнований: от справочника дисциплин и учёта команд/игроков до регистрации команд на турниры и фиксации спортивных событий. По каждому турниру сохраняются матчи, их разбиение на отдельные игры (карты) и персональная статистика игроков, что позволяет формировать сводные показатели и отчёты. Дополнительно предусмотрен контроль изменений данных (аудит) и поддержка пакетной загрузки с протоколированием ошибок.

Предметная область включает следующие ключевые сущности:

- Дисциплины — перечень дисциплин соревнований.
- Команды — участники турниров.
- Профили команд — расширенные сведения о команде (1:1).
- Игроки — участники команд.
- Составы команд — принадлежность игроков к командам (N:M).
- Турниры — соревнования по выбранной дисциплине.
- Регистрации на турнир — заявки команд на участие.
- Матчи — встречи команд в рамках турнира.
- Игры (карты) матча — детализация матча на отдельные игры.
- Статистика игроков — показатели игрока в конкретной игре.

Транзакционные сущности (регистрации, матчи/игры и статистика) связывают справочники и отражают ход турниров, а журнал аудита и таблица ошибок импорта обеспечивают прослеживаемость изменений и контроль качества данных.

1.2 Постановка задачи

Цель — разработать информационную систему «CyberTournament», которая обеспечивает полный цикл работы с киберспортивными турнирами: от описания дисциплин и регистрации команд до фиксации результатов матчей и аналитической отчетности. Для этого необходимо:

1) Спроектировать нормализованную реляционную модель (PostgreSQL) для сущностей дисциплин, команд, игроков, составов, регистраций на турниры, матчей, карт и игровых статистик, заложить ограничения целостности (PK/FK, UNIQUE, CHECK, NOT NULL) и специализированные индексы.
2) Реализовать физическую модель с представлениями и функциями: расчет KDA, турнирные таблицы, агрегаты по результатам матчей и карьерной статистике игроков.
3) Внедрить триггеры аудита (INSERT/UPDATE/DELETE) и автоматические пересчеты производных показателей (например, рейтинги команд по составу).
4) Создать backend на Go (Gin) с REST API и Swagger-документацией, покрывающую CRUD-операции, отчеты и батчевый импорт данных, включая параметризацию запросов и безопасное выполнение транзакций.
5) Обеспечить производительность за счет индексов и оптимизации ключевых запросов; предоставить воспроизводимый запуск в Docker/docker-compose и сценарии наполнения базы сид-данными.

Результат должен быть пригоден для развёртывания и практического использования.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Архитектура информационной системы

2.1.1 Обоснование выбора стека технологий

В основе системы — реляционная СУБД PostgreSQL с опорой на транзакционность, строгие связи и серверную логику (триггеры, функции PL/pgSQL, представления) для аудита и аналитики. Выбор «чистого» SQL через pgx + sqlx даёт полный контроль над запросами и сложными выборками. Backend на Go (Gin) реализует REST-слой поверх БД, не дублируя её правила целостности, а документация поддерживается через Swagger (swag). Контейнеризация (Docker) и Makefile обеспечивают воспроизводимое окружение и автоматизацию сборки, запуска и загрузки сид-данных.

2.1.2 Описание слоёв приложения

Архитектура системы следует классическому многослойному принципу:

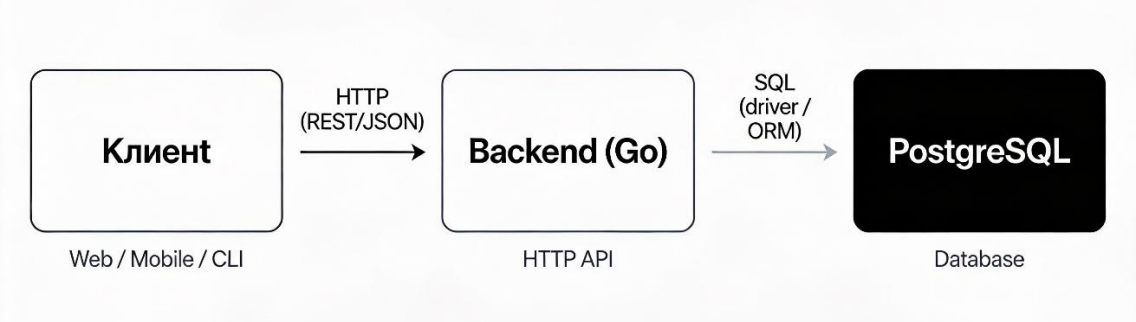

Рисунок 1 — Архитектура приложения

Реализация выглядит так:

