СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Несмотря на свою краткую историю киберспорт быстро обрел поклонников по всему миру. Лучше всего киберспорт развит в Корее, также он очень популярен в Америке и в Европе. Киберспорт начал очень быстро развиваться в Китае, при том, что он там появился совсем недавно. В России же наоборот киберспорт существует давно, но не настолько развит, как хотелось бы. Россия даже стала первой страной в мире, которая признала киберспорт официальным видом спорта 25 июля 2001 года. В июле 2006 г. киберспорт был исключён из Всероссийского реестра видов спорта вследствие того, что он не соответствовал критериям, необходимым для включения в этот реестр. Тем не менее, в связи с тем что киберспортивные дисциплины стали набирать популярность, а так же привлекать большие инвестиции, 7 июня 2016 года был опубликован приказ Министерства Спорта о включении Компьютерного спорта в реестр официальных видов спорта Российской Федерации [1].

Актуальность данной работы в том что киберспорт быстроразвивающийся вид спорта, который набирает популярность по всему миру, именно по этому он привлекает инвестиции, спонсоров и людей.

Цель работы – реализовать базу данных для организации турниров по киберспортивной дисциплине “Dota 2” и их управления.

Для реализации данного проекта, необходимо решить ряд задач:

- формализовать задание, определить необходимый функционал;
- определить роли пользователей;
- проанализировать существующие аналоги;
- проанализировать модели базы данных;
- построить инфологическую модель базы данных;
- спроектировать приложение для доступа к базе данных;
- создать и заполнить базу данных;
- реализовать интерфейс для доступа к базе данных;

- разработать программу, реализующую поставленную задачу.

1 Аналитическая часть

В данном разделе произведена формализация поставленного задания, определен необходимый функционал програмного обеспечения, описаны роли пользователей программы с use-case диаграммой, проанализированы существующие аналоги, приведена ER-диаграмма базы данных, произведен анализ существующих моделей базы данных, а также выбрана конкретная модель для данного проекта.

1.1 Формализация посталенного задания

Поскольку необходимо разработать программу для организации турниров по киберспортивной дисциплине “Dota 2” с возможностью управления ими, то необходимо понять какую информацию необходимо хранить в базе данных. Очевидно, самой важной будет информация о турнирах. Турниры состоят из матчей, матчи проходят между командами, команды состоят из игроков. Кроме того у каждой команды есть спонсор, а у каждого матча – комментатор, который в свою очередь работает на какую-то студию. Также необходимо предусмотреть то, что у каждого турнира имеется организатор, у каждой студии или команды – владелец.

1.2 Необходимый функционал

В соответствии с поставленной задачей, необходимо реализовать следующий функционал:

- создание турниров;
- создание и редактирование матчей;
- добавление свободных игроков в команду;
- удаление игроков из команды;
- добавление свободных комментаторов в студию;
- удаление комментаторов из студии.

1.3 Распределение ролей пользователей

В соответствии с описанным выше функционалом, можно распределить роли описаным ниже способом.

Организатор турниров может:

- создавать турниры;
- выбирать участвующие команды;
- создавать матчи для своего турнира.

Капитан команды может:

- добавлять свободных игроков в команду;
- удалять игроков из своей команды.

Владелец студии может:

- добавлять свободных; комментаторов в свою студию
- удалять комментаторов из своей студии.

Игроки и комментаторы не являются пользователями данного ПО, т.к. оно предназначено именно для организации турниров и их управления, а игроки с комментаторами являются их непосредственными участниками.

Ниже приведена use-case диаграмма.

