МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных сервиса футбольной статистики с использованием технологии репликации данных между серверами СУБД»

Выполнил студент Кришталь Максим Олегович

(Ф.И.О.)

Руководитель работы асс. Нистюк Ольга Александровна

(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2023

**Содержание**

[Введение 4](#_Toc153300222)

[1 Постановка задачи 5](#_Toc153300223)

[1.2 Аналитический обзор аналогов 5](#_Toc153300224)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc153300225)

[1.3 Вывод 9](#_Toc153300226)

[2 Проектирование базы данных 10](#_Toc153300227)

[2.1 Обобщенная структура управлением приложения 10](#_Toc153300228)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 10](#_Toc153300229)

[2.3 Описание информационных объектов 11](#_Toc153300230)

[2.4 Вывод 11](#_Toc153300231)

[3 Разработка объектов базы данных 12](#_Toc153300232)

[3.1 Создание необходимых объектов 12](#_Toc153300233)

[3.2 Создание таблиц 12](#_Toc153300234)

[3.3 Создание ролей для разграничения 13](#_Toc153300235)

[3.4 Создание пакетов процедур для базы данных 13](#_Toc153300236)

[3.4.1 Выборка данных из таблиц 16](#_Toc153300237)

[3.4.2 Выборка данных по поисковому запросу 17](#_Toc153300238)

[3.4.3 Заполнение таблиц 100 000 строк 17](#_Toc153300239)

[3.4.4 Добавление данных в таблицы 18](#_Toc153300240)

[3.4.5 Удаление данных в таблицы 18](#_Toc153300241)

[3.4.6 Дополнительные функции 19](#_Toc153300242)

[3.5 Представления базы данных 19](#_Toc153300243)

[3.6 Индексы базы данных 20](#_Toc153300244)

[3.7 Триггеры базы данных 21](#_Toc153300245)

[3.8 Вывод 21](#_Toc153300246)

[4 Описание процедур импорта и экспорта 22](#_Toc153300247)

[5 Тестирование 23](#_Toc153300248)

[6 Описание технологии и ее применение в базе данных 24](#_Toc153300249)

[7 Краткое описание приложения для демонстрации 25](#_Toc153300250)

[7.1 Краткое описание приложения для демонстрации со стороны посетителя 25](#_Toc153300251)

[7.2 Краткое описание приложения для демонстрации со стороны пользователя 26](#_Toc153300252)

[7.3 Краткое описание приложения для демонстрации со стороны администратора 26](#_Toc153300253)

[7.4 Вывод 27](#_Toc153300254)

[8 Руководство пользователя 28](#_Toc153300255)

[8.1 Руководство пользователя 28](#_Toc153300256)

[8.2 Установка приложения 33](#_Toc153300257)

[8.3 Вывод 33](#_Toc153300258)

[Заключение 34](#_Toc153300259)

[Список используемых источников 35](#_Toc153300260)

[Приложение А 36](#_Toc153300261)

# Введение

Целью данной курсовой работы является разработка реляционной базы данных для сервиса футбольной статистики с использованием технологии репликации данных между серверами СУБД (системы управления базами данных). Футбольная статистика - важный аспект в мире спорта, и эффективное хранение и обработка данных о матчах, клубах, сборных и игроках являются ключевыми для удовлетворения потребностей пользователей.

База данных представляет собой организованное собрание данных, структурированных в виде таблиц, которые обеспечивают эффективное управление информацией. В работе используется реляционная модель, где данные представлены в виде таблиц с определенными атрибутами. Для управления базой данных выбрана система управления базами данных PostgreSQL, известная своей надежностью и производительностью. PostgreSQL предоставляет возможности для хранения, организации и обработки данных, а также поддерживает передовые технологии, такие как репликация данных.

Приложение разрабатывается для демонстрации функциональности базы данных и взаимодействия с ней. В качестве технологического стека используется фреймворк Spring Boot для серверной части на платформе Java. Это обеспечивает разработку масштабируемого, эффективного и современного веб-приложения, используя возможности Spring Framework для управления базой данных и обработки бизнес-логики.

Для гарантированной безопасности пользователей приложения в курсовой работе применяется метод шифрования паролей перед их сохранением в базу данных. Также для обеспечения функциональности приложения используются мультимедийные форматы данных при сохранении аудио и картинок.

Основные требования к приложению:

* реализация ролей администратора и пользователя;
* отправка, редактирование и удаление сообщений;
* добавление в друзья, удаление из друзей;
* возможность подписаться на другого или отписать от него;
* взаимодействие с базой данных при помощи хранимых процедур и функций.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

# 1 Постановка задачи

## 1.2 Аналитический обзор аналогов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

Первый аналог – Flashscore.

Flashscore предоставляет возможность отслеживания спортивных событий, получения актуальной статистики и участия в обсуждениях. Пользователям предоставляется широкий функционал, включая получение обновлений, просмотр статистики, фотографий и видео.

Интерфейс Flashscore разработан с учетом простоты использования и предоставляет интуитивно понятные элементы управления. Главный экран отображает список текущих спортивных событий, а также кнопки для быстрого доступа к важным разделам, таким как подписки, настройки и обзор статистики. Тёмная цветовая гамма и плоские иконки придают приложению современный и лаконичный вид.

Страница пользователя в Flashscore содержит изображение пользователя, а кнопки и элементы управления выделяются контрастными цветами на общем фоне, что обеспечивает наглядность и удобство в использовании.

Интерфейс приложения представлен на рисунке 1.1.

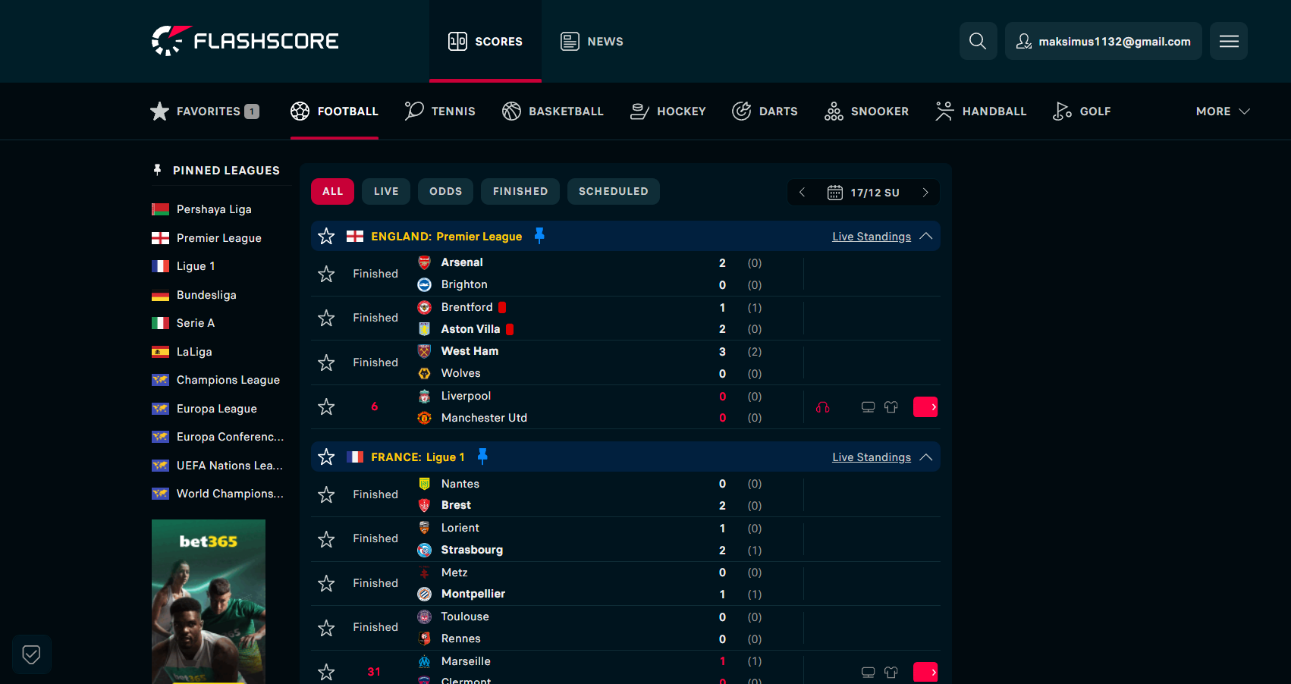


Рисунок 1.1 – Приложение «Flashscore»

Второй аналог – это приложение SofaScore.

SofaScore позволяет футбольным фанатам быть в курсе всех событий в мире футбола. В приложении вы можете получить доступ к живым результатам матчей, статистике, составам команд, новостям и многому другому. Возможности приложения включают: просмотр живых результатов и статистики, информацию о командах и игроках, уведомления и новости, графики и таблицы, интерактивные карты матчей.

Интерфейс SofaScore отличается легкостью в использовании, обеспечивая простой доступ к основной информации. Главный экран приложения отображает текущие матчи и результаты, а навигационные кнопки в верхней части экрана обеспечивают быстрый доступ к дополнительным функциям, таким как статистика, команды и новости. Дизайн приложения подчеркивает ясность и легкость восприятия, что делает SofaScore идеальным спутником для истинных ценителей футбола.

Интерфейс приложения представлен на рисунке 1.2.

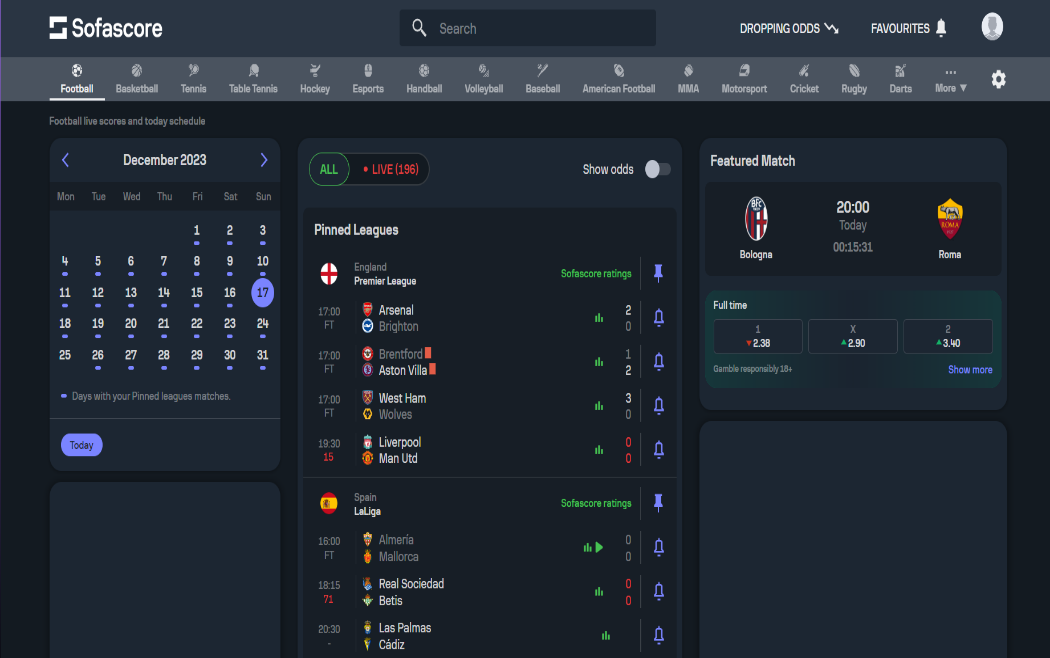


Рисунок 1.2 – Приложение «SofaScore»

Третий аналог – это onefootball.

Onefootball предоставляет вам всю необходимую информацию о футболе прямо у вас на устройстве. В этом приложении вы найдете слудующее. Новости и обзоры: Будьте в курсе последних новостей мира футбола с обширными новостными лентами и аналитическими обзорами. Живые результаты и статистика: Следите за матчами в режиме реального времени, получайте обновления о голах, ударах, угловых и других статистических данных. Команды и игроки: Получайте информацию о вашей любимой команде, составах, трансферах и статистике игроков. Пользовательские уведомления: Настройте уведомления о ваших любимых командах, чтобы быть в курсе всех важных моментов и результатов. Трансферные слухи: Следите за последними слухами и новостями о трансферах, переходах игроков между командами.

Интерфейс Onefootball обеспечивает удобство использования и интуитивную навигацию. Главный экран отображает текущие матчи и новости, а навигационные вкладки в верхней части экрана позволяют быстро переключаться между различными разделами, такими как новости, матчи, команды и другие. Дизайн приложения является современным, подчеркивая актуальность информации и простоту взаимодействия. Onefootball - идеальный спутник для тех, кто ценит футбол во всех его проявлениях.

Интерфейс приложения представлен на рисунке 1.3.

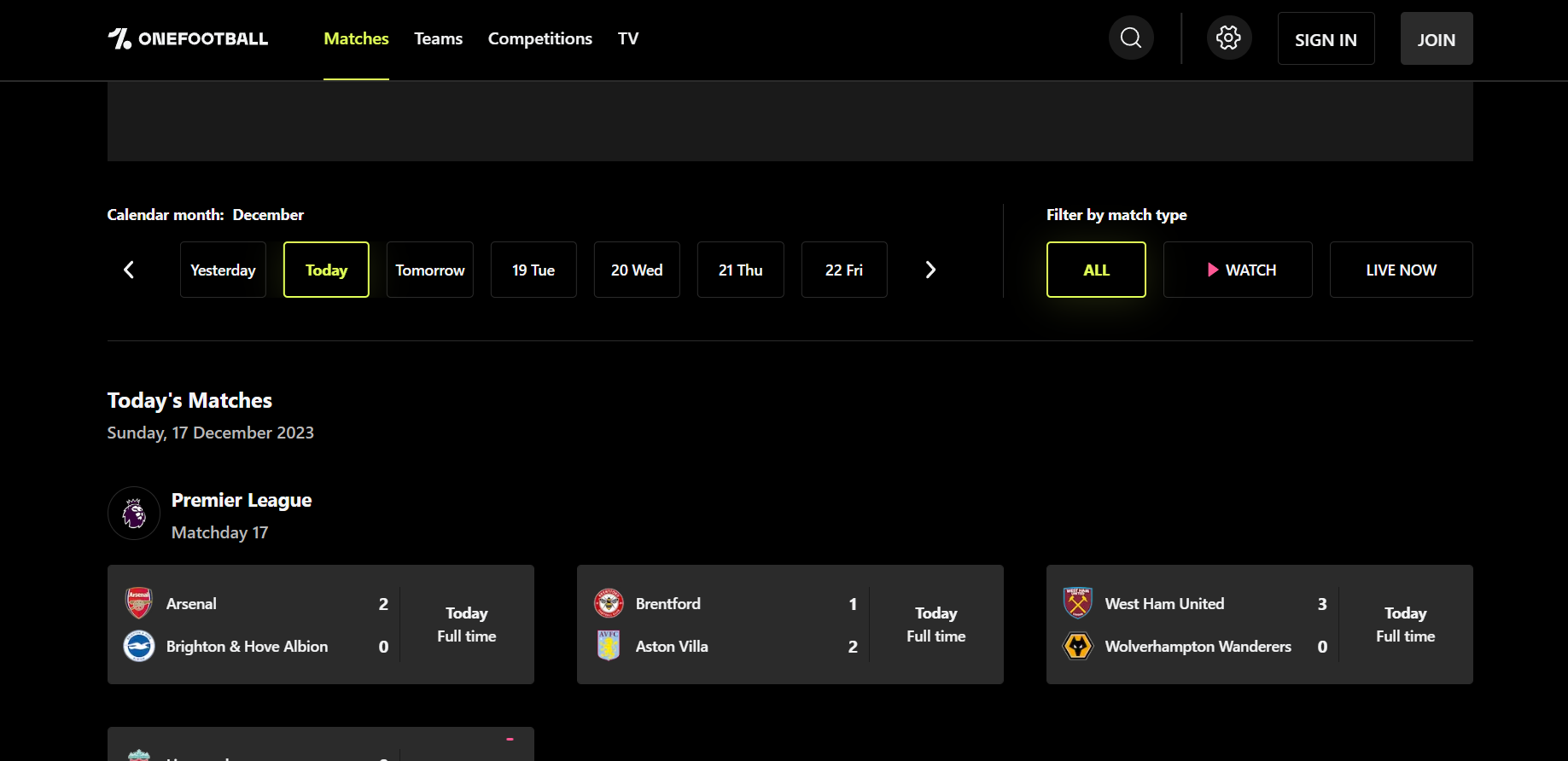


Рисунок 1.3 – Приложение «Onefootball»

FlashScore - это приложение, обеспечивающее футбольных болельщиков моментальной статистикой и обновлениями в режиме реального времени, охватывая не только футбол, но и различные виды спорта. Интуитивный интерфейс предоставляет доступ к таблицам турниров, графикам и уведомлениям. Onefootball, с фокусом на футбол, предоставляет полноту информации с новостными лентами, статистикой, трансферными слухами и настраиваемыми уведомлениями. Интерфейс Onefootball привлекателен, обеспечивая ясность и простоту в использовании. SofaScore, подходящий для всех видов спорта, предлагает подробные данные о матчах, командах и игроках в режиме реального времени, с простым и удобным интерфейсом. Каждое из этих приложений имеет свои особенности, обеспечивая болельщикам полный спектр информации и удобство использования.