1) Уровень хранения данных — PostgreSQL по schema.sql, использующая таблицы, индексы, ограничения, представления/функции/триггеры, транзакции.
2) Уровень доступа к данным — sqlx/pgx реализуют параметризованные SQL, CRUD, фильтры, пагинацию, отчётные выборки.
3) Уровень бизнес-логики — сервисы, которые реализуют прикладные сценарии, взаимодействующие с множествами сущностей: создание,

редактирование, удаление, получение данных с использованием валидации и возвратом ошибок.

4) Уровень представления — REST API на Gin. Обеспечивает обработчики, биндинг и валидацию JSON, единый формат ответа, Swagger документацию.

Классическое разделение позволяет приложению быть модульным, что сказывается на удобстве добавления/изменения функционала.

2.2 Проектирование логической модели БД

Проектирование базы данных выполнялось на основе анализа специфики проведения киберспортивных турниров. Главной целью стала разработка нормализованной схемы, которая исключает избыточность информации о матчах и игроках, а также обеспечивает целостность данных средствами СУБД.

2.2.1 Сущности и нормализация

Схема приведена к 3НФ:

1) атрибуты атомарны (1НФ).
2) неключевые атрибуты зависят от полного ключа (2НФ).
3) исключены транзитивные зависимости неключевых атрибутов (3НФ).

Ключевые предметные сущности:

1) Discipline — справочник дисциплин (код, название, team size, метаданные).
2) Team — команда в рамках дисциплины (название, тег, страна, рейтинг).
3) Player — игрок с персональными данными и MMR.
4) SquadMember — состав команды (роль, стендин, даты контракта, зарплата).

5) Tournament — турнир по дисциплине (даты, призовой фонд, статус, онлайн/оффлайн, сетка).
6) TournamentRegistration — заявка команды на турнир (статус, инвайт/квалификация, посев, снимок состава).
7) Match — матч турнира (пары команд, формат, стадия, победитель, forfeit, примечания).
8) MatchGame — карта/игра внутри матча (инфо о карте, победитель, счёт, длительность, прочая информация).
9) GamePlayerStat — статистика игрока на карте (kills/deaths/assists, урон, золото, MVP, KDA).

Операционные/служебные сущности:

1) TeamProfile — профиль команды (коуч, спонсоры, контакты, сайт).
2) AuditLog — аудит изменений по таблицам (операция, старое/новое значение, кто и когда).
3) BatchImportErrors — журнал ошибок пакетного импорта (источник, raw_data, сообщение об ошибке, время).

2.2.2 Связи и ограничения целостности

Для обеспечения согласованности и непротиворечивости данных на уровне СУБД реализован комплекс ограничений целостности:

– Суррогатные ключи (PRIMARY KEY) применяются во всех сущностях и обеспечивают стабильную идентификацию записей.
– Внешние ключи задают целостность ссылок с разным поведением при удалении: RESTRICT, CASCADE, SET NULL.
– UNIQUE ограничение предотвращает появление дублей среди записей разных сущностей.
– CHECK-ограничения контролируют логику данных, накладывая условия на некоторые поля.
– Используются значения по умолчанию для логических флагов.

Полная ER-диаграмма с установленными связями и вышеописанными

принципами выглядит так:

