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 организации и проведения киберспортивных турниров»

Выполнил:
студент группы М8О-301Б-23
Фокин Леонид Александрович

_____
(подпись)

Руководитель:
_____

Оценка:_____

_____
(подпись)

Москва 2025

РЕФЕРАТ

Пояснительная записка выполнена в 1 части и содержит: страниц — 27, иллюстраций — 0, приложений — 3, использованных источников — 8.

база данных; СУБД; PostgreSQL; Go; Gin; REST API; SQL; киберспорт; турниры; аудит; Docker; Swagger

Объект исследования — процессы организации и учета киберспортивных турниров, команд, игроков и матчей.

Цель работы — разработать информационную систему с реляционной базой данных для автоматизации ведения дисциплин, команд, составов, регистраций на турниры, расписаний матчей и статистики игроков.

В ходе работы спроектирована инфологическая и даталогическая модели, создан DDL-скрипт с ограничениями целостности, представлениями и функциями; реализованы триггеры аудита и автоматического пересчета рейтингов. Разработан сервер на Go (Gin), обеспечивающий REST API для CRUD-операций, отчетов и батчевой загрузки данных с логированием ошибок; подготовлена Swagger-документация.

Результатом является работоспособная система учета киберспортивных турниров, поддерживающая регистрацию команд и игроков, формирование матчей и карт, сбор статистики, построение отчетов (ростеры, результаты, статистика игроков) и безопасную массовую загрузку данных. Система может применяться для цифровизации турнирной деятельности киберспортивных лиг и клубов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Организация киберспортивных турниров требует учета сложного массива данных, от заявок до личной статистики. Ручное управление этими процессами вызывает ошибки и споры. Актуальность темы обусловлена необходимостью внедрения системы для автоматизации судейства, централизованного хранения истории и точного расчета разных показателей, что исключит человеческий фактор и повысит прозрачность соревнований.

Объект исследования — процесс организации и проведения киберспортивных турниров.

Предмет исследования — методы проектирования реляционных баз данных и серверных компонентов для обработки киберспортивной статистики.

Цель работы — разработка информационной системы для автоматизации регистрации команд и игроков, управления сеткой матчей, фиксации результатов по картам и вычисления статистики.

Для достижения цели решены задачи:

1) выполнен анализ предметной области и определены сущности (дисциплины, команды, игроки, заявки, матчи, карты, статистика);
2) спроектирована нормализованная схема БД и инфологическая модель;
3) реализована физическая модель в PostgreSQL с ограничениями целостности, представлениями и функциями;
4) разработаны триггеры аудита и перерасчета рейтингов, реализована серверная логика на Go (Gin);
5) создан REST API с документацией Swagger для CRUD, отчетов и батчевого импорта данных;
6) проведена оптимизация SQL-запросов и применены индексы.

Практическая значимость работы — готовый программный продукт для киберспортивных лиг и клубов, повышающий эффективность организации турниров и точность статистики.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Предметная область киберспортивных турниров

Процесс организации киберспортивных соревнований представляет собой сложную систему взаимосвязанных этапов: от объявления дисциплины и регистрации команд до проведения матчей и подведения итогов. Ключевой особенностью предметной области является необходимость учета большого объема разнородных данных:

- Справочная информация: перечень дисциплин, реестр профессиональных команд и профили игроков.
- Турнирная документация: регламенты, сетки соревнований, расписание матчей и протоколы заявок.
- Статистика: детальные показатели эффективности игроков и результаты каждой сыгранной карты.

В процессе проведения турниров критически важно обеспечивать целостность данных: контролировать составы команд на момент матча, исключать конфликты в расписании и гарантировать корректность итоговых протоколов. Ошибки в учете статистики или нарушении регламента (например, участие незаявленного игрока) недопустимы, так как напрямую влияют на распределение призового фонда и рейтинги команд.

Для автоматизации этих процессов требуется информационная система, способная централизованно хранить историю турниров, фиксировать все изменения данных (аудит) и предоставлять оперативные аналитические отчеты для судей и организаторов.