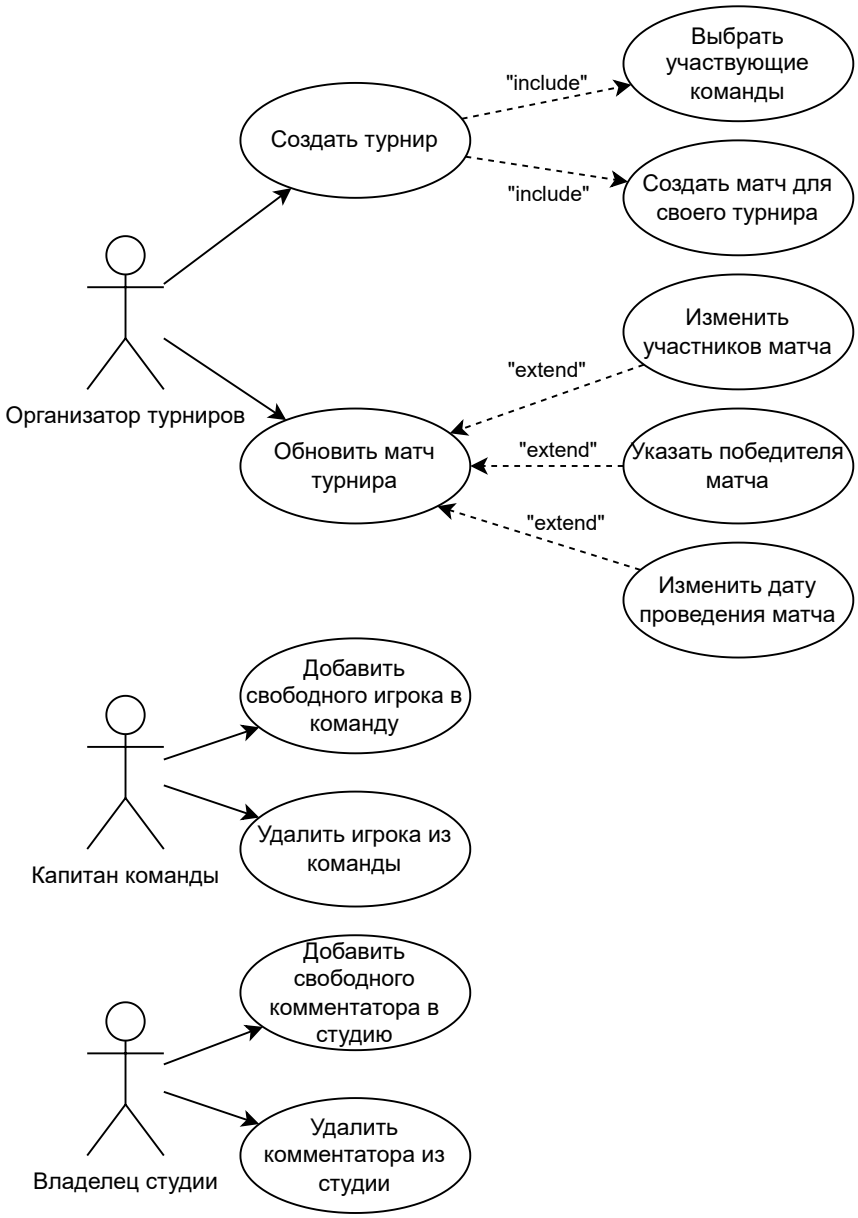

Рисунок 1.1 — use-case диаграмма

Кроме этого, очевидно, необходима роль администратора базы данных, который будет иметь доступ ко всему, чтобы следить за организацией работы базы данных.

1.4 Анализ существующих аналогов

Faceit - это независимая платформа для проведения профессиональных соревнований и любительских игр, которая была основана в 2012 году. Faceit

управляет лигами киберспорта в таких играх как, Counter-Strike: Global Offensive, League of Legends, Rocket League, Dota 2 и PUBG.

Epulze - это платформа для киберспортивных соревнований, где вы можете участвовать в ежедневных автоматических турнирах и соревнованиях, соревнуясь за призы. Данная платформа поддерживает такие дисциплины как Dota 2, CS:GO, Valorant, Mobile Legends и Rocket League.

Challengermode — шведская технологическая компания, основанная в 2014-м году с миссией сделать киберспорт доступным для геймеров. Challengermode строит платформу, которая обеспечивает фундаментальную инфраструктуру онлайн-киберспорта для всех соответствующих заинтересованных сторон в рамках экосистемы. Разработанная технология автоматизирует сложную турнирную инфраструктуру с помощью интеграции игр и позволяет геймерам легко играть в соревнованиях, турнирах и лигах на любом устройстве или консоли, одновременно позволяя создавать сообщества и монетизировать их в масштабе организаторов всех видов.

Ниже в таблице 1.1 приведено сравнение данных платформ.