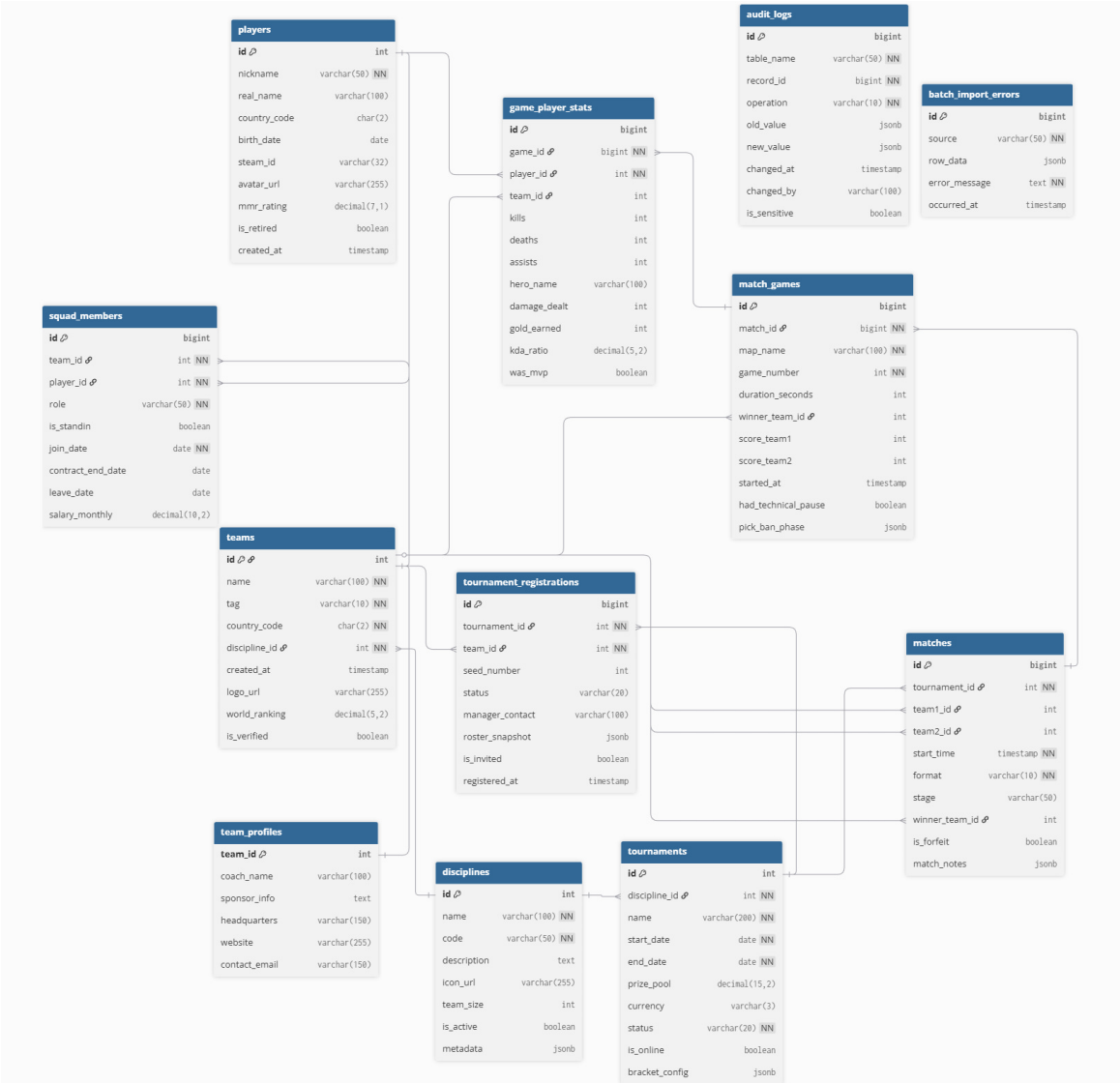

Рисунок 2 — Схема базы данных

Таким образом разработанная структура обеспечивает целостность данных.

2.3 Физическая структура базы данных

Физическая модель данных была реализована в СУБД PostgreSQL 16 в полном соответствии с ранее разработанной логической схемой и содержит 12 таблиц. Структура включает нормализованные таблицы с первичными и внешними ключами, ограничениями целостности (UNIQUE, CHECK), а также значениями по умолчанию. Для повышения производительности созданы индексы по часто используемым фильтрам. Гибкость хранения неструктурированных атрибутов обеспечивается за счет использования полей

типа JSONB, а вычисляемые показатели, такие как kda_ratio, реализованы на уровне СУБД как генерируемые столбцы.