## 1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования базы данных определяют, как база данных должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Кроме того, функциональные требования могут определять интеграцию базы данных с другими системами и программами. Например, для сервиса футбольной статистики, как будут хранится матчи, как будут храниться команды и игроки, как будут храниться избранные команды и игроки пользователей.

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

Роли пользователя – это набор прав, которые пользователь может получить в системе. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* ROLE\_UNAUTHORIZED;
* ROLE\_USER;
* ROLE\_ADMIN.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.4.

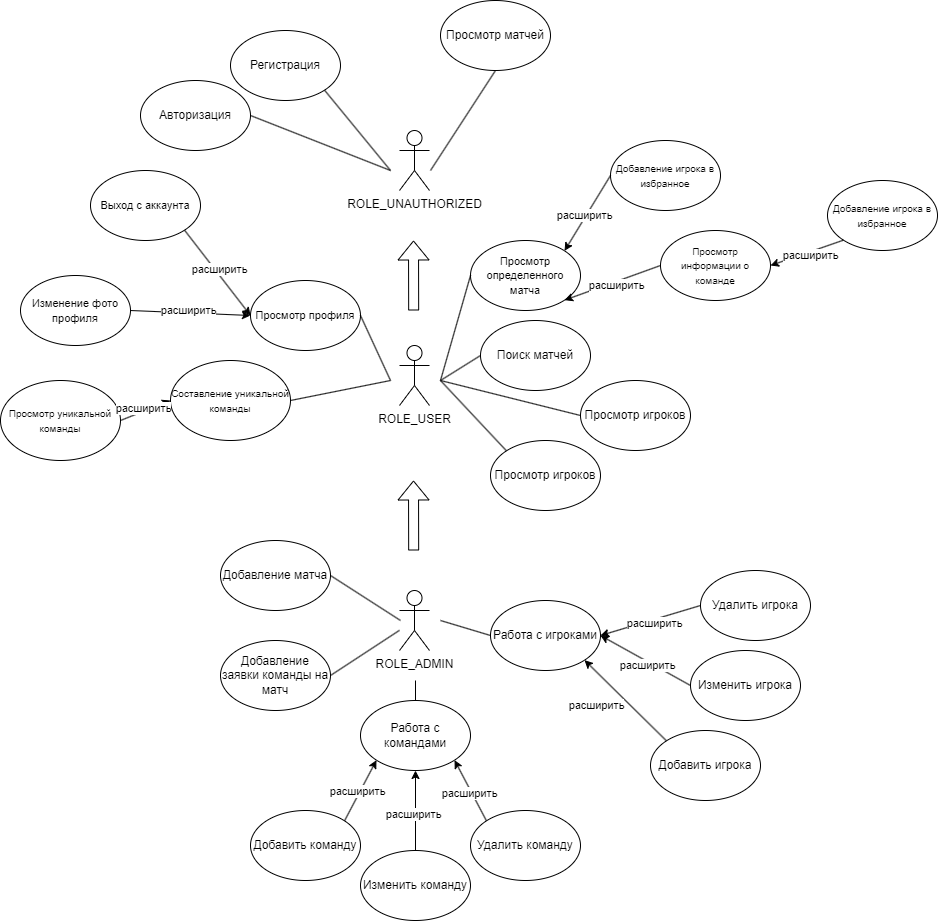


Рисунок 1.4 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является неавторизованным. Ему будут доступны лишь три функции: авторизация, регистрация и просмотр матчей. После регистрации пользователь становится пользователем ROLE\_USER.

Роль ROLE\_USER дает уже функцию просмотра определенного матча, добавление игроков и команд в избранное, изменение профиля, составление уникальной команды, просмотр игроков, просмотр команд, просмотр информации о команде.

Роль ROLE\_ADMIN может добавлять заявки команд на матч, создание матчей, добавлять команды, изменять команды, удалять команды, добавлять игроков, изменять игроков, удалять игроков.

Приложение обладает тремя ролями, на которые разложены все основные функции для корректного использования.

## 1.3 Вывод

Итого, был проведен аналитический обзор аналогов сервисов, которые уже существуют на рынке. Этот обзор позволил определить основные характеристики и функциональные возможности, которые необходимо предусмотреть в разрабатываемой системе. Также были определены функциональные требования базы данных, а также роли пользователей и варианты использования системы в зависимости от этих ролей. Была разработана UML-диаграмма, на которой отображены основные функции, которые доступны для каждой из ролей пользователей.

# 2 Проектирование базы данных

## 2.1 Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения для общения должны входить функции для удобного поиска людей, функции добавления в друзья, различные виды сообщение и другие подобные функции.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.1.

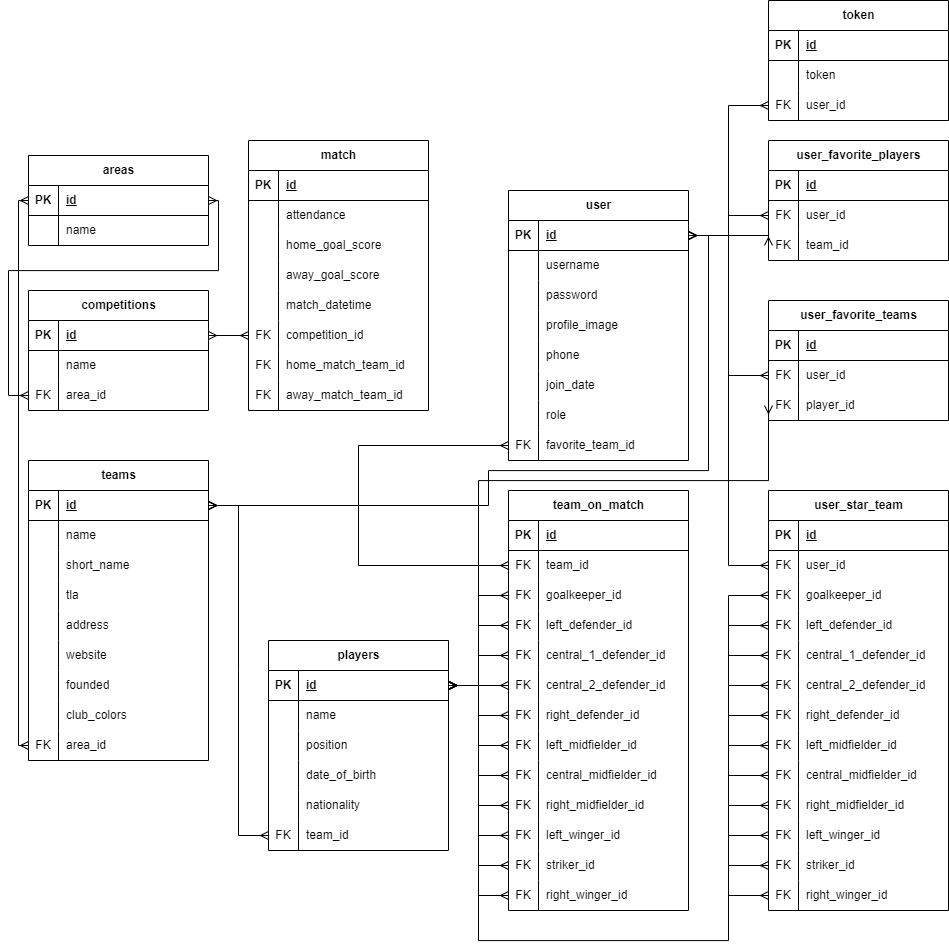


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-одному", "один-ко-многим", "многие-ко-многим". Например, таблица users связана с таблицами token, user\_favorite\_players, user\_favorite\_teams, user\_star\_team через внешние ключи.

## 2.3 Описание информационных объектов

Для реализации базы данных было разработано 11 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: areas, competitions, match, players, team\_on\_match, teams, token, user, user\_favorite\_players, user\_favorite\_teams и user\_star\_team. Ниже будет описание про каждую из них более подробно.

Таблица areas отвечает за хранение областей местанохаждений. Здесь следующие поля: id integer primary key, name character varying(32).

Таблица competitions отвечает за хранение всех соревнований и турниров. Здесь следующие поля: id integer primary key, area\_id integer FK(areas), name character varying(64).

Таблица teams отвечает за хранение команд. Здесь поля: id integer primary key, area\_id integer FK(areas), name character varying(64), short\_name character varying(32), tla character varying(8), address character varying(256), website character varying(128), founded integer, club\_colors character varying(64).

Таблица players отвечает за хранение игроков. Здесь поля: id integer primary key, name character varying(64), position character varying(32), date\_of\_birth date, nationality character(64), team\_id integer FK(teams).

Таблица team\_on\_match отвечает за хранение заявок команд на матч. Здесь поля: id integer primary key, team\_id integer FK(teams), goalkeeper\_id integer FK(players), left\_defender\_id integer FK(players), central\_1\_defender\_id integer FK(players), central\_2\_defender\_id integer FK(players), right\_defender\_id integer FK(players), left\_midfielder\_id integer FK(players), central\_midfielder\_id integer FK(players), right\_midfielder\_id integer FK(players), left\_winger\_id integer FK(players), striker\_id integer FK(players), right\_winger\_id integer FK(players),

Таблица match отвечает за хранение матчей. Здесь следующие поля: id integer primary key, competition\_id integer FK(competitions), attendance integer, home\_match\_team\_id integer FK(team\_on\_match), away\_match\_team\_id integer FK(team\_on\_match), home\_goal\_score smallint, away\_goal\_score smallint, match\_datetime timestamp without time zone.

Таблица user отвечает за хранение пользователей. Здесь поля: id integer primary key, username character varying(32), password character varying(256), favorite\_team\_id integer FK(teams), profile\_image bytea, phone character varying(16), join\_date date, role character varying(32).

Таблица token отвечает за хранение специальных токенов для использования в сессиях пользователей. Здесь три поля: id integer primary key, user\_id integer FK(user), token character varying(256).

Таблица user\_favorite\_players отвечает за хранение избранных игроков пользователей. Здесь три поля: id integer primary key, user\_id integer FK(user), player\_id integer FK(players).

Таблица user\_favorite\_teams отвечает за хранение избранных команд пользователей. Здесь три поля: id integer primary key, user\_id integer FK(user), team\_id integer FK(teams).

Таблица user\_star\_team отвечает за хранения пользовательских уникальных команд. Здесь два поля: id integer primary key, user\_id integer FK(user), goalkeeper\_id integer FK(players), left\_defender\_id integer FK(players), central\_1\_defender\_id integer FK(players), central\_2\_defender\_id integer FK(players), right\_defender\_id integer FK(players), left\_midfielder\_id integer FK(players), central\_midfielder\_id integer FK(players), right\_midfielder\_id integer FK(players), left\_winger\_id integer FK(players), striker\_id integer FK(players), right\_winger\_id integer FK(players).

## 2.4 Вывод

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой.

Описание информационных объектов является важной частью архитектуры проекта, так как это помогает понять, какие данные будут использоваться в приложении, и как они будут храниться и обрабатываться.

# 3 Разработка объектов базы данных

## 3.1 Создание необходимых объектов

Для создания web-приложения футбольного сервиса должны быть реализованы таблицы упомянутые в предыдущем разделе. Для более удобной работы с ними должны быть созданы индексы, функции, процедуры и представления. Скрипты создания таблиц представлены в приложении А.

## 3.2 Создание таблиц

В данном разделе мы создадим таблицы для нашей базы данных. Но перед тем, как приступить к созданию таблиц, нам нужно создать табличное пространство.

Табличное пространство - это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств будет представлен на листинге 3.1.

|  |
| --- |
| CREATE TABLESPACE goalscoreservice\_default  OWNER postgres  LOCATION D:\BSTU\Postgre\data\tbs; |

Листинг 3.1 – Cкрипт для создания табличных пространств

Для базы данных создадим одиннадцать основных таблиц: areas, competitions, match, players, team\_on\_match, teams, token, user, user\_favorite\_players, user\_favorite\_teams и user\_star\_team..

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. Одна из основных связей - это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице. Например, у каждого пользователя может быть множество фотографий, которые он добавил. Для этого мы добавим в таблицу players внешний ключ (FOREIGN KEY) на таблицу teams, который будет указывать на идентификатор пользователя.

Другой тип связи - это связь "многие ко многим" (many-to-many), которая определяет отношение между множеством записей в одной таблице и множеством записей в другой таблице. Например, есть две таблицы players и teams, у множества команд может быть множество игроков так же, как и у множества игроков может быть множество команд. Скрипт создание таблицы players будет представлен на листинге 3.2.

|  |
| --- |
| CREATE TABLE IF NOT EXISTS public.players  (  id integer NOT NULL,  name character varying(64) COLLATE pg\_catalog."default" NOT NULL,  "position" character varying(32) COLLATE pg\_catalog."default" NOT NULL,  date\_of\_birth date NOT NULL,  nationality character(64) COLLATE pg\_catalog."default" NOT NULL,  team\_id integer NOT NULL,  CONSTRAINT players\_pkey PRIMARY KEY (id),  CONSTRAINT team\_id FOREIGN KEY (team\_id)  REFERENCES public.teams (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE RESTRICT  ) TABLESPACE goalscoreservice\_default; |

Листинг 3.2 – Cкрипт создание таблицы User

Таким образом, было описано создание табличного пространства для базы данных, а также таблиц, которые будут храниться в этих пространствах. Было создано табличное пространство goalscoreservice\_default.

## 3.3 Создание ролей для разграничения

В этом разделе создаются роли для ограничения доступа к базе данных. Создание ролей позволяет установить границы доступа к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

## 3.4 Создание пакетов процедур для базы данных

Для управления данными через приложение пользователи и администраторы используют хранимые процедуры и функции. Хранимая процедура представляет собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Функция также представляет собой набор SQL-инструкций, но возвращает значение, которое может быть использовано внутри другой инструкции SQL.

Написанные в ходе разработки курсового проекта процедуры и функции можно разбить на несколько категорий:

1. Выборка данных из таблиц.
2. Выборка данных по поисковому запросу.
3. Заполнение таблиц 100 000 строк.
4. Добавление данных в таблицы.
5. Удаление данных из таблиц.
6. Изменение данных в таблицах.
7. Дополнительные функции.

Отличие функций от процедур состоит в том, что функции возвращают значение, которое может быть использовано в других SQL-запросах, а процедуры не возвращают значение. Кроме того, функции могут быть использованы в выражениях SQL, например, для вычисления значения поля в запросе SELECT.

В зависимости от того, какую задачу необходимо выполнить, следует использовать хранимую процедуру или функцию. Хранимые процедуры могут использоваться для выполнения сложных операций над данными, таких как массовые изменения в таблицах, а также для оптимизации производительности приложения. Функции же наиболее полезны в случаях, когда требуется выполнить вычисление на основе данных в базе данных, например, для подсчета статистики или фильтрации данных.

Будут созданы четыре роли для разграничения доступа к базе данных: ROLE\_UNAUTHORIZED, ROLE\_USER, ROLE\_ADMIN. Описание этих ролей будет идти далее.