Таблица 1.1 — Анализ аналогичных решений

Название	Возможность создавать свои турниры	Наличие desktop приложения	Отсутствие платного контента
Faceit	+	+	-
Epulze	-	-	+
Challengermode	+	-	+
Данное приложение	+	+	+

1.5 ER-диаграмма базы данных

Ниже, на рисунке 1.2 приведена ER-диаграмма базы данных. Много много текста.

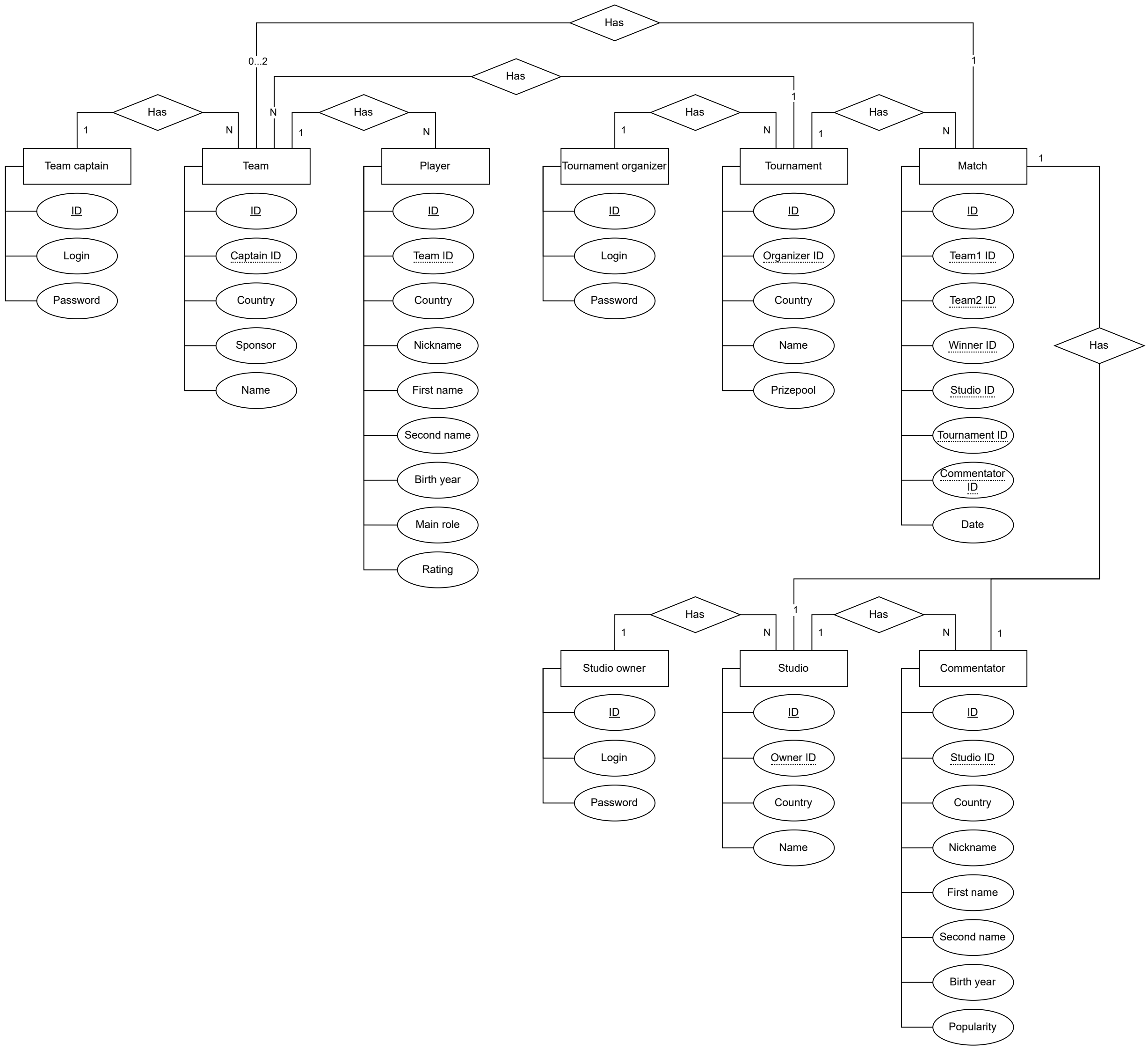

Рисунок 1.2 — ER-диаграмма базы данных

1.6 Выбор модели базы данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы —

поведение данных. Каждая БД и СУБД строится на основе некоторой явной или неявной модели данных [2].