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), сохранить логи загрузок с ошибками.
4) Создать серверную часть 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 практического использования лигами и клубами: ускорять подготовку и проведение турниров, повышать точность и прозрачность статистики, упрощать интеграцию через документированный API.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Архитектура информационной системы

2.1.1 Обоснование выбора стека технологий

Для реализации системы был выбран стек, обеспечивающий баланс между производительностью и скоростью разработки, а также полный контроль над взаимодействием с базой данных:

- Язык Go (Golang). Выбран за строгую статическую типизацию и компиляцию в один бинарный файл, что упрощает развертывание. Язык гарантирует высокую производительность и предсказуемое потребление памяти, что критично для серверных компонентов.
- Веб-фреймворк Gin. Обеспечивает минимальные накладные расходы при обработке HTTP-запросов. Фреймворк предоставляет понятный механизм middleware и быстрый JSON-биндинг.
- Библиотеки SQLX + pgx. Использование чистого SQL вместо ORM обусловлено необходимостью писать сложные аналитические запросы. SQLX убирает рутину сканирования данных в структуры, сохраняя возможность тонкой настройки запросов к PostgreSQL.
- СУБД PostgreSQL. Реляционная база данных с поддержкой сложной логики на уровне хранилища. Используются специфические возможности: триггеры для аудита, хранимые функции на PL/pgSQL и материализованные представления для статистики.
- Swagger (swag). Позволяет генерировать актуальную документацию API прямо из комментариев кода. Это упрощает интеграцию фронтенда и тестирование эндпоинтов через веб-интерфейс.
- Docker и Makefile. Контейнеризация гарантирует идентичность окружений разработки и демонстрации. Makefile автоматизирует рутинные задачи: накат миграций, загрузку тестовых данных и сборку проекта.

2.1.2 Описание слоёв приложения

Архитектура приложения построена по принципу разделения ответственности, где каждый компонент выполняет строго определенный набор функций. Взаимодействие между модулями организовано иерархически: от обработки внешних HTTP-запросов до выполнения низкоуровневых операций с базой данных. Такая структура обеспечивает модульность, упрощает тестирование и позволяет независимо модифицировать логику работы отдельных частей системы.

Слои и их реализация:

– API слой (Gin, пакет api). Обработчики регистрируют маршруты, принимают HTTP-запросы, биндинг/валидация JSON, отдают ответы в унифицированном формате (data/meta или error). Примеры: DisciplineHandler, MatchHandler, UtilityHandler (отчеты, batch-import). Swagger (swag + gin-swagger) публикует интерактивную OpenAPI-документацию.

– Сервисный слой (пакет service). Инкапсулирует бизнес-правила: нормализация строковых полей, проверки дат и идентичности команд, значения по умолчанию форматов матчей, пересчет лимитов пагинации. Сервисы оркестрируют репозитории и вспомогательные компоненты (отчеты, импорт), не зная деталей SQL. При ошибках возвращают доменные ErrNotFound или валидационные сообщения; сложные операции (batch-импорт) выполняют пошагово с накоплением ошибок и логированием.

– Слой данных (пакет repository, SQLX/pgx). Параметризованные запросы к PostgreSQL без ORM-магии: CRUD, фильтры, пагинация, отчетные выборки, вызовы функций и представлений (KDA, standings, v_match_results). Возвращают типовые ошибки (ErrDisciplineNotFound, ErrMatchNotFound и др.). Для batch-импорта ошибки пишутся в таблицу batch_import_errors.

– Хранилище (PostgreSQL, схема schema.sql). Нормализованные таблицы с PK/FK/UNIQUE/CHECK/NOT NULL, индексы для частых фильтров, JSONB для метаданных, представления и функции на PL/pgSQL, триггеры аудита и перерасчета рейтингов. ACID-гарантии и транзакционность обеспечивают согласованность при многошаговых операциях.
– Инфраструктура и утилиты. Docker/compose для воспроизводимого стенда (БД + backend). Сиды и скрипты для начального наполнения. Make-задачи для сборки, применения схемы, запуска и импорта данных. Swagger UI для быстрого тестирования API.

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) Tournament — турнир по дисциплине (даты, призовой фонд, статус, онлайн/оффлайн, сетка).
5) Match — матч турнира (пары команд, формат, стадия, победитель, forfeit, примечания).
6) GamePlayerStat — статистика игрока на карте (kills/deaths/assists, урон, золото, MVP, KDA).

2.2.2 Связи и ограничения целостности

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

Первичные ключи и идентификация — все сущности системы (дисциплины, команды, игроки, турниры, матчи и др.) используют суррогатные первичные ключи (PRIMARY KEY), генерируемые автоматически (GENERATED ALWAYS AS IDENTITY). Это гарантирует стабильность ссылок независимо от изменения бизнес-атрибутов.