Роль ROLE\_UNAUTHORIZED имеет доступ лишь к функциям, которые позволяют добавить нового пользователя и просмотра матчей но без подробностей. То есть ей доступна, вставка в таблицы users, выборка из таблиц users, matches, также это роли доступно выполнение трёх функций get\_searched\_matches\_info, register\_user, is\_token\_with\_user\_id, а еще и процедуры change\_token, insert\_new\_user. Реализация данной роли показана на листинге 3.3.

|  |
| --- |
| create role ROLE\_UNAUTHORIZED;  grant connect on database goalscore to ROLE\_UNAUTHORIZED;  grant SELECT on public.user to ROLE\_UNAUTHORIZED;  grant INSERT on public.user to ROLE\_UNAUTHORIZED;  grant INSERT on token to ROLE\_UNAUTHORIZED;  grant select on match to ROLE\_UNAUTHORIZED;  grant execute on FUNCTION get\_searched\_matches\_info to ROLE\_UNAUTHORIZED;  grant execute on procedure change\_token to ROLE\_UNAUTHORIZED;  grant execute on procedure insert\_new\_user to ROLE\_UNAUTHORIZED;  grant execute on FUNCTION register\_user to ROLE\_UNAUTHORIZED;  grant execute on FUNCTION is\_token\_with\_user\_id to ROLE\_UNAUTHORIZED; |

Листинг 3.3 – Реализация роли ROLE\_UNAUTHORIZED

Роль ROLE\_USER имеет право на выполнение всех операций, связанных с объектами, хранящимися в goalscoreservice\_default, что позволяет этой роли пользоваться всем функционалом данной БД. Выданные привилегии роли ROLE\_USER можно увидеть на листинге 3.4.

|  |
| --- |
| create role ROLE\_USER;  grant ROLE\_UNAUTHORIZED to ROLE\_USER;  grant insert on user\_favorite\_players to ROLE\_USER;  grant update on user\_favorite\_players to ROLE\_USER;  grant delete on user\_favorite\_players to ROLE\_USER;  grant insert on user\_favorite\_teams to ROLE\_USER;  grant update on user\_favorite\_teams to ROLE\_USER;  grant delete on user\_favorite\_teams to ROLE\_USER;  grant insert on user\_star\_team to ROLE\_USER;  grant update on user\_star\_team to ROLE\_USER;  grant delete on user\_star\_team to ROLE\_USER;  grant select on teams to ROLE\_USER;  grant select on players to ROLE\_USER; |

Листинг 3.4 – Привилегии, выданные роли ROLE\_USER

Роль ROLE\_ADMIN имеет все те же права, что и предыдущая роль, но плюс ко всему ей доступна работа с определенными объектами. Данная работа включает возможность создания, изменения, удаления игрока или команды, добавления заявки команды на матч, добавления матча и тому подобные. Выданные привилегии роли ROLE\_ADMIN можно увидеть на листинге 3.5.

|  |
| --- |
| create role ROLE\_ADMIN;  grant ROLE\_USER to ROLE\_ADMIN;  grant insert on match to ROLE\_ADMIN;  grant update on match to ROLE\_ADMIN;  grant delete on match to ROLE\_ADMIN;  grant insert on players to ROLE\_ADMIN;  grant update on players to ROLE\_ADMIN;  grant delete on players to ROLE\_ADMIN;  grant insert on team\_on\_match to ROLE\_ADMIN;  grant update on team\_on\_match to ROLE\_ADMIN;  grant delete on team\_on\_match to ROLE\_ADMIN;  grant insert on teams to ROLE\_ADMIN;  grant update on teams to ROLE\_ADMIN;  grant delete on teams to ROLE\_ADMIN;  grant insert on areas to ROLE\_ADMIN;  grant delete on areas to ROLE\_ADMIN;  grant insert on competitions to ROLE\_ADMIN;  grant delete on competitions to ROLE\_ADMIN; |

Листинг 3.5 – Привилегии, выданные роли ROLE\_ADMIN

Благодаря созданным ролям можно разграничить доступ к действиям с базой данных.

### 3.4.1 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: public.get\_all\_matches, public.get\_all\_teams, public.get\_area\_by\_id, public.get\_favorite\_players\_data, public.get\_favorite\_teams\_by\_user\_id, public.get\_match\_by\_id, public.get\_match\_info\_by\_id, public.get\_matches\_info, public.get\_players\_range, public.get\_starting\_players, public.get\_team\_by\_id, public.get\_team\_on\_match\_by\_id, public.get\_token\_id\_by\_user\_id, public.get\_total\_player\_pages, public.get\_user\_by\_id, public.get\_user\_id\_by\_token, public.get\_user\_role\_by\_token. Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

Функция get\_all\_matches предоставляет информацию обо всех матчах, включая данные о командах, времени проведения и результатах. Результат представлен в удобном формате для дальнейшего анализа.

Функция get\_all\_teams возвращает информацию обо всех командах, зарегистрированных в приложении. Результат включает основные данные о каждой команде, такие как идентификатор, название и другие характеристики.

Функция get\_area\_by\_id предоставляет информацию о конкретной области или районе на основе переданного идентификатора. Возвращает основные характеристики области, такие как название и другие данные.GetFriends - функции для выборки друзей.

Функция get\_favorite\_players\_data возвращает информацию о любимых игроках пользователя, включая их идентификаторы, имена, команды и другие характеристики. Позволяет получить обзор предпочтений пользователя в отношении футбольных игроков.

Функция get\_favorite\_teams\_by\_user\_id предоставляет информацию о командах, которые являются избранными для определенного пользователя. Возвращает данные о командах, включая их идентификаторы и названия.

Функция get\_match\_by\_id возвращает информацию о конкретном матче на основе переданного идентификатора. Результат включает основные характеристики матча, такие как команды, время проведения и результаты.

Функция get\_match\_info\_by\_id предоставляет подробную информацию о конкретном матче, включая составы команд, статистику и другие данные. Результат возвращается в структурированной форме.

Функция get\_matches\_info предоставляет общую информацию о всех матчах, включая основные характеристики каждого матча. Результат представлен в удобной форме для анализа.

Функция get\_players\_range возвращает информацию об игроках, находящихся в указанном диапазоне. Результат включает данные о каждом игроке, такие как идентификатор, имя, команда и другие характеристики.

Функция get\_starting\_players возвращает информацию о стартовых игроках в конкретном матче. Результат включает данные о каждом игроке, включенном в стартовый состав, такие как идентификатор, имя, команда и другие характеристики.

Функция get\_team\_by\_id возвращает информацию о конкретной команде на основе переданного идентификатора. Результат включает основные характеристики команды, такие как название и другие данные.

Функция get\_team\_on\_match\_by\_id предоставляет информацию о команде, участвующей в конкретном матче, на основе переданного идентификатора матча. Результат включает данные о команде, её составе и результате в матче.

Функция get\_token\_id\_by\_user\_id возвращает идентификатор токена для указанного пользователя. Результат включает информацию о токене, связанном с пользователем.

Функция get\_total\_player\_pages возвращает общее количество страниц с информацией об игроках. Результат полезен при постраничном отображении данных.

Функция get\_user\_by\_id предоставляет информацию о пользователе на основе его идентификатора. Результат включает основные характеристики пользователя, такие как логин, имя, фамилия и другие данные.

Функция get\_user\_id\_by\_token возвращает идентификатор пользователя на основе переданного токена. Результат полезен для идентификации пользователя при авторизации.

Функция get\_user\_role\_by\_token предоставляет информацию о роли пользователя на основе переданного токена. Результат включает роль пользователя в системе.

На листинге 3.6 будет функция get\_starting\_playersg, которая предназначена для получения игроков стартового состава в заявке на матч.

|  |
| --- |
| CREATE OR REPLACE FUNCTION public.get\_starting\_players(  team\_on\_match\_id integer)  RETURNS TABLE(player\_id integer, player\_name character varying, player\_position text)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT DISTINCT ON (p.id)  p.id AS player\_id,  p.name AS player\_name,  CASE  WHEN tm.goalkeeper\_id = p.id THEN 'goalkeeper'  WHEN tm.left\_defender\_id = p.id THEN 'left defender'  WHEN tm.central\_1\_defender\_id = p.id THEN 'central 1 defender'  WHEN tm.central\_2\_defender\_id = p.id THEN 'central 2 defender'  WHEN tm.right\_defender\_id = p.id THEN 'right defender'  WHEN tm.left\_midfielder\_id = p.id THEN 'left midfielder'  WHEN tm.central\_midfielder\_id = p.id THEN 'central midfielder'  WHEN tm.right\_midfielder\_id = p.id THEN 'right midfielder'  WHEN tm.left\_winger\_id = p.id THEN 'left winger'  WHEN tm.striker\_id = p.id THEN 'striker'  WHEN tm.right\_winger\_id = p.id THEN 'right winger'  END AS player\_position  FROM  team\_on\_match tm  INNER JOIN players p ON tm.team\_id = p.team\_id  WHERE  tm.id = team\_on\_match\_id;  RETURN;  END;  $BODY$; |

Листинг 3.6 – Функция get\_starting\_players

Все остальные функции и процедуры будут аналогичны, также предназначены для выборки данных из различных таблиц базы данных.

### 3.4.2 Выборка данных по поисковому запросу

Для поиска была разработана функция get\_searched\_matches\_info.

Функция get\_searched\_matches\_info предоставляет информацию о матчах, соответствующих заданным критериям поиска. Результат возвращается в удобной форме для обзора и анализа.

Пример функции представлен на листинге 3.7.

|  |
| --- |
| CREATE OR REPLACE FUNCTION public.get\_searched\_matches\_info(  team\_search character varying)  RETURNS TABLE(match\_id integer, competition\_name character varying, area\_name character varying, match\_datetime timestamp without time zone, home\_team\_name character varying, home\_team\_id integer, away\_team\_name character varying, away\_team\_id integer, home\_goal smallint, away\_goal smallint)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT  m.id AS match\_id,  c.name AS competition\_name,  a.name AS area\_name,  m.match\_datetime,  home\_team.name AS home\_team\_name,  home\_team.id AS home\_team\_id,  away\_team.name AS away\_team\_name,  away\_team.id AS away\_team\_id,  m.home\_goal\_score as home\_goal,  m.away\_goal\_score as away\_goal  FROM  public.match m  JOIN public.competitions c ON m.competition\_id = c.id  JOIN public.areas a ON c.area\_id = a.id  JOIN public.team\_on\_match tom\_home ON m.home\_match\_team\_id = tom\_home.id  JOIN public.teams home\_team ON tom\_home.team\_id = home\_team.id  JOIN public.team\_on\_match tom\_away ON m.away\_match\_team\_id = tom\_away.id  JOIN public.teams away\_team ON tom\_away.team\_id = away\_team.id  WHERE  team\_search IS NULL  OR home\_team.name ILIKE '%' || team\_search || '%'  OR away\_team.name ILIKE '%' || team\_search || '%';  END;  $BODY$; |

Листинг 3.7 – Функция get\_searched\_matches\_info

Таким образом, в данном разделе были представлены примеры функций, которые позволяют искать матчи.

### 3.4.3 Заполнение таблиц 100 000 строк

Для заполнения таблицы areas была разработана процедура populate\_areas, которая вставляет 100000 строк в таблицу. Данная процедура использует цикл for, который с каждой итерацией увеличивает значение переменной i на 1 и делает новую запись в таблицу areas. Процедура представлена на листинге 3.8.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.populate\_areas(  )  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  start\_id INTEGER := 3000;  end\_id INTEGER := 103000;  BEGIN  FOR i IN start\_id..end\_id LOOP  INSERT INTO public.areas (id, name)  VALUES (i, 'Area ' || i);  END LOOP;  END;  $BODY$;  ALTER PROCEDURE public.populate\_areas()  OWNER TO postgres; |

Листинг 3.8 – Процедура заполнения таблицы areas

Данная функция заполняет таблицу в течении несколько секунд. Производительность базы данных будет проверена дальше.

### 3.4.4 Добавление данных в таблицы

Были разработаны множество функций для добавления данных в таблицы. insert\_new\_user и insert\_new\_token для реализации регистрации и авторизации пользователя. Функция insert\_player для добавления нового игрока. insert\_team для добавления команды. insert\_match функция для добавления записи в таблицу match. Функция insert\_new\_team\_on\_match для создания новой записи в таблицу team\_on\_match. Функция insert\_user\_favorite\_player для добавления нового избранного игрока пользователя. insert\_user\_favorite\_team для добавления команды в избранное пользователя. insert\_user\_star\_team функция для добавления уникальной команды пользователя.

Реализации этих функций и процедур будет в приложении А.

### 3.4.5 Удаление данных в таблицы

Для удаления данных из базы данных были созданы процедуры:delete\_player для удаления игрока, delete\_team для удаления команды, remove\_favorite\_player для удаления избранного игрока, remove\_favorite\_team для удаления команды из избранного пользователя.

На листинге 3.9 будет процедура delete\_player.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.delete\_player(  IN p\_player\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF NOT EXISTS (SELECT 1 FROM public.players WHERE id = p\_player\_id) THEN  RAISE EXCEPTION 'Player with id % not found', p\_player\_id;  END IF;  DELETE FROM public.players WHERE id = p\_player\_id;  END;  $BODY$; |

Листинг 3.9 – Процедура для удаления игрока

Все остальные процедуры будут аналогичны, также предназначены для удаления соответствующих данных из основных таблиц базы данных.

### 3.4.6 Дополнительные функции

Дополнительные функции в базе данных могут быть полезны для решения различных задач, которые не решаются стандартными запросами.

Функция is\_token\_with\_user\_id для проверки существует ли уже созданный токен для пользователя. Принимает id пользователя. Функция представлена на листинге 3.10.

|  |
| --- |
| CREATE OR REPLACE FUNCTION public.is\_token\_with\_user\_id(  integer)  RETURNS boolean  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  BEGIN  IF ((SELECT COUNT(\*) FROM public.token WHERE user\_id=$1) = 1) THEN  RETURN TRUE;  END IF;  RETURN FALSE;  END;  $BODY$; |

Листинг 3.10 – Функция is\_token\_with\_user\_id

Функция check\_user\_star\_team\_exists для проверки существования уже созданной уникальной команды пользователя. Принимает id пользователя. Функция представлена на листинге 3.11.

|  |
| --- |
| CREATE OR REPLACE FUNCTION public.check\_user\_star\_team\_exists(  p\_user\_id integer)  RETURNS boolean  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  user\_exists BOOLEAN;  BEGIN  SELECT TRUE INTO user\_exists  FROM public.user\_star\_team  WHERE user\_id = p\_user\_id  LIMIT 1;  RETURN user\_exists;  END;  $BODY$; |

Листинг 3.11 – Функция check\_user\_star\_team\_exists

В целом, эти функции могут быть полезны для проверки наличия данных, для последующий действий.

## 3.5 Представления базы данных

Представление в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно, при этом не изменяя структуру этих таблиц.

В данном проекте было создано представление:

* matches\_info\_view, которое позволяет вернуть требуемую информацию о матчах такую как id матча, название соревнования, название местонахождения, дата проведения матча, название домашней команды, заявку домашней команды, название гостевой команды, заявку гостевой команды, счёт матча;

Представление matches\_info\_view представлено на листинге 3.12.

|  |
| --- |
| CREATE OR REPLACE VIEW public.matches\_info\_view AS  SELECT  m.id AS match\_id,  c.name AS competition\_name,  a.name AS area\_name,  m.match\_datetime,  home\_team.name AS home\_team\_name,  home\_team.id AS home\_team\_id,  away\_team.name AS away\_team\_name,  away\_team.id AS away\_team\_id,  m.home\_goal\_score as home\_goal,  m.away\_goal\_score as away\_goal  FROM  public.match m  JOIN public.competitions c ON m.competition\_id = c.id  JOIN public.areas a ON c.area\_id = a.id  JOIN public.team\_on\_match tom\_home ON m.home\_match\_team\_id = tom\_home.id  JOIN public.teams home\_team ON tom\_home.team\_id = home\_team.id  JOIN public.team\_on\_match tom\_away ON m.away\_match\_team\_id = tom\_away.id  JOIN public.teams away\_team ON tom\_away.team\_id = away\_team.id; |

Листинг 3.12 – Представление matches\_info\_view

Данные представления помогаю работать с таблицами более гибко.

## 3.6 Индексы базы данных

Индекс – объект базы данных, который используется для ускорения поиска данных. В случае большого количества строк в таблице, последовательный поиск данных может занимать много времени. Индекс формируется на основе значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет быстро искать строки, удовлетворяющие заданному критерию поиска. Использование индексов ускоряет работу с базой данных, потому что они имеют оптимизированную структуру для поиска, например, сбалансированное дерево.

Для быстрого поиска матчей и для быстрого команд сообщений были разработаны 4 индекса. Эти индексы предназначены для улучшения производительности запросов, вовлекающих указанные столбцы, обеспечивая более быстрый доступ и сортировку данных. Выбор между индексацией B-tree и хеш-индексацией зависит от типов запросов.

Сами индексы представлены на листинге 3.13.

|  |
| --- |
| CREATE INDEX areas\_id\_btree\_index  ON public.areas USING btree (id)  TABLESPACE goalscoreservice\_default;  CREATE INDEX areas\_name\_hash\_index  ON public.areas USING hash (name)  TABLESPACE goalscoreservice\_default;  CREATE INDEX idx\_teams\_name  ON public.teams (name)  TABLESPACE goalscoreservice\_default;  CREATE INDEX idx\_teams\_id  ON public.teams (id)  TABLESPACE goalscoreservice\_default; |

Листинг 3.13 – Индексы базы данных для таблицы Messages

Кроме этих индексов, есть и другие. Они позволяют более быстро искать пользователей, находить фотографии для определенного пользователя и многое другое. Эти индексы представлены в приложении А.