В ходе данной работы были проанализированы следующие модели базы данных

- реляционная;
- иерархическая;
- сетевая.

1.6.1 Реляционная модель базы данных

Реляционная модель данных является совокупностью данных и состоит из набора двумерных таблиц. При табличной организации отсутствует иерархия элементов. Таблицы состоят из строк – записей и столбцов – полей. На пересечении строк и столбцов находятся конкретные значения. Для каждого поля определяется множество его значений. За счет возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов.

Реляционная модель является удобной и наиболее широко используемой формой представления данных.

1.6.2 Иерархическая модель базы данных

В иерархической модели данных используется представление базы данных в виде древовидной структуры, состоящей из объектов различных уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка к потомку, при этом возможна ситуация, когда объект-предок имеет несколько потомков, тогда как у объекта-потомка обязателен только один предок.

1.6.3 Сетевая модель базы данных

В сетевой модели данных, в отличии от иерархической, у потомка может иметься любое число предков. Сетевая база данных состоит из набора экземпляров определенного типа записи и набора экземпляров определенного типа связей между этими записями.

Главным недостатком сетевой модели данных являются жесткость и высокая сложность схемы базы данных, построенной на основе этой модели. Так как логика процедуры выбора данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения. Иначе говоря, если будет необходимо изменить структуру данных, то нужно будет изменять и приложение.

Вывод из аналитической части

Поскольку реляционная модель базы данных является наиболее широко используемой и удобной, а также имеет возможность изменения базы данных без глобальных изменений программного обеспечения, то для реализации данного проекта была выброана именно она.

В данном разделе была произведена формализация поставленной задачи, описан необходимый функционал с распределением по ролям, приведена use-case диаграмма, проанализированы существующие аналоги, приведена ER-диаграмма базы данных, произведен анализ модели базы данных и выбрана реляционная модель.

2 Конструкторская часть

В данном разделе приведены таблицы, инфологическая модель и схемы триггеров базы данных. Много много текста.

2.1 Таблицы базы данных

Исходя из приведенных выше use-case и ER диаграмм можно выделить следующие таблицы:

- таблица ролей пользователей;
- таблица пользователей;
- таблица стран;
- таблица спонсоров;
- таблица команд;
- табилца игроков;
- таблица студий;
- таблица комментаторов;
- таблица турниров;
- таблица турниров и участвующих в них команд;
- таблица матчей.

2.2 Инфологическая модель базы данных

Ниже, на рисунке 2.1 приведена инфологическая модель базы данных, определяющая связи между полями будущих таблиц.