2.3.1 Таблицы дисциплин, команд, игроков и составов

Этот блок таблиц описывает основных участников экосистемы:

- disciplines – справочник киберспортивных дисциплин, содержащий уникальные name и code, стандартный размер команды (team_size) и дополнительные метаданные в формате JSONB.
- teams – содержит информацию о командах, включая название, тег, страну и ссылку на дисциплину. Уникальность команды в рамках одной дисциплины обеспечивается ключом (tag, discipline_id).
- team_profiles – таблица для дополнительной информации о команде (тренер, спонсоры), связанная с teams отношением «один к одному».
- players – хранит данные об игроках: никнейм, реальное имя, страна, MMR и другие идентификаторы.
- squad_members – связующая таблица для реализации отношения «многие ко многим» между игроками и командами. Хранит информацию о роли игрока, статусе (stand-in) и датах контракта. Частичный индекс по полю leave_date IS NULL обеспечивает быстрый доступ к активным составам.

2.3.2 Таблицы турниров, регистраций, матчей и игр

Данные таблицы отвечают за организацию и проведение соревнований:

- tournaments – описывает турниры: название, даты проведения, призовой фонд и формат. Конфигурация турнирной сетки (например, single-elimination) хранится в JSONB-поле bracket_config.
- tournament_registrations – фиксирует заявки команд на участие в турнире. Содержит статус регистрации, номер посева и «слепок» состава команды (roster_snapshot) на момент заявки в формате JSONB.

– matches – хранит информацию о матчах в рамках турнира: участвующие команды, стадию, формат (bo3, bo5) и результат
– match_games – детализирует каждый отдельный матч, описывая сыгранные карты (игры). Содержит номер карты, счет, длительность, а также данные о стадии выбора/запрета героев (pick/ban) в формате JSONB.

2.3.3 Таблица статистики игроков на картах

Таблица game_player_stats является ключевой для сбора статистики. Каждая запись связывает конкретного игрока с одной игрой в матче и хранит его индивидуальные показатели: убийства (kills), смерти (deaths), помощи (assists), нанесенный урон и т.д. Для обеспечения актуальности данных коэффициент kda_ratio реализован как вычисляемый столбец (GENERATED ... STORED), обновляемый автоматически при изменении исходных метрик.

2.3.4 Журналы аудита

Для обеспечения надежности и отслеживания изменений в системе реализованы две служебные таблицы:

– audit_logs – выполняет функцию журнала изменений. С помощью триггера audit_log_changes она автоматически фиксирует все операции INSERT, UPDATE и DELETE в ключевых таблицах, сохраняя старые и новые значения в формате JSONB.
– batch_import_errors – используется для логирования ошибок, возникающих при массовой загрузке данных. Хранит исходную строку (row_data), текст ошибки и время, что упрощает диагностику и повторную обработку некорректных данных.

2.4 Программирование на стороне СУБД

Бизнес-логика, критичная для обеспечения целостности данных и формирования отчетности, была реализована непосредственно на уровне базы

данных PostgreSQL. Это включает в себя использование триггеров для аудита, хранимых функций для сложных расчетов и представлений (Views) для аналитических выборок.

2.4.1 Триггеры аудита изменений

Для обеспечения безопасности и отслеживаемости изменений на всех ключевых таблицах системы был настроен универсальный триггер audit_log_changes. Данный механизм автоматически перехватывает операции INSERT, UPDATE и DELETE, сохраняя в журнальную таблицу audit_logs полную информацию об изменении.

```
CREATE OR REPLACE FUNCTION audit_log_changes() RETURNS trigger AS $$
DECLARE
    v_new JSONB;
    v_old JSONB;
    v_pk  BIGINT;
BEGIN
    IF TG_OP IN ('INSERT', 'UPDATE') THEN
        v_new := to_jsonb(NEW);
    END IF;
    IF TG_OP IN ('UPDATE', 'DELETE') THEN
        v_old := to_jsonb(OLD);
    END IF;

    v_pk := COALESCE(
        (v_new ->> 'id')::BIGINT,
        (v_old ->> 'id')::BIGINT,
        (v_new ->> 'team_id')::BIGINT,
        (v_old ->> 'team_id')::BIGINT
    );

    IF TG_OP = 'INSERT' THEN
        INSERT INTO audit_logs(table_name, record_id, operation, new_value, changed_by)
        VALUES (TG_TABLE_NAME, v_pk, TG_OP, v_new, current_user);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO audit_logs(table_name, record_id, operation, old_value, new_value, changed_by)
        VALUES (TG_TABLE_NAME, v_pk, TG_OP, v_old, v_new, current_user);
        RETURN NEW;
    ELSE
        INSERT INTO audit_logs(table_name, record_id, operation, old_value, changed_by)
        VALUES (TG_TABLE_NAME, v_pk, TG_OP, v_old, current_user);
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Рисунок 3 — Код триггера

2.4.2 Хранимые функции

Для выполнения ресурсоемких вычислений на стороне сервера были разработаны следующие хранимые функции:

– fn_player_kda(player_id) — рассчитывает агрегированный коэффициент эффективности (KDA) игрока на основе всей накопленной статистики.
– fn_tournament_standings(tournament_id) — формирует актуальную турнирную таблицу, подсчитывая количество сыгранных матчей, побед, поражений и технических проигрышей для каждой команды.
– refresh_team_rating(team_id) — процедура автоматического пересчета глобального рейтинга команды на основе среднего MMR её текущего состава. Вызов данной функции инициируется триггерами при изменении состава или индивидуального рейтинга игрока.