## 3.7 Триггеры базы данных

Триггер базы данных – это объект базы данных, который выполняет некоторое действие автоматически при определенных событиях в таблице или представлении базы данных. Триггер может быть запрограммирован на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для обеспечения целостности данных и контроля доступа к данным, а также для автоматической обработки данных при выполнении определенных операций в таблице.

Триггер check\_teams\_different\_trigger создан для того, чтобы проверить корректность id заявок команд. Данный триггер представлен в листинге 3.14.

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_teams\_different()  RETURNS TRIGGER AS $$  DECLARE  home\_team\_id integer;  away\_team\_id integer;  BEGIN  SELECT team\_id INTO home\_team\_id  FROM public.team\_on\_match  WHERE id = NEW.home\_match\_team\_id;  SELECT team\_id INTO away\_team\_id  FROM public.team\_on\_match  WHERE id = NEW.away\_match\_team\_id;  IF home\_team\_id = away\_team\_id THEN  RAISE EXCEPTION 'Home team and away team must be different';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_teams\_different\_trigger  BEFORE INSERT OR UPDATE ON public.match  FOR EACH ROW  EXECUTE FUNCTION check\_teams\_different(); |

Листинг 3.14 – Скрипт триггера check\_teams\_different\_trigger

Этот триггер создан для процесса добавления матча.

## 3.8 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для мессенджера, где мы увидели основные объект данной базы данных.

# 4 Описание процедур импорта и экспорта

База данных имеет возможность импортировать данные для таблиц areas, competitions, teams, players в формат JSON и экспортировать данные из JSON в таблицу users. Это может быть полезно в случае необходимости переноса данных на другой сервер или резервного копирования данных.

Для импорта в areas была разработана функция import\_from\_json\_areas. Остальные функции выполнены аналогично. Данная функция представлена в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.import\_from\_json\_areas(  IN path\_to\_file text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  json\_string jsonb;    err\_code text;  msg\_text text;  exc\_context text;  msg\_detail text;  exc\_hint text;  BEGIN  json\_string := pg\_read\_file(path\_to\_file);  insert into public.areas  select (obj->>'id')::integer as id, (obj->>'name') as name  from jsonb\_array\_elements(json\_string) as obj;    RAISE NOTICE 'Success';  RAISE NOTICE 'Value of path\_to\_file: %', path\_to\_file;  RAISE NOTICE 'Value of json\_string: %', json\_string;    EXCEPTION WHEN OTHERS  THEN  GET STACKED DIAGNOSTICS  err\_code = RETURNED\_SQLSTATE,  msg\_text = MESSAGE\_TEXT,  exc\_context = PG\_CONTEXT,  msg\_detail = PG\_EXCEPTION\_DETAIL,  exc\_hint = PG\_EXCEPTION\_HINT;  RAISE NOTICE  'ERROR CODE: %  MESSAGE TEXT: %  CONTEXT: %  DETAIL: %  HINT: %',  err\_code, msg\_text, exc\_context, msg\_detail, exc\_hint;  END;  $BODY$;  ALTER PROCEDURE public.import\_from\_json\_areas(text)  OWNER TO postgres; |

Листинг 4.1 – Функция import\_from\_json\_areas

Для экспорта из формата JSON была разработана функция export\_users\_to\_json. Данная функция представлена в листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.export\_users\_to\_json(  )  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  user\_data JSON;  BEGIN  SELECT array\_to\_json(array\_agg(row\_to\_json(users.\*)))  INTO user\_data  FROM public."user" users;  RAISE NOTICE 'Exported user data to JSON: %', user\_data;  END;  $BODY$;  ALTER PROCEDURE public.export\_users\_to\_json()  OWNER TO postgres; |

Листинг 4.2 – Функция export\_users\_to\_json

Данные функции могут пригодится в будущем для создания архивов сообщений, для резервного копирования данных из БД и во многом другом.

# 5 Тестирование

Тестирование производительности является важным этапом разработки, поскольку позволяет определить, насколько хорошо база данных может обрабатывать запросы и как быстро она может возвращать результаты.

Для тестирования производительности базы данных была выбрана таблица areas, содержащая больше всего данных. Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| EXPLAIN ANALYZE select \* from public.areas  where name = 'Area 80088'; |

Листинг 5.1 – Запрос к таблице Messages

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составило 9.532 мс, а время планирования – 0.249 мс. Результаты запроса будут представлены на рисунке 5.1.

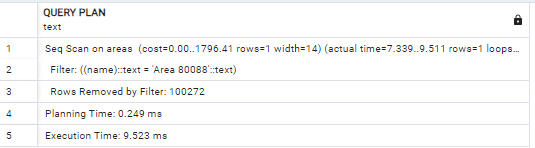


Рисунок 5.1 – Результат выполнения запроса

Для ускорения данного процесса можно создать два индекса. Первый позволит более быстро искать области по названию места, другой индекс ускорит работу с поиском по идентификатору. Результат будет представлен на рисунке 5.2.

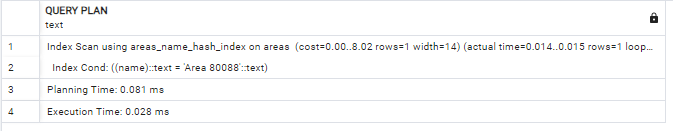


Рисунок 5.2 – Результат выполнения запроса

После создания индексов видно, что чаты стали генерировать намного быстрее. Время планирования заняло 0.081 мс, а время выполнения 0.028 мс, что более чем в 300 раз быстрее.

Результаты тестирования говорят о том, что индексы помогают быстрее сформировать запросы данных из таблицы areas.

# 6 Описание технологии и ее применение в базе данных

В данной базе данных используется репликация для обеспечения надежности и доступности данных. Репликация - это процесс создания и поддержания точных копий данных, которые распределены на различных узлах или серверах. Это позволяет достичь повышенной отказоустойчивости, улучшить производительность и обеспечить более быстрый доступ к данным.

В случае репликации данных, изменения, внесенные в одной копии данных, автоматически реплицируются на другие узлы. Например, в базе данных футбольного сервиса, таблица user может быть реплицирована на несколько серверов, чтобы обеспечить доступность профилей пользователей. Таким образом, если один сервер недоступен, данные о пользователях все равно будут доступны на других серверах.

Для реализации данной технологии в PostgreSQL 16 необходимо создать специального пользователя-репликатора на основном сервере и на всех репликах, куда мы хотим отправлять данные. Пользователь должен иметь специальное разрешение REPLICATION чтобы иметь возможность реализовывать репликацию.

Код создание пользователя приведён в листинге 6.1

|  |
| --- |
| CREATE USER replicator REPLICATION LOGIN CONNECTION LIMIT 10 ENCRYPTED PASSWORD '1234';  GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO replicator;  ALTER USER replicator WITH REPLICATION LOGIN; |

Листинг 6.1 – создание пользователя-репликатора

После создание пользователя репликатора на серверах, которые необходимо связать, надо изменить файлы параметров PostgreSQL.

В первую очередь необходимо в файле postresql.conf поменять значение параметра wal\_level, установив его на logical. Данное действие активирует логическу репликацию. Данное действие необходимо проделать на основном сервере и на репликах.

Далее необходимо создать публикацию на основном сервере, в ней мы будем определять данные, которые в дальнейшем будут реплицироваться. В случае данного проекта реплицироваться будут данные всех из всех таблиц.

Код создание публикации приведён в листинге 6.2

|  |
| --- |
| CREATE PUBLICATION goalscore\_publication FOR ALL TABLES; |

Листинг 6.2 – создание публикации на основном сервере

После создание публикации необходимо к ней подключиться со всех реплик, на которые мы будем отправлять данные. Для этого необходимо создать подписку на репликах с указанием необходимых параметров подключения к основному серверу.

Код создания подписки приведён в листинге 6.3

|  |
| --- |
| CREATE SUBSCRIPTION goalscore\_subscription  CONNECTION 'host=172.30.208.1 port=5432 dbname=goalscore user=replicator password=1234'  PUBLICATION goalscore\_publication; |

Листинг 6.3 – создание подписки на публикацию

Репликация данных в базе данных футбольного сервиса способствует обеспечению надежности и доступности данных. При этом следует учитывать, что репликация требует дополнительных ресурсов и может привести к небольшой задержке при обновлении данных. Оптимальное планирование и настройка репликации позволяет достичь баланса между надежностью и производительностью системы.

## 6.1 Вывод

В данном разделе была описана использованная технология репликации для обеспечения надежности и доступности данных.

Репликация позволяет создать копии базы данных на разных серверах, что обеспечивает резервное хранение данных и возможность их использования в случае сбоев или недоступности основного сервера. Таким образом, использование данной технологии повышает надежность и устойчивость работы футбольного сервиса.

# 7 Краткое описание приложения для демонстрации

## 7.1 Краткое описание приложения для демонстрации со стороны посетителя

Когда человек зайдет на сайт и до этого он не входил в аккаунт, ему будут доступны лишь три функции: просмотр главной страницы, регистрации и авторизация.

Форма регистрации содержит 3 поля: логин, который должен состоять от 2 до 32 символов; пароль, который должен содержать как минимум 2 символов и как максимум 32 символа; телефон, который должен состоять ровно из 12 чисел.

Форма авторизации содержит два поля: логин и пароль. Этих данных хватит для прохождения идентификации, аутентификации и авторизации. Они были описаны в прошлом абзаце.

При вводе неверных данных пользователь получит ошибки и уточнения к ним. Отображение ошибки показано на рисунке 7.1.

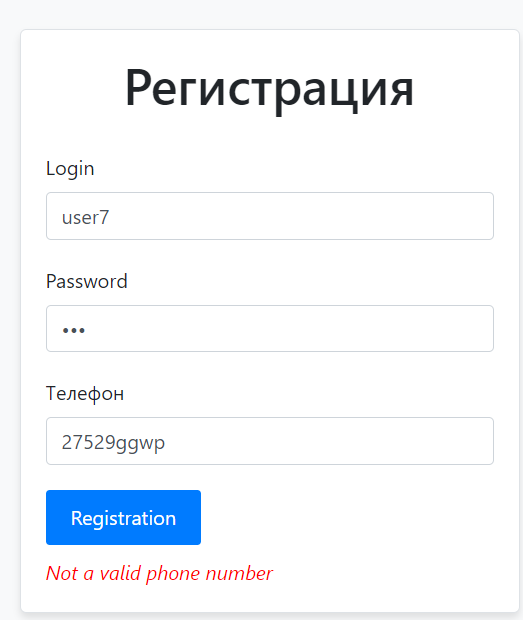


Рисунок 7.1 – Отображения ошибок при регистрации

Тем самым пользователю доступно лишь три функции и, чтобы воспользоваться приложением полностью, он должен создать или войти в существующий аккаунт.

## 7.2 Краткое описание приложения для демонстрации со стороны пользователя

Пользователю приложения доступно уже намного больше функций. Он может просматривать списки команд, списки игроков, матчи, просматривать информацию конкретного матча. Также он может создавать уникальную команду, управлять избранным и редактировать профиль.

Основная задача футбольного сервиса – предоставить футбольные данные и возможность их анализировать.

В управлении избранным пользователь может как добавлять игроков и команды, так и удалять их из избранного. Пример управления списка избранными игроками пользователя показан на рисунке 7.2.

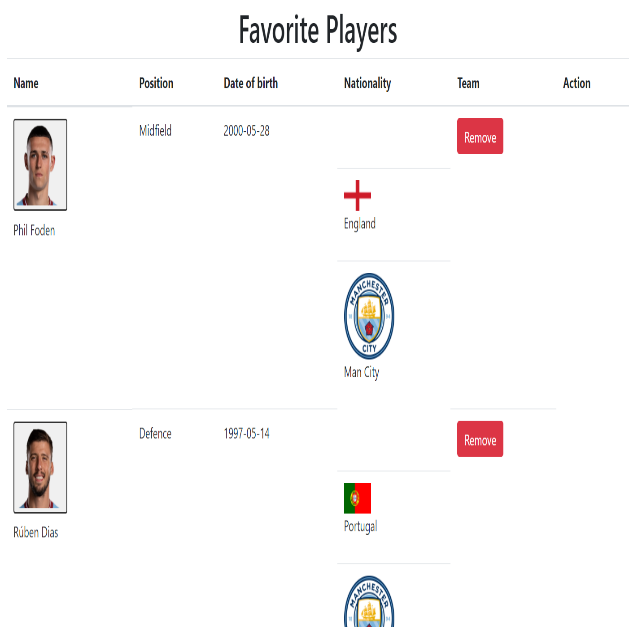


Рисунок 7.2 – Упраление избранными игроками

Пользователю доступно намного большем, чем обычному посетителю. Эта роль позволяет насладиться всем функционалом приложения Goalscore.

## 7.3 Краткое описание приложения для демонстрации со стороны администратора

Администратор является обычным пользователем, но с большим функционалом, который позволяет сохранять в приложении порядок.

Главные функции администратора – добавление футбольных данных. Это создание заявок команд на матч, создание матча, добавление игрока, изменение игрока, удаление игрока, добавление команды, изменение команды, удаление команды.

Пример панели администратора представлен на рисунке 7.3.

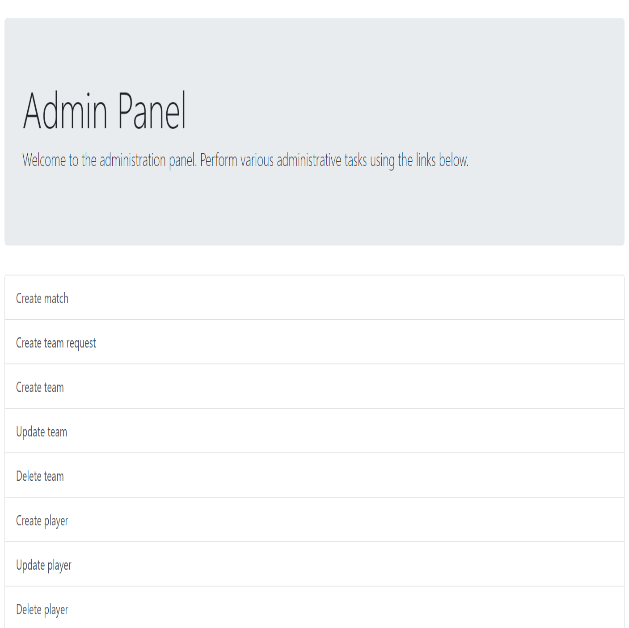


Рисунок 7.3 – Панель администратора

Аккаунт администратора позволяет управлять футбольной статистикой.

## 7.4 Вывод

В данном разделе были рассмотрены все основные роли, которые были реализованы в приложении, а также было рассмотрено, какие привилегии они дают. Приложение предоставляет 3 роли, на которые распределен весь функционал, что позволяет улучшить систему безопасности внутри сайта. Система ролей способствует эффективному использованию ресурсов приложения. Например, посетителям доступны только базовые функции, тогда как администраторам предоставлены расширенные возможности управления. Это позволяет каждой категории пользователей получать доступ к тем функциям, которые соответствуют их ролям и потребностям в приложении.

# 8 Руководство пользователя

## 8.1 Руководство пользователя

При открытии веб-приложения Goalscore вас будет встречать главная страница, в которой вы можете вы можете посмотреть список матчей без подробностей, а также перейти к входу и регистрации. Рисунок главной страницы на рисунке 8.1.