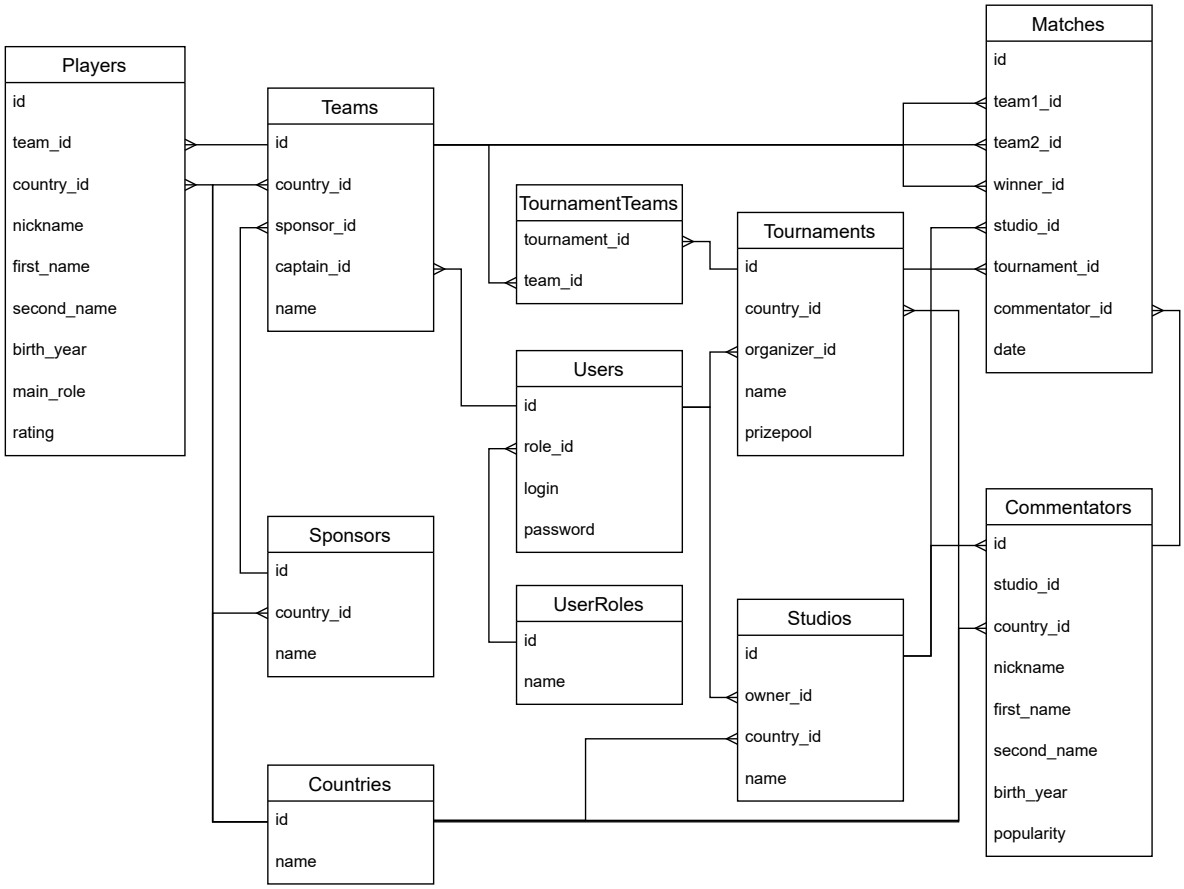

Рисунок 2.1 — Инфологическая модель базы данных

2.3 Схемы триггеров

На рисунке 2.2 изображена схема триггера который срабатывает при удаление пользователя с ролью “Team captain”.

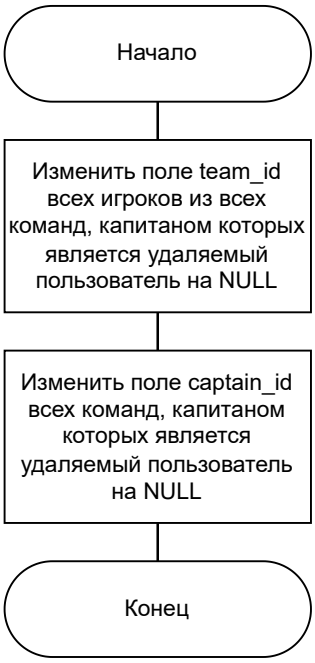

Рисунок 2.2 — схема триггера $tr\_whenDeleteTeamCaptain$

На рисунке 2.3 изображена схема триггера который срабатывает при удаление пользователя с ролью “Studio owner”.

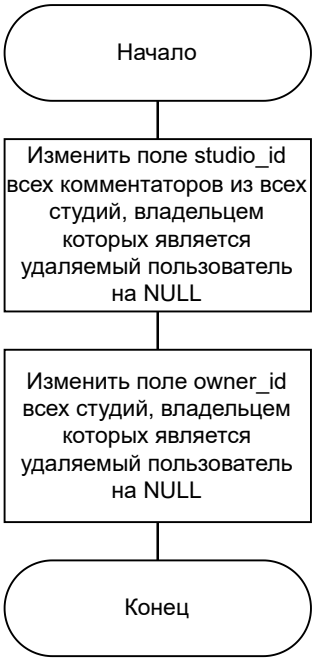

Рисунок 2.3 — схема триггера $tr\_whenDeleteStudioOwner$

На рисунке 2.4 изображена схема триггера который срабатывает при удаление пользователя с ролью “Tournament organizer”.

Рисунок 2.4 — схема триггера $tr\_whenDeleteTournamentOrganizer$

Вывод из конструкторской части

В данном разделе были выделены таблицы, разработна инфологическая модель и схемы триггеров базы данных.