```
CREATE OR REPLACE FUNCTION fn_player_kda(p_player_id INT)
RETURNS DECIMAL(10,2) AS $$
DECLARE
    v_kills BIGINT;
    v_assists BIGINT;
    v_deaths BIGINT;
BEGIN
    SELECT COALESCE(SUM(kills),0), COALESCE(SUM(assists),0), COALESCE(SUM(deaths),0)
    INTO v_kills, v_assists, v_deaths
    FROM game_player_stats
    WHERE player_id = p_player_id;

    IF v_deaths = 0 THEN
        RETURN (v_kills + v_assists)::DECIMAL(10,2);
    END IF;
    RETURN ((v_kills + v_assists)::DECIMAL) / v_deaths;
END;
$$ LANGUAGE plpgsql STABLE;
```

Рисунок 4 — Функция fn_player_kda

Функция fn_player_kda является хорошим примером простой функции, которая возвращает одно значение типа Decimal.

```sql
CREATE OR REPLACE FUNCTION fn_tournament_standings(p_tournament_id INT)
RETURNS TABLE (
    team_id INT,
    matches_played BIGINT,
    wins BIGINT,
    losses BIGINT,
    forfeits BIGINT
) AS $$
BEGIN
    RETURN QUERY
    WITH teams_in_matches AS (
        SELECT m.id AS match_id, m.tournament_id, m.team1_id AS team_id, m.winner_team_id, m.is_forfeit
        FROM matches m
        WHERE m.tournament_id = p_tournament_id
        UNION ALL
        SELECT m.id, m.tournament_id, m.team2_id AS team_id, m.winner_team_id, m.is_forfeit
        FROM matches m
        WHERE m.tournament_id = p_tournament_id
    )
        SELECT tim.team_id,
               COUNT(*) AS matches_played,
               COUNT(*) FILTER (WHERE tim.winner_team_id = tim.team_id) AS wins,
               COUNT(*) FILTER (WHERE tim.winner_team_id IS NOT NULL AND tim.winner_team_id <> tim.team_id) AS losses,
               COUNT(*) FILTER (WHERE tim.is_forfeit) AS forfeits
        FROM teams_in_matches tim
        WHERE tim.team_id IS NOT NULL
        GROUP BY tim.team_id
        ORDER BY wins DESC, losses ASC;
END;
$$ LANGUAGE plpgsql STABLE;
```

Рисунок 5 — Функция fn_tournament_standings

Функция fn_tournament_standings выступает примером сложной функции, которая возвращает Table.

2.4.3 Представления

Для упрощения доступа к агрегированным данным и снижения сложности SQL-запросов со стороны приложения были созданы следующие представления (Views):

- v_active_rosters — возвращает список текущих актуальных составов команд (без учета бывших игроков).
- v_match_results — предоставляет сводные результаты матчей, агрегируя данные по всем сыгранным картам.
- v_player_career_stats — отображает суммарную карьерную статистику игрока (всего убийств, смертей, урона и средний KDA).

2.5 Описание API и взаимодействия с БД

Серверная часть приложения (Backend) реализована на языке Go с использованием фреймворка Gin и архитектурного паттерна, разделяющего

систему на слои: API, Service и Repository. Взаимодействие с базой данных осуществляется через библиотеки sqlx и pgx, что позволяет использовать параметризованные SQL-запросы для безопасности и производительности. Репозитории возвращают типизированные доменные ошибки (например, ErrMatchNotFound), которые затем обрабатываются на уровне сервисов или API.

2.5.1 REST-контроллеры

В пакете api реализованы обработчики HTTP-запросов (хендлеры) для выполнения CRUD-операций над основными сущностями системы: дисциплинами, командами, игроками, турнирами, матчами и статистикой. Например, модуль обработки матчей валидирует входные данные, парсит параметры фильтрации из URL, вызывает методы бизнес-логики сервиса MatchService и формирует унифицированный JSON-ответ в формате {data: ..., meta: ...} либо возвращает структуру ошибки. Аналогичным образом организована работа контроллеров для всех остальных сущностей.

2.5.2 Механизм пакетной загрузки данных

Функционал массового импорта данных (Batch Import) сосредоточен в сервисе ImportService. Специализированные эндпоинты в модуле утилит принимают массивы объектов (игроки, команды, матчи, статистика) и последовательно сохраняют их через вызовы соответствующих доменных сервисов. В случае возникновения ошибок (например, нарушение ограничений целостности) проблемные записи фиксируются функцией логирования в специальной таблице batch_import_errors. Это позволяет не прерывать весь процесс загрузки и предоставляет возможность последующего анализа причин сбоев.