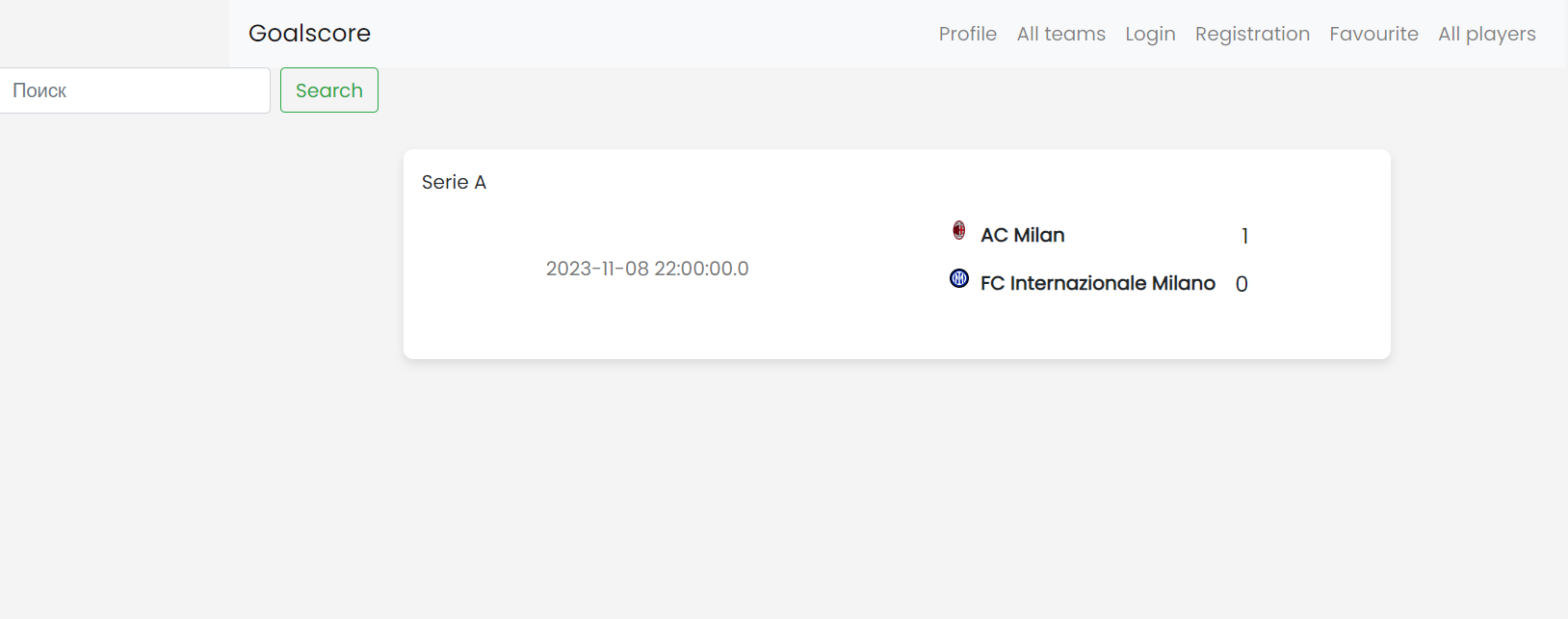


Рисунок 8.1 – Главная страница

После входа под определенным пользователем вам откроется возможность более подробно узнавать информацию о матчах, командах и игроках. Например более подробная информация о матче с возможностью добавления в избранное. Страница матча представлена на рисунке 8.2.

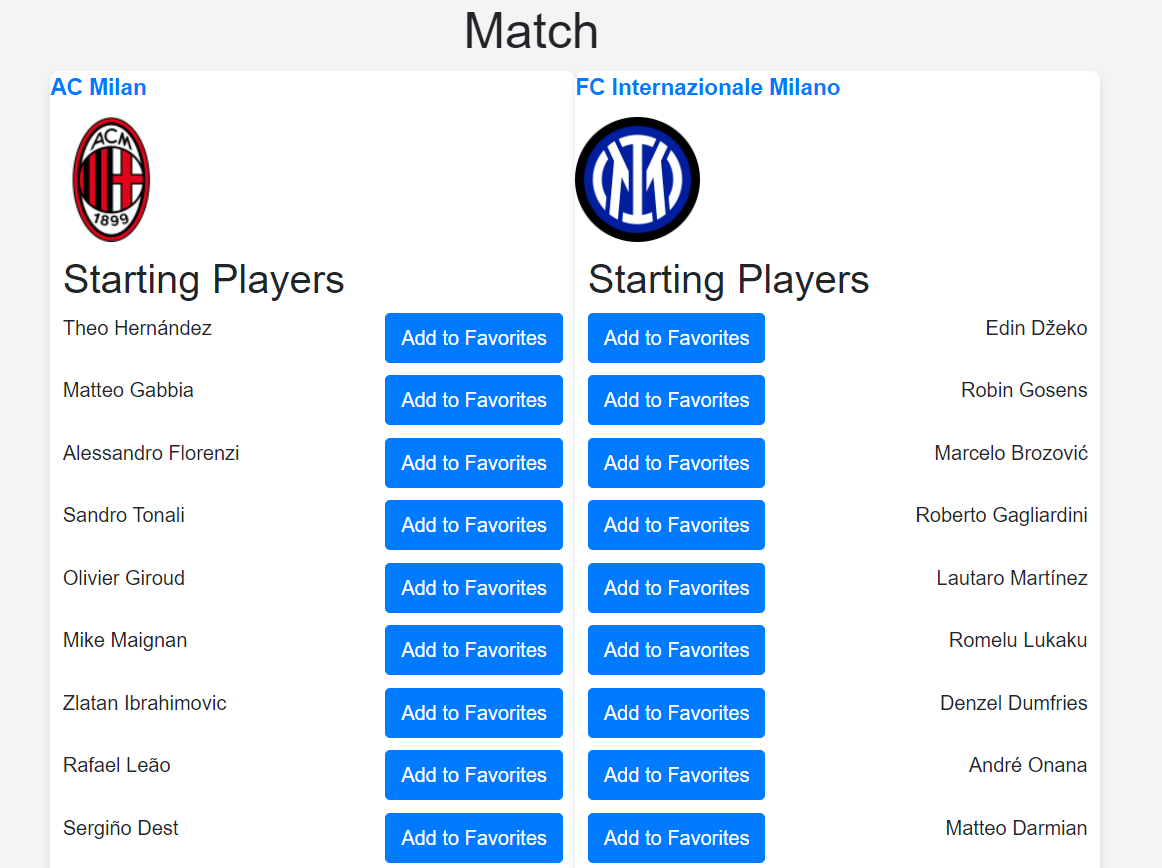


Рисунок 8.2 – Страница матча

Пользователь может создать свою команду из избранных игроков. Для этого ему надо перейти на страницу с избранным и перейти на страницу создания команды, где ему будет доступна интуитивно понятная форма. Рисунок формы создания команды представлен на рисунке 8.3.



Рисунок 8.3 –Создания создания команды пользователя

Пользователь может добавлять команды в избранное. Для этого он должен нажать на кнопку “Add to Favourites” на странице информации о команде, показанное на рисунке 8.4.

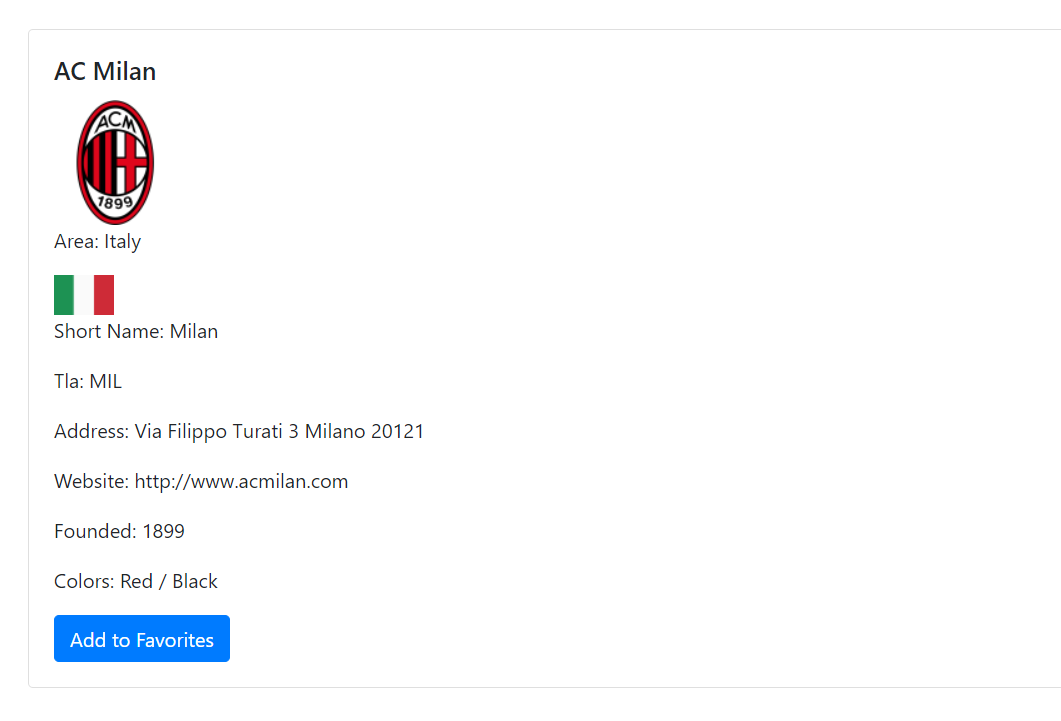


Рисунок 8.4 – Страница команды

Так же есть аналогичная возможность с игроками. Можно добавлять в избранное и просматривать. Список избранных игроков показан на рисунке 8.5, что означает, что запись голоса начата. Для отправки голосового сообщения, нужно снова нажать на эту кнопку.

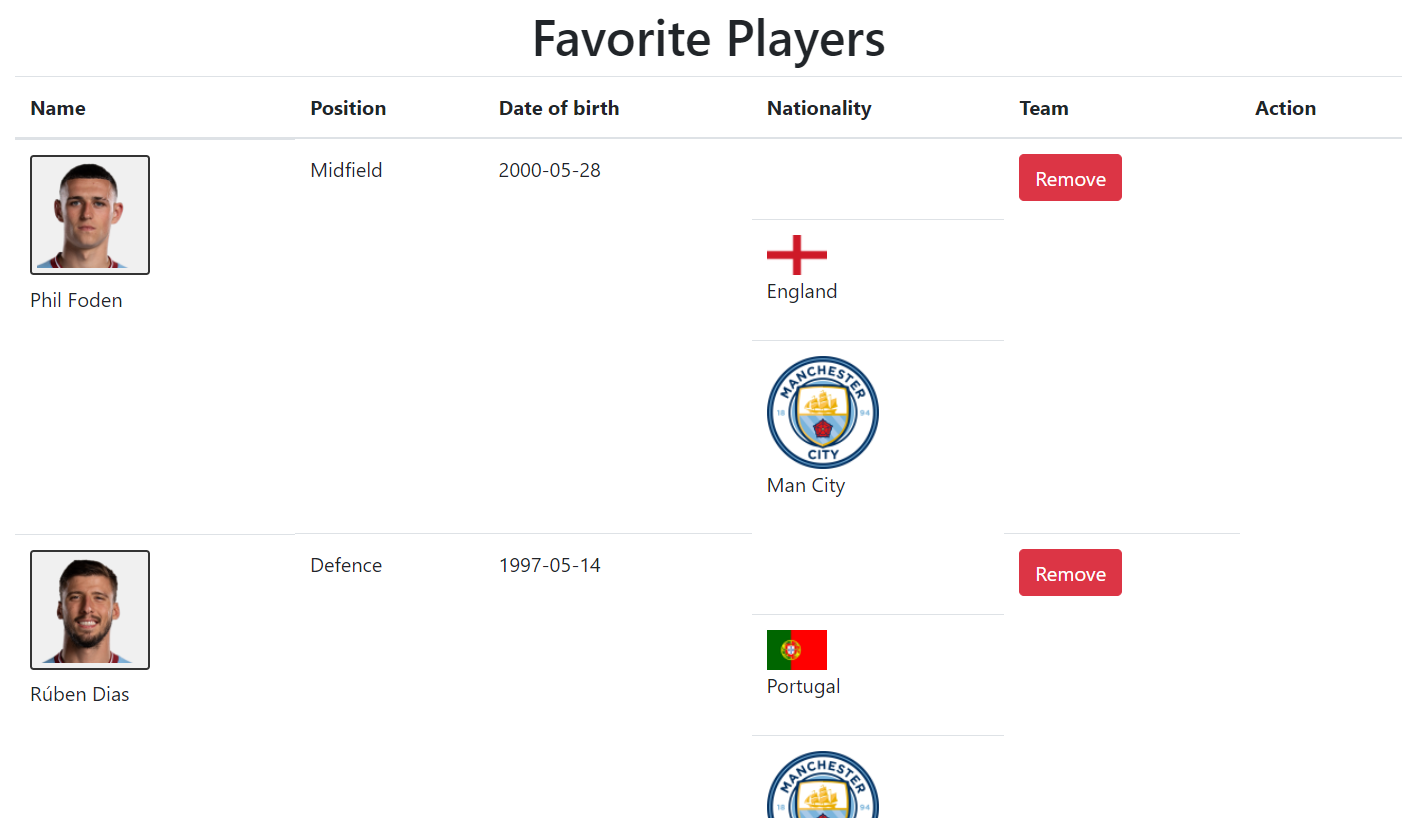


Рисунок 8.5 – Управление избранными игроками

Чтобы пользователю было удобнее искать матчи, осуществлен поиск матчей по названию команд матча, как показано на рисунке 8.6.

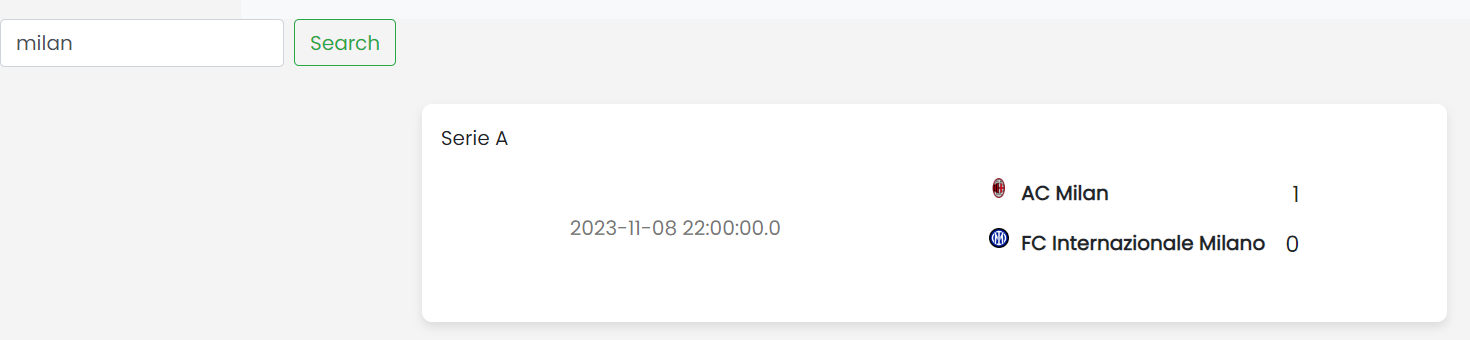


Рисунок 8.6 – Поиск матчей

Мы рассмотрели основной функционал данного приложения. Он должен помочь освоится новому пользователю. Мы увидели, что интерфейс более-менее понятный и отзывчивый. Дизайн приложения довольно-таки привлекательный.

## 8.2 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить Java приложение, которое будет обрабатывать пользовательские запросы и взаимодействовать с сервером.

После выполнения этих шагов приложение будет полностью готово к работе и пользователь сможет начать использовать его функционал.

## 8.3 Вывод

В данном разделе были рассмотрены функциональные возможности приложения, а также права доступа для пользователей с различными ролями. Было показано, что наша платформа позволяет пользователям создавать плейлисты, искать треки по названию или исполнителю, а также изменять свой профиль и пароль.

Кроме того, администратор имеет доступ к управлению пользователями, треками и жанрами. Для запуска приложения необходимо запустить серверную часть, которая соединяет базу данных и Java-приложение.

# Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление информацией. В данной работе была поставлена задача разработки базы данных для музыкальной площадки с использованием технологии применения мультимедийных типов данных в СУБД PostgreSQL.

В процессе выполнения работы были использованы различные объекты, включая таблицы, триггеры и функции, чтобы обеспечить структурированное хранение данных и своевременный доступ к ним. В результате, цель работы была успешно достигнута, и база данных готова к использованию. Были разработаны роли для управления доступом к данным и обеспечения безопасности.

Тестирование базы данных было проведено при использовании большого объема данных, и результаты были положительными. Были реализованы процедуры для импорта и экспорта данных в формате JSON, что обеспечило удобство использования и управления данными.

Одной из ключевых особенностей разработанной базы данных является технология хранения мультимедийных данных, что позволяет эффективно управлять медиа-файлами на площадке.

Таким образом, была успешно выполнена задача по разработке базы данных для музыкальной площадки на основе СУБД PostgreSQL. Разработанная база данных позволяет хранить и управлять большим объемом музыкальных данных, обеспечивает безопасный доступ к ним и предоставляет возможность импорта и экспорта данных в различных форматах. Кроме того, технология хранения мультимедийных данных позволяет эффективно управлять медиа-файлами на площадке.

# Список используемых источников

1. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 18.04.2023.
2. Postgresqltutorial.com [Электронный ресурс] / Режим доступа: https://www.postgresqltutorial.com/ – Дата доступа: 18.04.2023.
3. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 18.04.2023.
4. Нистюк О. А. курс лекций по дисциплине «Базы данных».