Внешние ключи и каскадные операции — связи между таблицами реализованы через внешние ключи (FOREIGN KEY) с различными политиками удаления:

– RESTRICT — используется для справочников (дисциплины), чтобы предотвратить случайное удаление записей, на которые есть ссылки (связи teams.discipline_id, tournaments.discipline_id).
– CASCADE — применяется для подчиненных сущностей, жизненный цикл которых неразрывно связан с родителем. При удалении команды удаляются ее профиль и история составов; при удалении турнира — регистрации и матчи; при удалении матча — игры и статистика.
– SET NULL — используется в матчах для ссылок на команды (team1_id, team2_id), чтобы сохранить историю матча даже в случае расформирования команды.

Уникальные индексы и бизнес-правила — для предотвращения дублирования данных наложены ограничения уникальности (UNIQUE):

– в таблице дисциплин: поля name и code;
– в таблице команд: комбинация (tag, discipline_id);
– в регистрациях на турнир: пара (tournament_id, team_id);
– в структуре матча: номер игры в рамках матча (match_id, game_number);
– в статистике: запись игрока в рамках конкретной игры (game_id, player_id).

Проверка допустимости значений (CHECK) — ограничения CHECK контролируют логическую корректность данных:

– валидация дат: дата окончания турнира не может быть раньше начала; дата ухода игрока из команды (leave_date) должна быть позже даты вступления;
– валидация участников матча: запрет матча команды самой с собой (team1_id <> team2_id);
– числовые ограничения: размер команды (team_size) должен быть положительным.

Значения по умолчанию и вычисляемые поля — для логических флагов (is_active, is_verified, is_online) и числовых метрик установлены значения по умолчанию. Поле kda_ratio в таблице статистики реализовано как вычисляемый столбец (GENERATED ALWAYS AS ... STORED), что гарантирует автоматический пересчет коэффициента при изменении убийств, смертей или помощи.

Индексирование — для ускорения проверок целостности и выборок созданы индексы на всех полях внешних ключей, а также специализированные индексы:

– частичный индекс на таблице squad_members (WHERE leave_date IS NULL) для быстрого доступа к текущим составам;

– индексы по временным меткам (start_time, registered_at) для сортировки списков.

Таким образом, разработанная структура обеспечивает целостность данных за счет использования внешних ключей и каскадных операций удаления. Полная ER-диаграмма спроектированной базы данных представлена в Приложении А.

2.3 Физическая структура базы данных

Физическая модель данных была реализована в СУБД PostgreSQL в полном соответствии с ранее разработанной логической схемой. 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. Такой подход гарантирует унификацию правил обработки данных независимо от внешнего приложения и значительно ускоряет выполнение аналитических запросов.

2.4.1 Триггеры аудита изменений

Для обеспечения безопасности и отслеживаемости изменений на всех ключевых таблицах системы (включая teams, players, matches, game_player_stats и др.) 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: тип операции, старое и новое состояние записи (в формате JSON), временную метку и идентификатор пользователя.

Это позволяет проводить детальный разбор инцидентов и восстанавливать хронологию событий.

2.4.2 Хранимые функции и расчёт показателей

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.

2.4.3 Представления для аналитических отчётов

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 Взаимодействие 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-обработчики

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 игроков и турниров.

2.5.2 Механизм пакетной загрузки данных и логирование ошибок

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 Анализ производительности

Оптимизация производительности системы выполнена за счет применения целевых индексов для типовых запросов (выборки турниров, матчей, составов, статистики) и использования материализованной логики в виде представлений. Все SQL-запросы параметризованы, что позволяет планировщику PostgreSQL кэшировать и эффективно переиспользовать планы выполнения.

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 (Partial Index) с условием WHERE leave_date IS NULL. Позволяет мгновенно получать текущие составы команд, игнорируя исторические данные об ушедших игроках.
- 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_audit_date — индекс типа BRIN по полю даты в таблице аудита. Выбран из-за его компактности и высокой эффективности для монотонно возрастающих данных (временных рядов), 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 полного сканирования таблиц (Seq Scan) в ключевых сценариях использования, заменяя его на более эффективные Index Scan или Bitmap Scan.