2.5.3 Документирование API

Для автоматического формирования документации используется спецификация OpenAPI (Swagger). Она генерируется на основе аннотаций в

коде обработчиков и публикуется через middleware gin-swagger по адресу /swagger/*. Веб-интерфейс Swagger UI предоставляет возможность интерактивного тестирования всех методов API (получение списков, фильтрация, создание записей) без необходимости использования сторонних HTTP-клиентов.

2.6 Оптимизация и анализ производительности

Оптимизация производительности системы выполнена за счет применения целевых индексов для типовых запросов (выборки турниров, матчей, составов, статистики). Тесты производительности проводились, используя запросы к базе данных через ПО Datagrip.

2.6.1 Индексы и обоснование выбора

Для ускорения доступа к данным были созданы следующие индексы, покрывающие наиболее частотные условия фильтрации и соединения таблиц:

- idx_teams_discipline — индекс по полю discipline_id в таблице команд. Обеспечивает быструю выборку всех команд конкретной дисциплины.
- idx_squad_active — частичный индекс с условием WHERE leave_date IS NULL. Позволяет мгновенно получать текущие составы команд, игнорируя исторические данные.
- idx_registrations_tournament, idx_registrations_team — индексы для фильтрации заявок по идентификаторам турнира и команды соответственно.
- idx_matches_tournament, idx_matches_start_time — индексы для выборки списка матчей конкретного турнира и их сортировки или фильтрации по времени начала.
- idx_stats_player, idx_stats_game — индексы для агрегации и выборки статистики по игроку или конкретной игре.
- idx_audit_date — индекс типа BRIN по полю даты в таблице аудита. Выбран из-за его компактности и высокой эффективности для

монотонно возрастающих данных, что критично для больших объемов логов.

– idx_import_errors_source — композитный индекс для быстрого поиска ошибок импорта по источнику и времени возникновения.

Использование данных индексов позволяет избежать Seq Scan, используя Index Scan и Bitmap Scan.

2.6.2 Пример и анализ плана выполнения

Для анализа плана выполнения был выбрал запрос получения статистики игрока в рамках одной игры. На этот запрос влияет индекс idx_stats_player, что можно увидеть на сравнении до индекса и после:

Рисунок 6 — Анализ до индекса

Обратим внимание на использованный Scan, а также стоимость выполнения, время планирования и время выполнения запроса.

Рисунок 7 — Анализ после индекса

Все озвученные показатели снизились, что подтверждает эффективность индекса для этого запроса.

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1 Инструменты разработки и программное окружение

Реализация серверной части информационной системы выполнена на компилируемом языке программирования Go с использованием веб-фреймворка Gin, обеспечивающего высокую производительность обработки HTTP-запросов. Взаимодействие с базой данных реализовано посредством библиотек pgx (драйвер PostgreSQL) и sqlx (расширение для удобного маппинга данных в структуры).

В качестве системы управления зависимостями используется стандартный модуль go mod. Для документирования API применяется спецификация OpenAPI (Swagger), которая генерируется автоматически на основе комментариев в коде с помощью инструментов swag и gin-swagger.

3.2 Контейнеризация и развёртывание

Для обеспечения переносимости и упрощения развертывания системы используется технология контейнеризации Docker. Оркестрация сервисов описывается в файле docker-compose.yml, который определяет конфигурацию двух основных контейнеров:

1) db — контейнер с СУБД PostgreSQL. При первом запуске выполняется автоматическая инициализация схемы базы данных из файла schema.sql, смонтированного в директорию инициализации контейнера.