# Приложение А

|  |
| --- |
| CREATE OR REPLACE FUNCTION public.check\_user\_star\_team\_exists(  p\_user\_id integer)  RETURNS boolean  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  user\_exists BOOLEAN;  BEGIN  SELECT TRUE INTO user\_exists  FROM public.user\_star\_team  WHERE user\_id = p\_user\_id  LIMIT 1;  RETURN user\_exists;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.find\_user\_by\_login(  text)  RETURNS SETOF "user"  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT \*  FROM public.user  WHERE username = $1;    IF NOT FOUND THEN  RAISE EXCEPTION 'no user with such login: %.', $1;  END IF;  RETURN;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_all\_matches(  )  RETURNS TABLE(match\_id integer, competition\_id integer, attendance integer, home\_match\_team\_id integer, away\_match\_team\_id integer, home\_goal\_score smallint, away\_goal\_score smallint, match\_datetime timestamp without time zone)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT  m.id AS match\_id,  m.competition\_id,  m.attendance,  m.home\_match\_team\_id,  m.away\_match\_team\_id,  m.home\_goal\_score,  m.away\_goal\_score,  m.match\_datetime  FROM  public.match m;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_all\_teams(  )  RETURNS SETOF teams  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  DECLARE  r teams%rowtype;  BEGIN  FOR r IN  SELECT \* FROM public.teams  LOOP  RETURN NEXT r;  END LOOP;  RETURN;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_area\_by\_id(  p\_area\_id integer)  RETURNS TABLE(area\_id integer, area\_name character varying)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT  id AS area\_id,  name AS area\_name  FROM  public.areas  WHERE  id = p\_area\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_favorite\_players\_data(  p\_user\_id integer)  RETURNS TABLE(player\_id integer, player\_name character varying, player\_position character varying, player\_date\_of\_birth date, player\_nationality character, team\_short\_name character varying, team\_id integer)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT  p.id AS player\_id,  p.name AS player\_name,  p."position" AS player\_position,  p.date\_of\_birth AS player\_date\_of\_birth,  p.nationality AS player\_nationality,  t.short\_name AS team\_short\_name,  t.id AS team\_id  FROM  public.user\_favorite\_players ufp  JOIN  public.players p ON ufp.player\_id = p.id  JOIN  public.teams t ON p.team\_id = t.id  WHERE  ufp.user\_id = p\_user\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_favorite\_teams\_by\_user\_id(  p\_user\_id integer)  RETURNS TABLE(favorite\_team\_id integer, user\_id integer, team\_id integer)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT  uft.id AS favorite\_team\_id,  uft.user\_id,  uft.team\_id  FROM  public.user\_favorite\_teams uft  WHERE  uft.user\_id = p\_user\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_match\_by\_id(  p\_match\_id integer)  RETURNS TABLE(match\_id integer, competition\_id integer, attendance integer, home\_match\_team\_id integer, away\_match\_team\_id integer, home\_goal\_score smallint, away\_goal\_score smallint, match\_datetime timestamp without time zone)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT  m.id AS match\_id,  m.competition\_id,  m.attendance,  m.home\_match\_team\_id,  m.away\_match\_team\_id,  m.home\_goal\_score,  m.away\_goal\_score,  m.match\_datetime  FROM  public.match m  WHERE  m.id = p\_match\_id;  RETURN;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_match\_info\_by\_id(  \_match\_id integer)  RETURNS TABLE(match\_id integer, competition\_name character varying, area\_name character varying, match\_datetime timestamp without time zone, home\_team\_name character varying, home\_team\_id integer, away\_team\_name character varying, away\_team\_id integer, home\_goal smallint, away\_goal smallint, attendance integer)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT  m.id AS match\_id,  c.name AS competition\_name,  a.name AS area\_name,  m.match\_datetime,  home\_team.name AS home\_team\_name,  home\_team.id AS home\_team\_id,  away\_team.name AS away\_team\_name,  away\_team.id AS away\_team\_id,  m.home\_goal\_score as home\_goal,  m.away\_goal\_score as away\_goal,  m.attendance  FROM  public.match m  JOIN public.competitions c ON m.competition\_id = c.id  JOIN public.areas a ON c.area\_id = a.id  JOIN public.team\_on\_match tom\_home ON m.home\_match\_team\_id = tom\_home.id  JOIN public.teams home\_team ON tom\_home.team\_id = home\_team.id  JOIN public.team\_on\_match tom\_away ON m.away\_match\_team\_id = tom\_away.id  JOIN public.teams away\_team ON tom\_away.team\_id = away\_team.id  WHERE m.id = \_match\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_matches\_info(  )  RETURNS TABLE(match\_id integer, competition\_name character varying, area\_name character varying, match\_datetime timestamp without time zone, home\_team\_name character varying, home\_team\_id integer, away\_team\_name character varying, away\_team\_id integer, home\_goal smallint, away\_goal smallint)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT  m.id AS match\_id,  c.name AS competition\_name,  a.name AS area\_name,  m.match\_datetime,  home\_team.name AS home\_team\_name,  home\_team.id AS home\_team\_id,  away\_team.name AS away\_team\_name,  away\_team.id AS away\_team\_id,  m.home\_goal\_score as home\_goal,  m.away\_goal\_score as away\_goal  FROM  public.match m  JOIN public.competitions c ON m.competition\_id = c.id  JOIN public.areas a ON c.area\_id = a.id  JOIN public.team\_on\_match tom\_home ON m.home\_match\_team\_id = tom\_home.id  JOIN public.teams home\_team ON tom\_home.team\_id = home\_team.id  JOIN public.team\_on\_match tom\_away ON m.away\_match\_team\_id = tom\_away.id  JOIN public.teams away\_team ON tom\_away.team\_id = away\_team.id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_players\_range(  start\_index integer)  RETURNS TABLE(player\_id integer, player\_name character varying, player\_position character varying, player\_date\_of\_birth date, player\_nationality character varying, team\_short\_name character varying, team\_id integer)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT  p.id::INT AS player\_id,  p.name::VARCHAR AS player\_name,  p."position"::VARCHAR AS player\_position,  p.date\_of\_birth::DATE AS player\_date\_of\_birth,  p.nationality::VARCHAR AS player\_nationality,  t.short\_name::VARCHAR AS team\_short\_name,  t.id::INT AS team\_id  FROM  public.players p  JOIN public.teams t ON p.team\_id = t.id  ORDER BY  p.id  OFFSET start\_index  LIMIT 100;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_searched\_matches\_info(  team\_search character varying)  RETURNS TABLE(match\_id integer, competition\_name character varying, area\_name character varying, match\_datetime timestamp without time zone, home\_team\_name character varying, home\_team\_id integer, away\_team\_name character varying, away\_team\_id integer, home\_goal smallint, away\_goal smallint)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT  m.id AS match\_id,  c.name AS competition\_name,  a.name AS area\_name,  m.match\_datetime,  home\_team.name AS home\_team\_name,  home\_team.id AS home\_team\_id,  away\_team.name AS away\_team\_name,  away\_team.id AS away\_team\_id,  m.home\_goal\_score as home\_goal,  m.away\_goal\_score as away\_goal  FROM  public.match m  JOIN public.competitions c ON m.competition\_id = c.id  JOIN public.areas a ON c.area\_id = a.id  JOIN public.team\_on\_match tom\_home ON m.home\_match\_team\_id = tom\_home.id  JOIN public.teams home\_team ON tom\_home.team\_id = home\_team.id  JOIN public.team\_on\_match tom\_away ON m.away\_match\_team\_id = tom\_away.id  JOIN public.teams away\_team ON tom\_away.team\_id = away\_team.id  WHERE  team\_search IS NULL  OR home\_team.name ILIKE '%' || team\_search || '%'  OR away\_team.name ILIKE '%' || team\_search || '%';  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_starting\_players(  team\_on\_match\_id integer)  RETURNS TABLE(player\_id integer, player\_name character varying, player\_position text)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT DISTINCT ON (p.id)  p.id AS player\_id,  p.name AS player\_name,  CASE  WHEN tm.goalkeeper\_id = p.id THEN 'goalkeeper'  WHEN tm.left\_defender\_id = p.id THEN 'left defender'  WHEN tm.central\_1\_defender\_id = p.id THEN 'central 1 defender'  WHEN tm.central\_2\_defender\_id = p.id THEN 'central 2 defender'  WHEN tm.right\_defender\_id = p.id THEN 'right defender'  WHEN tm.left\_midfielder\_id = p.id THEN 'left midfielder'  WHEN tm.central\_midfielder\_id = p.id THEN 'central midfielder'  WHEN tm.right\_midfielder\_id = p.id THEN 'right midfielder'  WHEN tm.left\_winger\_id = p.id THEN 'left winger'  WHEN tm.striker\_id = p.id THEN 'striker'  WHEN tm.right\_winger\_id = p.id THEN 'right winger'  END AS player\_position  FROM  team\_on\_match tm  INNER JOIN players p ON tm.team\_id = p.team\_id  WHERE  tm.id = team\_on\_match\_id;  RETURN;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_team\_by\_id(  team\_id integer)  RETURNS TABLE(id integer, area\_id integer, name character varying, short\_name character varying, tla character varying, address character varying, website character varying, founded integer, club\_colors character varying)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT  t.id,  t.area\_id,  t.name,  t.short\_name,  t.tla,  t.address,  t.website,  t.founded,  t.club\_colors  FROM  public.teams t  WHERE  t.id = get\_team\_by\_id.team\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_team\_on\_match\_by\_id(  team\_on\_match\_id integer)  RETURNS TABLE(id integer, team\_id integer, goalkeeper\_id integer, left\_defender\_id integer, central\_1\_defender\_id integer, central\_2\_defender\_id integer, right\_defender\_id integer, left\_midfielder\_id integer, central\_midfielder\_id integer, right\_midfielder\_id integer, left\_winger\_id integer, striker\_id integer, right\_winger\_id integer, comment character varying)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT  tm.id,  tm.team\_id,  tm.goalkeeper\_id,  tm.left\_defender\_id,  tm.central\_1\_defender\_id,  tm.central\_2\_defender\_id,  tm.right\_defender\_id,  tm.left\_midfielder\_id,  tm.central\_midfielder\_id,  tm.right\_midfielder\_id,  tm.left\_winger\_id,  tm.striker\_id,  tm.right\_winger\_id,  tm.comment  FROM  public.team\_on\_match tm  WHERE  tm.id = get\_team\_on\_match\_by\_id.team\_on\_match\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_token\_id\_by\_user\_id(  integer)  RETURNS integer  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  BEGIN  IF ((SELECT id FROM public.token WHERE user\_id=$1) IS NOT NULL) THEN  RETURN (SELECT id FROM public.token WHERE user\_id=$1);  END IF;  RETURN -1;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_total\_player\_pages(  )  RETURNS integer  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  total\_players INTEGER;  players\_per\_page INTEGER := 100;  BEGIN  SELECT COUNT(\*) INTO total\_players FROM public.players;  RETURN CEIL(total\_players / players\_per\_page) + 1;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_user\_by\_id(  \_user\_id integer)  RETURNS TABLE(id integer, username character varying, password character varying, favorite\_team\_id integer, profile\_image bytea, phone character varying, join\_date date, role character varying)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT  u.id,  u.username,  u.password,  u.favorite\_team\_id,  u.profile\_image,  u.phone,  u.join\_date,  u.role  FROM  public."user" u  WHERE  u.id = \_user\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_user\_id\_by\_token(  \_token character varying)  RETURNS integer  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  \_user\_id integer;  BEGIN  SELECT user\_id INTO \_user\_id  FROM public.token  WHERE token = \_token;  RETURN \_user\_id;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.get\_user\_role\_by\_token(  input\_token character varying)  RETURNS character varying  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  user\_role character varying;  BEGIN  SELECT u.role INTO user\_role  FROM public."user" u  JOIN public.token t ON u.id = t.user\_id  WHERE t.token = input\_token;  RETURN user\_role;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.is\_token\_with\_user\_id(  integer)  RETURNS boolean  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  BEGIN  IF ((SELECT COUNT(\*) FROM public.token WHERE user\_id=$1) = 1) THEN  RETURN TRUE;  END IF;  RETURN FALSE;  END;  $BODY$;  CREATE OR REPLACE FUNCTION public.register\_user(  \_username text,  \_password text,  \_phone text)  RETURNS integer  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE \_user\_id integer;  BEGIN  call public.insert\_new\_user(\_username, \_password, \_phone, \_user\_id);  RETURN \_user\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.change\_token(  IN \_user\_id integer,  IN \_token text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  \_id integer;  BEGIN  \_id := public.get\_token\_id\_by\_user\_id(\_user\_id);  IF \_id = -1 THEN  RAISE EXCEPTION 'no token';  END IF;  UPDATE public.token tr  SET user\_id = \_user\_id,  token = \_token  WHERE id = \_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.delete\_player(  IN p\_player\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF NOT EXISTS (SELECT 1 FROM public.players WHERE id = p\_player\_id) THEN  RAISE EXCEPTION 'Player with id % not found', p\_player\_id;  END IF;  DELETE FROM public.players WHERE id = p\_player\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.delete\_team(  IN p\_team\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  player\_count integer;  team\_on\_match\_count integer;  player\_ids integer[];  BEGIN  IF NOT EXISTS (SELECT 1 FROM public.teams WHERE id = p\_team\_id) THEN  RAISE EXCEPTION 'Team with id % does not exist.', p\_team\_id;  END IF;    SELECT COUNT(\*) INTO player\_count  FROM public.players  WHERE team\_id = p\_team\_id;  IF player\_count > 0 THEN  DELETE FROM public.user\_favorite\_players  WHERE player\_id IN (SELECT id FROM public.players WHERE team\_id = p\_team\_id);  DELETE FROM public.players  WHERE team\_id = p\_team\_id;  END IF;  SELECT COUNT(\*) INTO team\_on\_match\_count  FROM public.team\_on\_match  WHERE team\_id = p\_team\_id;  IF team\_on\_match\_count > 0 THEN  DELETE FROM public.team\_on\_match  WHERE team\_id = p\_team\_id;  END IF;  DELETE FROM public.teams  WHERE id = p\_team\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.export\_users\_to\_json(  )  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  user\_data JSON;  BEGIN  SELECT array\_to\_json(array\_agg(row\_to\_json(users.