3 Технологическая часть

В данном разделе приведен код. Много много текста.

Код создания таблиц смотри в Приложении А.

4 Исследовательская часть

В данном разделе проведено исследование

ЗАКЛЮЧЕНИЕ

По итогу проделанной работы была достигнута цель – разработана база данных для организации турниров по киберспортивной дисциплине “Dota 2” и их управления.

Также были решены все поставленные задачи, а именно:

- формализовно задание, определен необходимый функционал;
- определены роли пользователей;
- проанализированы существующие аналоги;
- проанализированы модели базы данных;
- построена инфологическая модель базы данных;
- спроектировано приложение для доступа к базе данных;
- создана и заполнена база данных;
- реализован интерфейс для доступа к базе данных;
- разработана программа, реализующая поставленную задачу.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный интернет-портал правовой информации [Электронный ресурс]: приказ Министерства Спорта о включении Компьютерного спорта в реестр официальных видов спорта Российской Федерации. URL: http://publication.pravo.gov.ru/Document/View/0001201606070022?index=0&rangeSize=1 (дата обращения: 12.05.21)

2. Дейт К. Дж. Введение в системы баз данных. — 8-е изд. — М.: «Вильямс», 2006. (дата обращения: 12.05.21)

ПРИЛОЖЕНИЕ А

Листинг 1 — создание таблиц, часть 1

```
create table if not exists UserRoles
(
    id serial primary key,
    name text
);

create table if not exists Users
(
    id serial primary key,
    role_id int,
    FOREIGN KEY (role_id) REFERENCES public.UserRoles (id),
    login text,
    password text
);

create table if not exists Countries
(
    id serial primary key,
    name text
);

create table if not exists Sponsors
(
    id serial primary key,
    country_id int,
    FOREIGN KEY (country_id) REFERENCES public.Countries (id),
    name text
);
create table if not exists Teams
(
    id serial primary key,
    country_id int,
    sponsor_id int,
    captain_id int,
    FOREIGN KEY (country_id) REFERENCES public.Countries (id),
    FOREIGN KEY (sponsor_id) REFERENCES public.Sponsors (id),
    FOREIGN KEY (captain_id) REFERENCES public.Users (id),
    name text
);
```

Листинг 2 — создание таблиц, часть 2

```
create table if not exists Players
(
    id serial primary key,
    team_id int,
    country_id int,
    FOREIGN KEY (team_id) REFERENCES public.Teams (id),
    FOREIGN KEY (country_id) REFERENCES public.Countries (id),
    nickname text,
    first_name text,
    second_name text,
    birth_year int,
    main_role text,
    rating int
);

create table if not exists Studios
(
    id serial primary key,
    country_id int,
    owner_id int,
    FOREIGN KEY (country_id) REFERENCES public.Countries (id),
    FOREIGN KEY (owner_id) REFERENCES public.Users (id),
    name text
);

create table if not exists Commentators
(
    id serial primary key,
    studio_id int,
    country_id int,
    FOREIGN KEY (studio_id) REFERENCES public.Studios (id),
    FOREIGN KEY (country_id) REFERENCES public.Countries (id),
    nickname text,
    first_name text,
    second_name text,
    birth_year int,
    popularity int
);
```

Листинг 3 — создание таблиц, часть 3

```
create table if not exists Tournaments
(
    id serial primary key,
    country_id int,
    organizer_id int,
    FOREIGN KEY (country_id) REFERENCES public.Countries (id),
    FOREIGN KEY (organizer_id) REFERENCES public.Users (id),
    name text,
    prizepool int
);

create table if not exists Matches
(
    id serial primary key,
    team1_id int,
    team2_id int,
    winner_id int,
    studio_id int,
    commentator_id int,
    tournament_id int,
    FOREIGN KEY (team1_id) REFERENCES public.Teams (id),
    FOREIGN KEY (team2_id) REFERENCES public.Teams (id),
    FOREIGN KEY (winner_id) REFERENCES public.Teams (id),
    FOREIGN KEY (studio_id) REFERENCES public.Studios (id),
    FOREIGN KEY (commentator_id) REFERENCES public.Commentators (id),
    FOREIGN KEY (tournament_id) REFERENCES public.Tournaments (id),
    date date
);

create table if not exists TournamentTeams
(
    tournament_id int,
    team_id int,
    FOREIGN KEY (tournament_id) REFERENCES public.Tournaments (id),
    FOREIGN KEY (team_id) REFERENCES public.Teams (id)
);
```