2) backend — контейнер с приложением API, который собирается из исходного кода (Dockerfile).

Развертывание системы осуществляется командой docker-compose up, которая выполняет сборку образа приложения и запуск всех сервисов в единой виртуальной сети. Сервис API доступен по порту, указанному в переменной окружения HTTP_ADDR (по умолчанию 8000), а взаимодействие с базой данных происходит по внутреннему сетевому имени сервиса.

3.3 Тестирование функциональности

Проверка работоспособности системы производится через веб-интерфейс Swagger UI, доступный по адресу /swagger/index.html. Тестирование включает следующие сценарии:

1) Проверка CRUD-операций: выполнение запросов на создание, чтение, обновление и удаление сущностей (дисциплины, команды, игроки, турниры, матчи). Корректность подтверждается кодами HTTP-ответов (200 OK, 201 Created) и телом ответа в формате JSON.
2) Тестирование аналитических функций: верификация корректности работы сложных SQL-запросов и представлений (получение активных составов, турнирных таблиц, расчет KDA и карьерной статистики).
3) Проверка пакетной загрузки (Batch Import): тестирование массового импорта данных через специализированный эндпоинт. Система проверяется на устойчивость к ошибкам: некорректные записи отсеиваются, а информация о сбоях сохраняется в таблицу batch_import_errors для последующего анализа.

Дополнительно для нагрузочного тестирования и проверки идемпотентности запросов могут использоваться внешние HTTP-клиенты (например, CURL или Postman).

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была спроектирована и разработана информационная система для управления данными киберспортивных турниров. Целью работы являлось создание надежного и производительного backend-приложения с реляционной базой данных, способного обеспечивать целостность данных и эффективную обработку сложной статистики матчей.

В результате исследования предметной области была построена нормализованная структура базы данных, включающая более 10 взаимосвязанных сущностей. Реализована строгая ссылочная целостность за счет использования внешних ключей и каскадных операций. Для обеспечения достоверности данных внедрены механизмы аудита изменений.

Ключевым результатом работы стала реализация сложной бизнес-логики на уровне базы данных и API:

1) Производительность — использование индексов существенно сократило стоимость выполнения аналитических запросов, что подтверждено анализом планов выполнения.
2) Функциональность — REST API предоставляет полный набор инструментов для выполнения поставленных задач проекта.
3) Аналитика — система позволяет формировать динамические отчеты, такие как турнирные таблицы, карьерная статистика игроков и анализ эффективности команд.

Практическая значимость работы заключается в создании масштабируемого решения, готового к развертыванию в контейнеризированной среде. Поставленные в техническом задании цели достигнуты в полном объеме, разработанная система соответствует всем заявленным требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.32-2017. Отчет о научно-исследовательской работе. М., 2017. 26 с.

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 16 [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/16/index.html (дата обращения: 17.12.2025).

4. Go Documentation [Электронный ресурс]. — Режим доступа: https://golang.org/doc/ (дата обращения: 17.12.2025).

5. Docker Documentation [Электронный ресурс]. — Режим доступа: https://docs.docker.com/ (дата обращения: 17.12.2025).

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

7. Gin Web Framework Documentation [Электронный ресурс]. — Режим доступа: https://gin-gonic.com/en/docs/ (дата обращения: 17.12.2025).

8. PGX [Электронный ресурс]. — Режим доступа: https://github.com/jackc/pgx (дата обращения: 17.12.2025).

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/paintingpromisesss/db_course_project.git