\*)))  INTO user\_data  FROM public."user" users;  RAISE NOTICE 'Exported user data to JSON: %', user\_data;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.import\_from\_json\_areas(  IN path\_to\_file text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  json\_string jsonb;    err\_code text;  msg\_text text;  exc\_context text;  msg\_detail text;  exc\_hint text;  BEGIN  json\_string := pg\_read\_file(path\_to\_file);  insert into public.areas  select (obj->>'id')::integer as id, (obj->>'name') as name  from jsonb\_array\_elements(json\_string) as obj;    RAISE NOTICE 'Success';  RAISE NOTICE 'Value of path\_to\_file: %', path\_to\_file;  RAISE NOTICE 'Value of json\_string: %', json\_string;    EXCEPTION WHEN OTHERS  THEN  GET STACKED DIAGNOSTICS  err\_code = RETURNED\_SQLSTATE,  msg\_text = MESSAGE\_TEXT,  exc\_context = PG\_CONTEXT,  msg\_detail = PG\_EXCEPTION\_DETAIL,  exc\_hint = PG\_EXCEPTION\_HINT;  RAISE NOTICE  'ERROR CODE: %  MESSAGE TEXT: %  CONTEXT: %  DETAIL: %  HINT: %',  err\_code, msg\_text, exc\_context, msg\_detail, exc\_hint;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.import\_from\_json\_competitions(  IN path\_to\_file text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  json\_string jsonb;    err\_code text;  msg\_text text;  exc\_context text;  msg\_detail text;  exc\_hint text;  BEGIN  json\_string := pg\_read\_file(path\_to\_file);  insert into public.competitions  select (obj->>'id')::integer as id, (obj->>'area\_id')::integer as area\_id, (obj->>'name') as name  from jsonb\_array\_elements(json\_string) as obj;    RAISE NOTICE 'Success';  RAISE NOTICE 'Value of path\_to\_file: %', path\_to\_file;  RAISE NOTICE 'Value of json\_string: %', json\_string;    EXCEPTION WHEN OTHERS  THEN  GET STACKED DIAGNOSTICS  err\_code = RETURNED\_SQLSTATE,  msg\_text = MESSAGE\_TEXT,  exc\_context = PG\_CONTEXT,  msg\_detail = PG\_EXCEPTION\_DETAIL,  exc\_hint = PG\_EXCEPTION\_HINT;  RAISE NOTICE  'ERROR CODE: %  MESSAGE TEXT: %  CONTEXT: %  DETAIL: %  HINT: %',  err\_code, msg\_text, exc\_context, msg\_detail, exc\_hint;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.import\_from\_json\_players(  IN path\_to\_file text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  json\_string jsonb;    err\_code text;  msg\_text text;  exc\_context text;  msg\_detail text;  exc\_hint text;  BEGIN  json\_string := pg\_read\_file(path\_to\_file);  insert into public.players  select (obj->>'id')::integer as id,  (obj->>'name') as name,  (obj->>'position') as position,  (obj->>'date\_of\_birth')::date as date\_of\_birth,  (obj->>'nationality') as nationality,  (obj->>'team\_id')::integer as team\_id  from jsonb\_array\_elements(json\_string) as obj;    RAISE NOTICE 'Success';  RAISE NOTICE 'Value of path\_to\_file: %', path\_to\_file;  RAISE NOTICE 'Value of json\_string: %', json\_string;    EXCEPTION WHEN OTHERS  THEN  GET STACKED DIAGNOSTICS  err\_code = RETURNED\_SQLSTATE,  msg\_text = MESSAGE\_TEXT,  exc\_context = PG\_CONTEXT,  msg\_detail = PG\_EXCEPTION\_DETAIL,  exc\_hint = PG\_EXCEPTION\_HINT;  RAISE NOTICE  'ERROR CODE: %  MESSAGE TEXT: %  CONTEXT: %  DETAIL: %  HINT: %',  err\_code, msg\_text, exc\_context, msg\_detail, exc\_hint;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.import\_from\_json\_teams(  IN path\_to\_file text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  json\_string jsonb;    err\_code text;  msg\_text text;  exc\_context text;  msg\_detail text;  exc\_hint text;  BEGIN  json\_string := pg\_read\_file(path\_to\_file);  insert into public.teams  select (obj->>'id')::integer as id,  (obj->>'area\_id')::integer as area\_id,  (obj->>'name') as name,  (obj->>'short\_name') as short\_name,  (obj->>'tla') as tla,  (obj->>'address') as address,  (obj->>'website') as website,  (obj->>'founded')::integer as founded,  (obj->>'club\_colors') as club\_colors  from jsonb\_array\_elements(json\_string) as obj;    RAISE NOTICE 'Success';  RAISE NOTICE 'Value of path\_to\_file: %', path\_to\_file;  RAISE NOTICE 'Value of json\_string: %', json\_string;    EXCEPTION WHEN OTHERS  THEN  GET STACKED DIAGNOSTICS  err\_code = RETURNED\_SQLSTATE,  msg\_text = MESSAGE\_TEXT,  exc\_context = PG\_CONTEXT,  msg\_detail = PG\_EXCEPTION\_DETAIL,  exc\_hint = PG\_EXCEPTION\_HINT;  RAISE NOTICE  'ERROR CODE: %  MESSAGE TEXT: %  CONTEXT: %  DETAIL: %  HINT: %',  err\_code, msg\_text, exc\_context, msg\_detail, exc\_hint;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_match(  IN \_competition\_id integer,  IN \_attendance integer,  IN \_home\_match\_team\_id integer,  IN \_away\_match\_team\_id integer,  IN \_home\_goal\_score smallint,  IN \_away\_goal\_score smallint,  IN \_match\_datetime timestamp without time zone)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF \_home\_match\_team\_id = \_away\_match\_team\_id THEN  RAISE EXCEPTION 'home\_match\_team\_id cannot be equal to away\_match\_team\_id';  END IF;  INSERT INTO public.match (  competition\_id,  attendance,  home\_match\_team\_id,  away\_match\_team\_id,  home\_goal\_score,  away\_goal\_score,  match\_datetime  )  VALUES (  \_competition\_id,  \_attendance,  \_home\_match\_team\_id,  \_away\_match\_team\_id,  \_home\_goal\_score,  \_away\_goal\_score,  \_match\_datetime  );  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_new\_team\_on\_match(  IN \_team\_id integer,  IN \_goalkeeper\_id integer,  IN \_left\_defender\_id integer,  IN \_central\_1\_defender\_id integer,  IN \_central\_2\_defender\_id integer,  IN \_right\_defender\_id integer,  IN \_left\_midfielder\_id integer,  IN \_central\_midfielder\_id integer,  IN \_right\_midfielder\_id integer,  IN \_left\_winger\_id integer,  IN \_striker\_id integer,  IN \_right\_winger\_id integer,  IN \_comment text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  team\_ids INTEGER[];  player\_set INTEGER[];  BEGIN  team\_ids := ARRAY[\_goalkeeper\_id, \_left\_defender\_id, \_central\_1\_defender\_id,  \_central\_2\_defender\_id, \_right\_defender\_id, \_left\_midfielder\_id,  \_central\_midfielder\_id, \_right\_midfielder\_id, \_left\_winger\_id,  \_striker\_id, \_right\_winger\_id];  player\_set := ARRAY(SELECT DISTINCT unnest(team\_ids));  IF array\_length(team\_ids, 1) <> array\_length(player\_set, 1) THEN  RAISE EXCEPTION 'All player IDs must be unique';  END IF;    FOR i IN 1..array\_length(team\_ids, 1) LOOP  IF \_team\_id <> (SELECT team\_id FROM public.players WHERE id = team\_ids[i]) THEN  RAISE EXCEPTION 'team\_id does not match for player with id %', team\_ids[i];  END IF;  END LOOP;  INSERT INTO public.team\_on\_match (  team\_id,  goalkeeper\_id,  left\_defender\_id,  central\_1\_defender\_id,  central\_2\_defender\_id,  right\_defender\_id,  left\_midfielder\_id,  central\_midfielder\_id,  right\_midfielder\_id,  left\_winger\_id,  striker\_id,  right\_winger\_id,  comment  ) VALUES (  \_team\_id,  \_goalkeeper\_id,  \_left\_defender\_id,  \_central\_1\_defender\_id,  \_central\_2\_defender\_id,  \_right\_defender\_id,  \_left\_midfielder\_id,  \_central\_midfielder\_id,  \_right\_midfielder\_id,  \_left\_winger\_id,  \_striker\_id,  \_right\_winger\_id,  \_comment  );  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_new\_token(  IN \_user\_id integer,  IN \_token text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  id\_before\_error integer;    err\_code text;  msg\_text text;  exc\_context text;  msg\_detail text;  exc\_hint text;  BEGIN  INSERT INTO public.token(user\_id, token)  VALUES(\_user\_id, \_token);    EXCEPTION WHEN OTHERS  THEN    GET STACKED DIAGNOSTICS  err\_code = RETURNED\_SQLSTATE,  msg\_text = MESSAGE\_TEXT,  exc\_context = PG\_CONTEXT,  msg\_detail = PG\_EXCEPTION\_DETAIL,  exc\_hint = PG\_EXCEPTION\_HINT;  --для выводы в postgree  RAISE NOTICE 'ERROR CODE: % MESSAGE TEXT: % CONTEXT: % DETAIL: % HINT: %',  err\_code, msg\_text, exc\_context, msg\_detail, exc\_hint;  --для вывода в приложении  RAISE EXCEPTION 'ERROR CODE: % MESSAGE TEXT: % CONTEXT: % DETAIL: % HINT: %',  err\_code, msg\_text, exc\_context, msg\_detail, exc\_hint;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_new\_user(  IN \_username text,  IN \_password text,  IN \_phone text,  OUT user\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  user\_id:=NULL::integer;  INSERT INTO public."user"(id, username, password, phone, join\_date, role)  VALUES(nextval('public.user\_id\_sequence'), \_username, \_password, \_phone, now(),  CASE WHEN currval('public.user\_id\_sequence') < 16 THEN 'ROLE\_ADMIN'  ELSE 'ROLE\_USER'  END)  RETURNING id INTO user\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_player(  IN p\_id integer,  IN p\_name character varying,  IN p\_position character varying,  IN p\_date\_of\_birth date,  IN p\_nationality character,  IN p\_team\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF EXISTS (SELECT 1 FROM public.players WHERE id = p\_id) THEN  RAISE EXCEPTION 'Player with id % already exists', p\_id;  END IF;  IF NOT EXISTS (SELECT 1 FROM public.teams WHERE id = p\_team\_id) THEN  RAISE EXCEPTION 'Team with id % not found', p\_team\_id;  END IF;  IF p\_date\_of\_birth > CURRENT\_DATE THEN  RAISE EXCEPTION 'Date of birth cannot be in the future';  END IF;  IF NOT EXISTS (  SELECT 1  FROM public.areas  WHERE name = p\_nationality  ) THEN  RAISE EXCEPTION 'Nationality % not found in areas table', p\_nationality;  END IF;  CASE  WHEN p\_position IN ('Goalkeeper', 'Defence', 'Midfield', 'Offence') THEN  -- ok  ELSE  RAISE EXCEPTION 'Invalid position: %', p\_position;  END CASE;    INSERT INTO public.players (id, name, "position", date\_of\_birth, nationality, team\_id)  VALUES (p\_id, p\_name, p\_position, p\_date\_of\_birth, p\_nationality, p\_team\_id);  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_team(  IN p\_id integer,  IN p\_area\_id integer,  IN p\_name character varying,  IN p\_short\_name character varying,  IN p\_tla character varying,  IN p\_address character varying,  IN p\_website character varying,  IN p\_founded integer,  IN p\_club\_colors character varying)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF EXISTS (SELECT 1 FROM public.teams WHERE id = p\_id) THEN  RAISE EXCEPTION 'Team with id % already exists.', p\_id;  END IF;  IF NOT EXISTS (SELECT 1 FROM public.areas WHERE id = p\_area\_id) THEN  RAISE EXCEPTION 'Invalid area\_id: %.', p\_area\_id;  END IF;  IF p\_founded > 2023 THEN  RAISE EXCEPTION 'Invalid founded year: %, must be less than or equal to 2023.', p\_founded;  END IF;  IF p\_name = '' OR p\_short\_name = '' OR p\_tla = '' OR p\_address = '' OR p\_website = '' OR p\_club\_colors = '' THEN  RAISE EXCEPTION 'All string parameters must be non-empty.';  END IF;  INSERT INTO public.teams (  id, area\_id, name, short\_name, tla,  address, website, founded, club\_colors  ) VALUES (  p\_id, p\_area\_id, p\_name, p\_short\_name, p\_tla,  p\_address, p\_website, p\_founded, p\_club\_colors  );  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_user\_favorite\_player(  IN p\_user\_id integer,  IN p\_player\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF EXISTS (SELECT 1 FROM public.user\_favorite\_players WHERE user\_id = p\_user\_id AND player\_id = p\_player\_id) THEN  RAISE EXCEPTION 'User % has already added Player % to favorites', p\_user\_id, p\_player\_id;  ELSE  INSERT INTO public.user\_favorite\_players (user\_id, player\_id)  VALUES (p\_user\_id, p\_player\_id);  END IF;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_user\_favorite\_team(  IN p\_user\_id integer,  IN p\_team\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF EXISTS (SELECT 1 FROM public.user\_favorite\_teams WHERE user\_id = p\_user\_id AND team\_id = p\_team\_id) THEN  RAISE EXCEPTION 'User % has already added Team % to favorites', p\_user\_id, p\_team\_id;  ELSE  INSERT INTO public.user\_favorite\_teams (user\_id, team\_id)  VALUES (p\_user\_id, p\_team\_id);  END IF;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.insert\_user\_star\_team(  IN p\_user\_id integer,  IN p\_goalkeeper\_id integer,  IN p\_left\_defender\_id integer,  IN p\_central\_1\_defender\_id integer,  IN p\_central\_2\_defender\_id integer,  IN p\_right\_defender\_id integer,  IN p\_left\_midfielder\_id integer,  IN p\_central\_midfielder\_id integer,  IN p\_right\_midfielder\_id integer,  IN p\_left\_winger\_id integer,  IN p\_striker\_id integer,  IN p\_right\_winger\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  team\_ids INTEGER[];  player\_set INTEGER[];  BEGIN  team\_ids := ARRAY[p\_goalkeeper\_id, p\_left\_defender\_id, p\_central\_1\_defender\_id,  p\_central\_2\_defender\_id, p\_right\_defender\_id, p\_left\_midfielder\_id,  p\_central\_midfielder\_id, p\_right\_midfielder\_id, p\_left\_winger\_id,  p\_striker\_id, p\_right\_winger\_id];  player\_set := ARRAY(SELECT DISTINCT unnest(team\_ids));  IF array\_length(team\_ids, 1) <> array\_length(player\_set, 1) THEN  RAISE EXCEPTION 'All player IDs must be unique';  END IF;  INSERT INTO public.user\_star\_team (  user\_id,  goalkeeper\_id,  left\_defender\_id,  central\_1\_defender\_id,  central\_2\_defender\_id,  right\_defender\_id,  left\_midfielder\_id,  central\_midfielder\_id,  right\_midfielder\_id,  left\_winger\_id,  striker\_id,  right\_winger\_id  ) VALUES (  p\_user\_id,  p\_goalkeeper\_id,  p\_left\_defender\_id,  p\_central\_1\_defender\_id,  p\_central\_2\_defender\_id,  p\_right\_defender\_id,  p\_left\_midfielder\_id,  p\_central\_midfielder\_id,  p\_right\_midfielder\_id,  p\_left\_winger\_id,  p\_striker\_id,  p\_right\_winger\_id  );  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.populate\_areas(  )  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  start\_id INTEGER := 3000;  end\_id INTEGER := 103000;  BEGIN  FOR i IN start\_id..end\_id LOOP  INSERT INTO public.areas (id, name)  VALUES (i, 'Area ' || i);  END LOOP;  END;  $BODY$;  ALTER PROCEDURE public.populate\_areas()  OWNER TO postgres;  CREATE OR REPLACE PROCEDURE public.remove\_favorite\_player(  IN p\_user\_id integer,  IN p\_player\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM public.user\_favorite\_players  WHERE user\_id = p\_user\_id AND player\_id = p\_player\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.remove\_favorite\_team(  IN p\_user\_id integer,  IN p\_team\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM public.user\_favorite\_teams  WHERE user\_id = p\_user\_id AND team\_id = p\_team\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.update\_player(  IN p\_id integer,  IN p\_name character varying,  IN p\_position character varying,  IN p\_date\_of\_birth date,  IN p\_nationality character,  IN p\_team\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF NOT EXISTS (SELECT 1 FROM public.teams WHERE id = p\_team\_id) THEN  RAISE EXCEPTION 'Team with id % not found', p\_team\_id;  END IF;  IF p\_date\_of\_birth > CURRENT\_DATE THEN  RAISE EXCEPTION 'Date of birth cannot be in the future';  END IF;  IF NOT EXISTS (  SELECT 1  FROM public.areas  WHERE name = p\_nationality  ) THEN  RAISE EXCEPTION 'Nationality % not found in areas table', p\_nationality;  END IF;  CASE  WHEN p\_position IN ('Goalkeeper', 'Defence', 'Midfield', 'Offence') THEN  -- ok  ELSE  RAISE EXCEPTION 'Invalid position: %', p\_position;  END CASE;    UPDATE public.players  SET name = p\_name, "position" = p\_position,  date\_of\_birth = p\_date\_of\_birth, nationality = p\_nationality, team\_id = p\_team\_id  WHERE id = p\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.update\_team(  IN p\_id integer,  IN p\_area\_id integer,  IN p\_name character varying,  IN p\_short\_name character varying,  IN p\_tla character varying,  IN p\_address character varying,  IN p\_website character varying,  IN p\_founded integer,  IN p\_club\_colors character varying)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  IF NOT EXISTS (SELECT 1 FROM public.areas WHERE id = p\_area\_id) THEN  RAISE EXCEPTION 'Invalid area\_id: %.', p\_area\_id;  END IF;  IF p\_founded > 2023 THEN  RAISE EXCEPTION 'Invalid founded year: %, must be less than or equal to 2023.', p\_founded;  END IF;  IF p\_name = '' OR p\_short\_name = '' OR p\_tla = '' OR p\_address = '' OR p\_website = '' OR p\_club\_colors = '' THEN  RAISE EXCEPTION 'All string parameters must be non-empty.';  END IF;  UPDATE public.teams  SET  area\_id = p\_area\_id,  name = p\_name,  short\_name = p\_short\_name,  tla = p\_tla,  address = p\_address,  website = p\_website,  founded = p\_founded,  club\_colors = p\_club\_colors  WHERE id = p\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.update\_user\_profile\_image(  IN \_user\_id integer,  IN \_profile\_image bytea)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  UPDATE public."user"  SET profile\_image = \_profile\_image  WHERE id = \_user\_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.update\_user\_star\_team(  IN p\_user\_id integer,  IN p\_goalkeeper\_id integer,  IN p\_left\_defender\_id integer,  IN p\_central\_1\_defender\_id integer,  IN p\_central\_2\_defender\_id integer,  IN p\_right\_defender\_id integer,  IN p\_left\_midfielder\_id integer,  IN p\_central\_midfielder\_id integer,  IN p\_right\_midfielder\_id integer,  IN p\_left\_winger\_id integer,  IN p\_striker\_id integer,  IN p\_right\_winger\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  team\_ids INTEGER[];  player\_set INTEGER[];  BEGIN  team\_ids := ARRAY[p\_goalkeeper\_id, p\_left\_defender\_id, p\_central\_1\_defender\_id,  p\_central\_2\_defender\_id, p\_right\_defender\_id, p\_left\_midfielder\_id,  p\_central\_midfielder\_id, p\_right\_midfielder\_id, p\_left\_winger\_id,  p\_striker\_id, p\_right\_winger\_id];  player\_set := ARRAY(SELECT DISTINCT unnest(team\_ids));  IF array\_length(team\_ids, 1) <> array\_length(player\_set, 1) THEN  RAISE EXCEPTION 'All player IDs must be unique';  END IF;  UPDATE public.user\_star\_team  SET  goalkeeper\_id = p\_goalkeeper\_id,  left\_defender\_id = p\_left\_defender\_id,  central\_1\_defender\_id = p\_central\_1\_defender\_id,  central\_2\_defender\_id = p\_central\_2\_defender\_id,  right\_defender\_id = p\_right\_defender\_id,  left\_midfielder\_id = p\_left\_midfielder\_id,  central\_midfielder\_id = p\_central\_midfielder\_id,  right\_midfielder\_id = p\_right\_midfielder\_id,  left\_winger\_id = p\_left\_winger\_id,  striker\_id = p\_striker\_id,  right\_winger\_id = p\_right\_winger\_id  WHERE user\_id = p\_user\_id;  IF NOT FOUND THEN  RAISE EXCEPTION 'User with user\_id % not found', p\_user\_id;  END IF;  END;  $BODY$; |