2.6.2 Примеры планов выполнения (EXPLAIN)

Анализ планов выполнения запросов (Query Plan) подтверждает эффективность выбранной стратегии индексации:

1) Список матчей турнира с сортировкой. Планировщик использует комбинацию индексов idx_matches_tournament и idx_matches_start_time (или эффективный Index Scan по одному из них с фильтрацией), что исключает необходимость дорогостоящей сортировки всего массива данных в памяти (Sort).
2) Получение текущих составов (v_active_rosters). Запрос использует Bitmap Index Scan по частичному индексу idx_squad_active, после чего выполняет эффективное соединение (Nested Loop) с таблицами команд и игроков, выбирая только актуальные записи.
3) Карьерная статистика игрока (v_player_career_stats). Для агрегации данных применяется Bitmap Heap Scan с предварительным поиском по индексу idx_stats_player. Это позволяет читать только необходимые страницы данных статистики конкретного игрока, игнорируя остальные миллионы записей.
4) Расчет турнирной таблицы (fn_tournament_standings). Функция выполняет два прохода по индексу idx_matches_tournament (для команд Team1 и Team2) с последующим объединением (Append) и

агрегацией результатов. При больших объемах данных это на порядки быстрее последовательного чтения всей таблицы матчей.

Для подтверждения результатов оптимизации в Приложении Б приведены сравнительные планы выполнения (EXPLAIN ANALYZE) для наиболее ресурсоемкого запроса агрегации статистики, демонстрирующие сокращение времени выполнения и количества операций ввода-вывода после создания целевого индекса.

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. Сборка проекта и выполнение служебных команд автоматизированы через утилиту Make.

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 --build, 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 и верификация 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).

Интерфейс автоматически сгенерированной документации API показан в Приложении В.

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 транзакционной обработки и журналирования изменений (аудит).

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

1) Производительность — благодаря использованию материализованных представлений и целевых индексов (B-Tree, BRIN, Partial Index), удалось значительно сократить стоимость выполнения аналитических запросов, что подтверждено анализом планов выполнения (EXPLAIN ANALYZE).
2) Функциональность — разработанный REST API предоставляет полный набор инструментов для администрирования турниров, включая пакетную загрузку больших объемов данных с обработкой ошибок.
3) Аналитика — система позволяет формировать динамические отчеты, такие как турнирные таблицы, карьерная статистика игроков и анализ эффективности команд в реальном времени.

Практическая значимость работы заключается в создании масштабируемого решения, готового к развертыванию в контейнеризированной среде. Полученные навыки проектирования баз

данных, оптимизации SQL-запросов и разработки микросервисной архитектуры на языке Go демонстрируют готовность к решению профессиональных задач в сфере разработки высоконагруженных информационных систем.

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).