Листинг 4 — копирование сгенерированных таблиц в базу данных

```
copy UserRoles from '/UserRoles.csv' delimiter '|' csv;

copy Users(role_id, login, password) from '/Users.csv' delimiter '|' csv;

copy Countries from '/Countries.csv' delimiter '|' csv;

copy Sponsors(country_id, name) from '/Sponsors.csv' delimiter '|' csv;

copy Teams(country_id, sponsor_id, captain_id, name) from '/Teams.csv'
    delimiter '|' csv;

copy Players(team_id, country_id, nickname, first_name, second_name,
    birth_year, main_role, rating) from '/Players.csv' delimiter '|' csv;

copy Studios(country_id, owner_id, name) from '/Studios.csv' delimiter '|' csv;

copy Commentators(studio_id, country_id, nickname, first_name, second_name,
    birth_year, popularity) from '/Commentators.csv' delimiter '|' csv;

copy Tournaments(country_id, organizer_id, name, prizepool) from
    '/Tournaments.csv' delimiter '|' csv;

copy Matches(team1_id, team2_id, winner_id, studio_id, commentator_id,
    tournament_id, date) from '/Matches.csv' delimiter '|' csv;

copy TournamentTeams(tournament_id, team_id) from '/TournamentTeams.csv'
    delimiter '|' csv;
```

Листинг 5 — создание views, часть 1

```
create or replace VIEW players_view AS
SELECT p.id, t.name as team, c.name as country, p.nickname, p.first_name,
    p.second_name, p.birth_year, p.main_role, p.rating, p.team_id, p.country_id
FROM players p
    LEFT OUTER JOIN teams t ON t.id = p.team_id
    LEFT OUTER JOIN countries c ON c.id = p.country_id
order by p.id;

create or replace VIEW players_view_by_teams AS
SELECT p.id, t.name as team, c.name as country, p.nickname, p.first_name,
    p.second_name, p.birth_year, p.main_role, p.rating, p.team_id, p.country_id,
    t.captain_id
FROM players p
    LEFT OUTER JOIN teams t ON t.id = p.team_id
    LEFT OUTER JOIN countries c ON c.id = p.country_id
order by t.name, p.id;

create or replace VIEW teams_avg as
select t.id, avg(p.rating)
from teams t
    join players p on p.team_id = t.id
group by t.id
order by t.id;

create or replace VIEW teams_view AS
SELECT t.id, c.name as country, s.name as sponsor, u.login as captain, t.name,
    ta.avg as avg_rating, t.country_id, t.sponsor_id, t.captain_id
FROM teams t
    LEFT OUTER JOIN countries c  on c.id = t.country_id
    LEFT OUTER JOIN sponsors s   on s.id = t.sponsor_id
    LEFT OUTER JOIN users u      on u.id = t.captain_id
    LEFT OUTER JOIN teams_avg ta on ta.id = t.id
order by t.id;

create or replace VIEW commentators_view AS
SELECT c.id, s.name as studio, coun.name as country, c.nickname, c.first_name,
    c.second_name, c.birth_year, c.popularity, c.studio_id, c.country_id,
    s.owner_id
FROM commentators c
    LEFT OUTER JOIN studios s ON s.id = c.studio_id
    LEFT OUTER JOIN countries coun ON coun.id = c.country_id
order by c.id;
```