Листинг 1 – Српиты процедур и функций

|  |
| --- |
| CREATE USER replicator REPLICATION LOGIN CONNECTION LIMIT 10 ENCRYPTED PASSWORD '1234';  create role ROLE\_UNAUTHORIZED;  grant connect on database goalscore to ROLE\_UNAUTHORIZED;  grant SELECT on public.user to ROLE\_UNAUTHORIZED;  grant INSERT on public.user to ROLE\_UNAUTHORIZED;  grant INSERT on token to ROLE\_UNAUTHORIZED;  grant select on match to ROLE\_UNAUTHORIZED;  grant execute on FUNCTION get\_searched\_matches\_info to ROLE\_UNAUTHORIZED;  grant execute on procedure change\_token to ROLE\_UNAUTHORIZED;  grant execute on procedure insert\_new\_user to ROLE\_UNAUTHORIZED;  grant execute on FUNCTION register\_user to ROLE\_UNAUTHORIZED;  grant execute on FUNCTION is\_token\_with\_user\_id to ROLE\_UNAUTHORIZED;  create role ROLE\_USER;  grant ROLE\_UNAUTHORIZED to ROLE\_USER;  grant insert on user\_favorite\_players to ROLE\_USER;  grant update on user\_favorite\_players to ROLE\_USER;  grant delete on user\_favorite\_players to ROLE\_USER;  grant insert on user\_favorite\_teams to ROLE\_USER;  grant update on user\_favorite\_teams to ROLE\_USER;  grant delete on user\_favorite\_teams to ROLE\_USER;  grant insert on user\_star\_team to ROLE\_USER;  grant update on user\_star\_team to ROLE\_USER;  grant delete on user\_star\_team to ROLE\_USER;  grant select on teams to ROLE\_USER;  grant select on players to ROLE\_USER;  create role ROLE\_ADMIN;  grant ROLE\_USER to ROLE\_ADMIN;  grant insert on match to ROLE\_ADMIN;  grant update on match to ROLE\_ADMIN;  grant delete on match to ROLE\_ADMIN;  grant insert on players to ROLE\_ADMIN;  grant update on players to ROLE\_ADMIN;  grant delete on players to ROLE\_ADMIN;  grant insert on team\_on\_match to ROLE\_ADMIN;  grant update on team\_on\_match to ROLE\_ADMIN;  grant delete on team\_on\_match to ROLE\_ADMIN;  grant insert on teams to ROLE\_ADMIN;  grant update on teams to ROLE\_ADMIN;  grant delete on teams to ROLE\_ADMIN;  grant insert on areas to ROLE\_ADMIN;  grant delete on areas to ROLE\_ADMIN;  grant insert on competitions to ROLE\_ADMIN;  grant delete on competitions to ROLE\_ADMIN; |

Листинг 2 – Скрипты создание ролей и пользователей

|  |
| --- |
| CREATE TABLE IF NOT EXISTS public.areas  (  id integer NOT NULL,  name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  CONSTRAINT areas\_pkey PRIMARY KEY (id)  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.competitions  (  id integer NOT NULL,  area\_id integer NOT NULL,  name character varying(64) COLLATE pg\_catalog."default" NOT NULL,  CONSTRAINT competitions\_pkey PRIMARY KEY (id),  CONSTRAINT area\_id FOREIGN KEY (area\_id)  REFERENCES public.areas (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.match  (  id integer NOT NULL DEFAULT nextval('match\_id\_seq'::regclass),  competition\_id integer NOT NULL,  attendance integer NOT NULL,  home\_match\_team\_id integer NOT NULL,  away\_match\_team\_id integer NOT NULL,  home\_goal\_score smallint NOT NULL,  away\_goal\_score smallint NOT NULL,  match\_datetime timestamp without time zone,  CONSTRAINT match\_pkey PRIMARY KEY (id),  CONSTRAINT match\_away\_match\_team\_id\_fkey FOREIGN KEY (away\_match\_team\_id)  REFERENCES public.team\_on\_match (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT match\_competition\_id\_fkey FOREIGN KEY (competition\_id)  REFERENCES public.competitions (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE RESTRICT,  CONSTRAINT match\_home\_match\_team\_id\_fkey FOREIGN KEY (home\_match\_team\_id)  REFERENCES public.team\_on\_match (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT check\_match\_datetime CHECK (match\_datetime <= (now() + '00:05:00'::interval))  )  TABLESPACE pg\_default;  CREATE TABLE IF NOT EXISTS public.players  (  id integer NOT NULL,  name character varying(64) COLLATE pg\_catalog."default" NOT NULL,  "position" character varying(32) COLLATE pg\_catalog."default" NOT NULL,  date\_of\_birth date NOT NULL,  nationality character(64) COLLATE pg\_catalog."default" NOT NULL,  team\_id integer NOT NULL,  CONSTRAINT players\_pkey PRIMARY KEY (id),  CONSTRAINT team\_id FOREIGN KEY (team\_id)  REFERENCES public.teams (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE RESTRICT  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.team\_on\_match  (  id integer NOT NULL DEFAULT nextval('team\_on\_match\_id\_seq'::regclass),  team\_id integer NOT NULL,  goalkeeper\_id integer NOT NULL,  left\_defender\_id integer NOT NULL,  central\_1\_defender\_id integer NOT NULL,  central\_2\_defender\_id integer NOT NULL,  right\_defender\_id integer NOT NULL,  left\_midfielder\_id integer NOT NULL,  central\_midfielder\_id integer NOT NULL,  right\_midfielder\_id integer NOT NULL,  left\_winger\_id integer NOT NULL,  striker\_id integer NOT NULL,  right\_winger\_id integer NOT NULL,  comment character varying(256) COLLATE pg\_catalog."default",  CONSTRAINT team\_on\_match\_pkey PRIMARY KEY (id),  CONSTRAINT central\_1\_defender\_id FOREIGN KEY (central\_1\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT central\_2\_defender\_id FOREIGN KEY (central\_2\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT central\_midfielder\_id FOREIGN KEY (central\_midfielder\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT goalkeeper\_id FOREIGN KEY (goalkeeper\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT left\_defender\_id FOREIGN KEY (left\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT left\_midfielder\_id FOREIGN KEY (left\_midfielder\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT left\_winger\_id FOREIGN KEY (left\_winger\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT right\_defender\_id FOREIGN KEY (right\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT right\_midfielder\_id FOREIGN KEY (right\_midfielder\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT right\_winger\_id FOREIGN KEY (right\_winger\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT striker\_id FOREIGN KEY (striker\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT team\_id FOREIGN KEY (team\_id)  REFERENCES public.teams (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.teams  (  id integer NOT NULL,  area\_id integer NOT NULL,  name character varying(64) COLLATE pg\_catalog."default" NOT NULL,  short\_name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  tla character varying(8) COLLATE pg\_catalog."default" NOT NULL,  address character varying(256) COLLATE pg\_catalog."default" NOT NULL,  website character varying(128) COLLATE pg\_catalog."default" NOT NULL,  founded integer NOT NULL,  club\_colors character varying(64) COLLATE pg\_catalog."default" NOT NULL,  CONSTRAINT teams\_pkey PRIMARY KEY (id),  CONSTRAINT area\_id FOREIGN KEY (area\_id)  REFERENCES public.areas (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.token  (  id integer NOT NULL DEFAULT nextval('token\_id\_sequence'::regclass),  user\_id integer,  token character varying(256) COLLATE pg\_catalog."default",  CONSTRAINT token\_pkey PRIMARY KEY (id),  CONSTRAINT token\_user\_id\_fkey FOREIGN KEY (user\_id)  REFERENCES public."user" (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public."user"  (  id integer NOT NULL,  username character varying(32) COLLATE pg\_catalog."default" NOT NULL,  password character varying(256) COLLATE pg\_catalog."default" NOT NULL,  favorite\_team\_id integer,  profile\_image bytea,  phone character varying(16) COLLATE pg\_catalog."default",  join\_date date,  role character varying(32) COLLATE pg\_catalog."default",  CONSTRAINT user\_pkey PRIMARY KEY (id),  CONSTRAINT user\_username\_key UNIQUE (username),  CONSTRAINT favorite\_team\_id FOREIGN KEY (favorite\_team\_id)  REFERENCES public.teams (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.user\_favorite\_players  (  id integer NOT NULL DEFAULT nextval('user\_favorite\_players\_id\_seq'::regclass),  user\_id integer NOT NULL,  player\_id integer NOT NULL,  CONSTRAINT user\_favorite\_players\_pkey PRIMARY KEY (id),  CONSTRAINT user\_favorite\_players\_player\_id\_fkey FOREIGN KEY (player\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION,  CONSTRAINT user\_favorite\_players\_user\_id\_fkey FOREIGN KEY (user\_id)  REFERENCES public."user" (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE RESTRICT  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.user\_favorite\_teams  (  id integer NOT NULL DEFAULT nextval('user\_favorite\_teams\_id\_seq'::regclass),  user\_id integer NOT NULL,  team\_id integer NOT NULL,  CONSTRAINT user\_favorite\_teams\_pkey PRIMARY KEY (id),  CONSTRAINT user\_favorite\_teams\_team\_id\_fkey FOREIGN KEY (team\_id)  REFERENCES public.teams (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT user\_favorite\_teams\_user\_id\_fkey FOREIGN KEY (user\_id)  REFERENCES public."user" (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE CASCADE  )  TABLESPACE goalscoreservice\_default;  CREATE TABLE IF NOT EXISTS public.user\_star\_team  (  id integer NOT NULL DEFAULT nextval('user\_star\_team\_id\_seq'::regclass),  user\_id integer NOT NULL,  goalkeeper\_id integer,  left\_defender\_id integer,  central\_1\_defender\_id integer,  central\_2\_defender\_id integer,  right\_defender\_id integer,  left\_midfielder\_id integer,  central\_midfielder\_id integer,  right\_midfielder\_id integer,  left\_winger\_id integer,  striker\_id integer,  right\_winger\_id integer,  CONSTRAINT user\_star\_team\_pkey PRIMARY KEY (id),  CONSTRAINT user\_star\_team\_user\_id\_unique UNIQUE (user\_id),  CONSTRAINT user\_star\_team\_central\_1\_defender\_id\_fkey FOREIGN KEY (central\_1\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_central\_2\_defender\_id\_fkey FOREIGN KEY (central\_2\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_central\_midfielder\_id\_fkey FOREIGN KEY (central\_midfielder\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_goalkeeper\_id\_fkey FOREIGN KEY (goalkeeper\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_left\_defender\_id\_fkey FOREIGN KEY (left\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_left\_midfielder\_id\_fkey FOREIGN KEY (left\_midfielder\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_left\_winger\_id\_fkey FOREIGN KEY (left\_winger\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_right\_defender\_id\_fkey FOREIGN KEY (right\_defender\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_right\_midfielder\_id\_fkey FOREIGN KEY (right\_midfielder\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_right\_winger\_id\_fkey FOREIGN KEY (right\_winger\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL,  CONSTRAINT user\_star\_team\_striker\_id\_fkey FOREIGN KEY (striker\_id)  REFERENCES public.players (id) MATCH SIMPLE  ON UPDATE CASCADE  ON DELETE SET NULL  )  TABLESPACE goalscoreservice\_default; |

Листинг 3 – Скрипты создания таблиц

|  |
| --- |
| CREATE SEQUENCE IF NOT EXISTS public.user\_id\_sequence  INCREMENT 1  START 16  MINVALUE 16  MAXVALUE 2147483640  CACHE 8  OWNED BY "user".id;  ALTER SEQUENCE public.user\_id\_sequence  OWNER TO postgres;  CREATE SEQUENCE IF NOT EXISTS public.token\_id\_sequence  INCREMENT 1  START 1  MINVALUE 1  MAXVALUE 2147483640  CACHE 8  OWNED BY "token".id;  ALTER SEQUENCE public.token\_id\_sequence  OWNER TO postgres;  CREATE SEQUENCE team\_on\_match\_id\_seq START 1;  ALTER SEQUENCE public.team\_on\_match\_id\_seq  OWNER TO postgres;  CREATE SEQUENCE match\_id\_seq START 1;  ALTER SEQUENCE public.match\_id\_seq  OWNER TO postgres;  CREATE SEQUENCE user\_favorite\_players\_id\_seq START 1;  ALTER SEQUENCE public.user\_favorite\_players\_id\_seq  OWNER TO postgres;  CREATE SEQUENCE user\_favorite\_teams\_id\_seq START 1;  ALTER SEQUENCE public.user\_favorite\_teams\_id\_seq  OWNER TO postgres;    CREATE SEQUENCE user\_star\_team\_id\_seq START 1;  ALTER SEQUENCE public.user\_star\_team\_id\_seq  OWNER TO postgres; |

Листинг 4 – Скрипты создания SEQUENCE

|  |
| --- |
| CREATE OR REPLACE VIEW public.matches\_info\_view AS  SELECT  m.id AS match\_id,  c.name AS competition\_name,  a.name AS area\_name,  m.match\_datetime,  home\_team.name AS home\_team\_name,  home\_team.id AS home\_team\_id,  away\_team.name AS away\_team\_name,  away\_team.id AS away\_team\_id,  m.home\_goal\_score as home\_goal,  m.away\_goal\_score as away\_goal  FROM  public.match m  JOIN public.competitions c ON m.competition\_id = c.id  JOIN public.areas a ON c.area\_id = a.id  JOIN public.team\_on\_match tom\_home ON m.home\_match\_team\_id = tom\_home.id  JOIN public.teams home\_team ON tom\_home.team\_id = home\_team.id  JOIN public.team\_on\_match tom\_away ON m.away\_match\_team\_id = tom\_away.id  JOIN public.teams away\_team ON tom\_away.team\_id = away\_team.id; |

Листинг 5 – Скрипты создание представлений

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_teams\_different()  RETURNS TRIGGER AS $$  DECLARE  home\_team\_id integer;  away\_team\_id integer;  BEGIN  SELECT team\_id INTO home\_team\_id  FROM public.team\_on\_match  WHERE id = NEW.home\_match\_team\_id;  SELECT team\_id INTO away\_team\_id  FROM public.team\_on\_match  WHERE id = NEW.away\_match\_team\_id;  IF home\_team\_id = away\_team\_id THEN  RAISE EXCEPTION 'Home team and away team must be different';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_teams\_different\_trigger  BEFORE INSERT OR UPDATE ON public.match  FOR EACH ROW  EXECUTE FUNCTION check\_teams\_different(); |

Листинг 6 – Скрипты создания триггеров

|  |
| --- |
| CREATE INDEX areas\_id\_btree\_index  ON public.areas USING btree (id)  TABLESPACE goalscoreservice\_default;  CREATE INDEX areas\_name\_hash\_index  ON public.areas USING hash (name)  TABLESPACE goalscoreservice\_default;  CREATE INDEX idx\_teams\_name  ON public.teams (name)  TABLESPACE goalscoreservice\_default;  CREATE INDEX idx\_teams\_id  ON public.teams (id)  TABLESPACE goalscoreservice\_default; |

Листинг 7 – Скрипты индексов