ПРИЛОЖЕНИЕ А

Схема базы данных

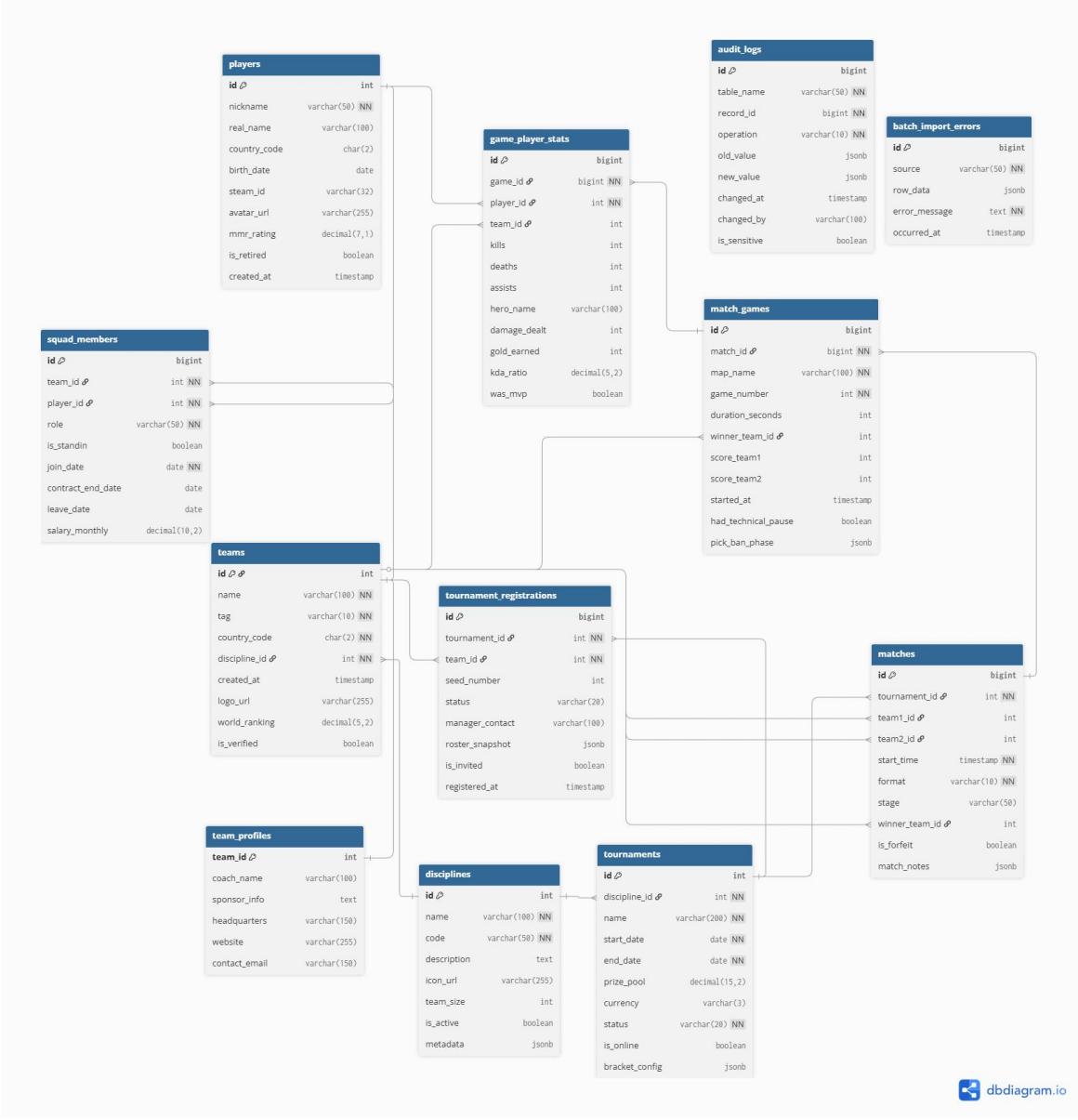

Рисунок А.1 – ER-диаграмма базы данных

ПРИЛОЖЕНИЕ Б

Анализ производительности запросов

Рисунок Б.1 – Анализ до создания индекса

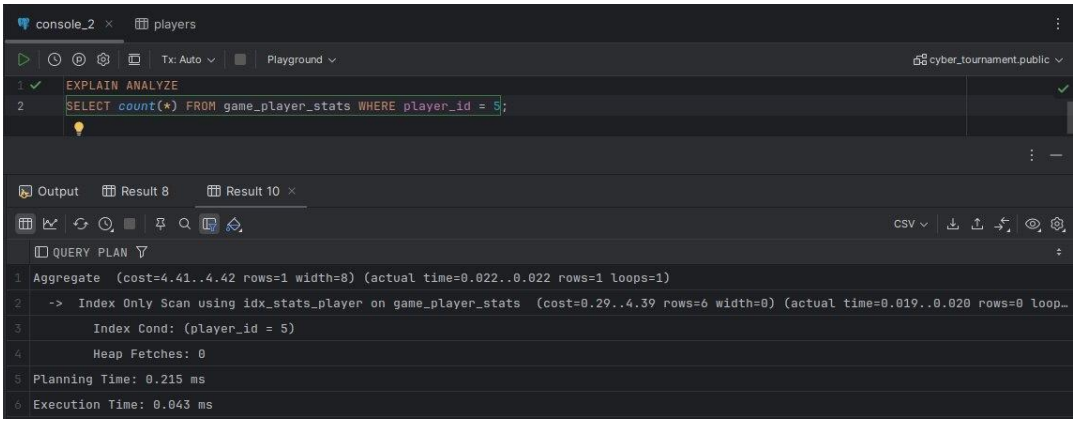