Листинг 6 — создание views, часть 2

```
create or replace VIEW studios_avg as
select s.id, avg(c.popularity)
from studios s
    join commentators c on c.studio_id = s.id
group by s.id
order by s.id;
create or replace VIEW studios_view AS
SELECT s.id, c.name as country, u.login as owner, s.name, sa.avg as
    avg_popularity, s.country_id, s.owner_id
FROM studios s
    LEFT OUTER JOIN countries c  on c.id = s.country_id
    LEFT OUTER JOIN users u      on u.id = s.owner_id
    LEFT OUTER JOIN studios_avg sa on sa.id = s.id
order by s.id;
create or replace VIEW tournaments_view AS
SELECT t.id, c.name as country, u.login as organizer, t.name, t.prizepool,
    t.country_id, t.organizer_id
FROM tournaments t
    LEFT OUTER JOIN countries c  on c.id = t.country_id
    LEFT OUTER JOIN users u      on u.id = t.organizer_id
order by t.id;
create or replace VIEW matches_view AS
SELECT m.id, t1.name as team1, t2.name as team2, w.name as winner, s.name as
    studio, c.nickname as commentator, t.name as tournament, m.date,
        m.team1_id, m.team2_id, m.winner_id, m.studio_id, m.commentator_id,
            m.tournament_id
FROM matches m
    LEFT OUTER JOIN teams t1 on m.team1_id = t1.id
    LEFT OUTER JOIN teams t2 on m.team2_id = t2.id
    LEFT OUTER JOIN teams w  on m.winner_id = w.id
    LEFT OUTER JOIN studios s  on m.studio_id = s.id
    LEFT OUTER JOIN commentators c  on m.commentator_id = c.id
    LEFT OUTER JOIN tournaments t  on m.tournament_id = t.id
order by m.date;
create or replace VIEW tournament_teams_view AS
SELECT t.id, c.name as country, s.name as sponsor, u.login as captain, t.name,
    ta.avg as avg_rating, t.country_id, t.sponsor_id, t.captain_id,
    tt.tournament_id
FROM tournamentteams tt
    LEFT OUTER JOIN teams t on tt.team_id = t.id
    LEFT OUTER JOIN countries c  on c.id = t.country_id
    LEFT OUTER JOIN sponsors s   on s.id = t.sponsor_id
    LEFT OUTER JOIN users u      on u.id = t.captain_id
    LEFT OUTER JOIN teams_avg ta on ta.id = t.id
order by tt.tournament_id;
```

Листинг 7 — создание триггеров, часть 1

```
create or replace function whenDeleteTeamCaptain()
returns trigger
as $$
BEGIN
    UPDATE players
    SET team_id = NULL
    WHERE team_id IN
        (SELECT id AS team_id
         FROM teams t
         WHERE t.captain_id = OLD.id);

    UPDATE teams
    set captain_id = null
    where captain_id = old.id;
    return OLD;
end
$$ language plpgsql;

create or replace trigger tr_whenDeleteTeamCaptain
before delete
on Users
for each row
when (OLD.role_id = 4)
execute procedure whenDeleteTeamCaptain();
```

Листинг 8 — создание триггеров, часть 2

```
create or replace function whenDeleteStudioOwner()
returns trigger
as $$
BEGIN
    UPDATE commentators
    SET studio_id = NULL
    WHERE studio_id IN
        (SELECT id AS studio_id
         FROM studios s
         WHERE s.owner_id = OLD.id);

    UPDATE studios
    set owner_id = null
    where owner_id = old.id;
    return OLD;
end
$$ language plpgsql;

create or replace trigger tr_whenDeleteStudioOwner
before delete
on Users
for each row
when (OLD.role_id = 3)
execute procedure whenDeleteStudioOwner();

create or replace function whenDeleteTournamentOrganizer()
returns trigger
as $$
BEGIN
    UPDATE tournaments
    SET organizer_id = NULL
    WHERE organizer_id = old.id;

    return OLD;
end
$$ language plpgsql;

create or replace trigger tr_whenDeleteTournamentOrganizer
before delete
on Users
for each row
when (OLD.role_id = 2)
execute procedure whenDeleteTournamentOrganizer();
```

Листинг 9 — создание ролей

```
--guest
CREATE USER guest password 'guest';
--granting access rights to guest
grant pg_read_all_data to guest;
grant insert on users to guest;
grant update on users_id_seq to guest;

--team_captain
CREATE USER team_captain password 'team_captain';
--granting access rights to team_captain
grant postgres to team_captain;

--studio_owner
CREATE USER studio_owner password 'studio_owner';
--granting access rights to studio_owner
grant postgres to studio_owner;

--tournament_organizer
CREATE USER tournament_organizer password 'tournament_organizer';
--granting access rights to tournament_organizer
grant postgres to tournament_organizer;
```