Рисунок Б.2 – Анализ после создания индекса

ПРИЛОЖЕНИЕ В

Интерфейс взаимодействия с API

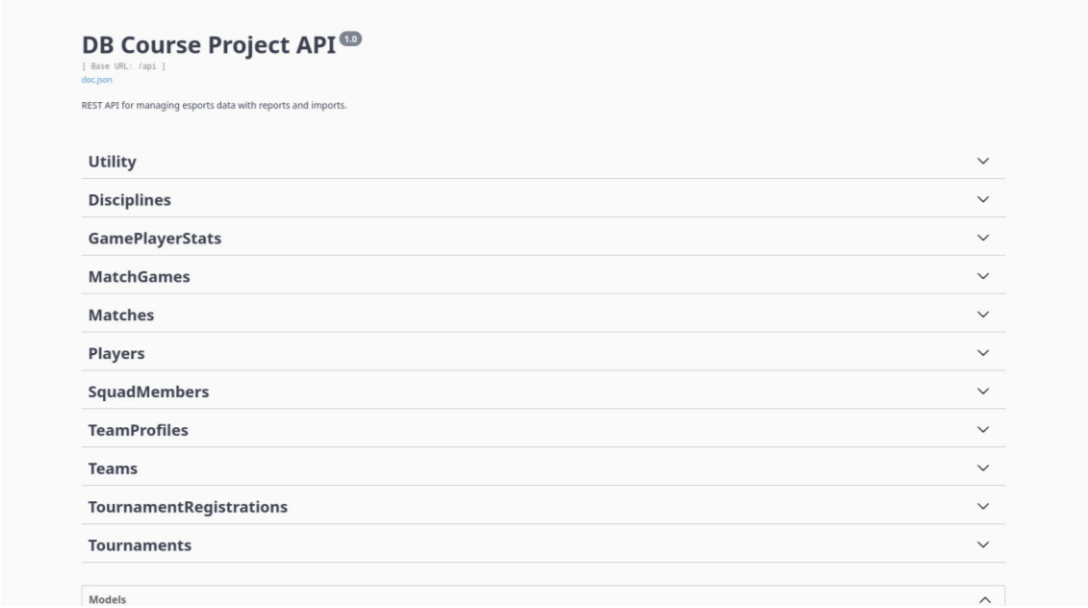

Рисунок В.1 – Все группы Swagger UI

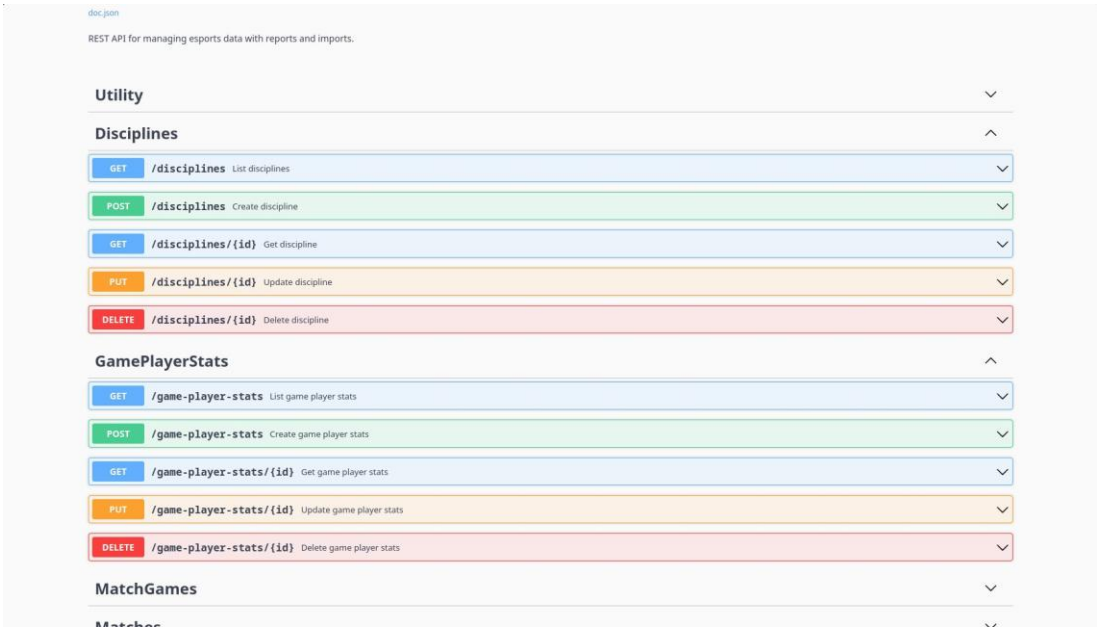

Рисунок В.2 – Детализация методов в группах Disciplines и GamePlayerStats