T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

SAKARYA

ÜNİVERSİTESİ

<< Aralık – 2023 >>

Ders: **VERİTABANI YÖNETİM SİSTEMLERİ**

Grub: 1. Öğretim A Grub

Hazırlayan: Muhammed İyd (B211210569).

Ders Sorumlusu: Prof. Dr. Celal ÇEKEN ve Arş.Gör. Mustafa Alp Eren Kılıç.

E-Posta: muhamad.iyd1@ogr.sakarya.edu.tr

GitHub: https://github.com/muhamm7d.eid/VTYS-Proje

# Senaryo :

"Futbol Ligi Sezonu" adlı senaryo, bir << futbol ligi yönetim sistemini temsil eder >>.

Ligdeki takımların oluşturulmasıyla başlayan senaryo,

her bir takımın oyuncularını transfer etmesiyle devam eder.

Ligdeki maçlar düzenlenir ve bu maçlara hakem atamaları yapılır.

Maçlar sırasında kart ve gol kayıtları tutularak,

oyuncuların performansları izlenir.

Her olayın ardından, puan durumu güncellenir ve

ligdeki takımlar arasındaki sıralama belirlenir.

Lig sezonu boyunca oyunculara ödüller verilebilir,

transferler gerçekleşebilir ve takımların yetenekleri değerlendirilebilir.

Bu bilgiler, veritabanındaki ilgili tablolar üzerinden tutulur ve yönetilir.

Senaryo, futbol ligi yönetimi için gerekli olan temel işlemleri içerir

ve veritabanındaki tablolar aracılığıyla bu bilgilerin tutulmasını sağlar.

Ligdeki her takım, oyuncu, maç,

hakem ve diğer katılımcıların verileri düzenli bir şekilde saklanarak,

ligin başarılı bir şekilde yönetilmesine imkan tanır.

# İş Kuralları :

- Bir Oyuncu yalnızca bir takıma bağlı olabilir.
- Bir maça en az bir hakem atanmalıdır.
- Bir hakem aynı anda birden fazla maça atanamaz.
- Bir oyuncu bir maçta aynı dakikada birden fazla kart alamaz.
- Bir oyuncu bir maçta aynı dakikada birden fazla gol atamaz.
- Gol atan oyuncu ve maç bilgileri geçerli olmalıdır.
- Bir oyuncu aynı transfer pencersinde birden fazla transfer yapmaz.
- Bir oyuncunun transfer tarihi, geçerli bir tarih olmalıdır.
- Bir oyuncu, aynı ödülü birden fazla kez alamaz.
- Ödül tarihi geçerli bir tarih olmalıdır.
- Bir takımların ofansif, orta saha, defansif yetenklerini temsil eden bir tablo oluşturmalısınız.
- Antrenmanlar sadece var olan takımlara atttrib edilmeli.
- Bir maçtaki hakem ve stadium farklı olmalı.

• Bir maçtaki ev sahibi ve misafir takımlar farklı olamlı.

• Her bir koç bir takıma bağlı olmalı, ve koçları temsil eden bir tablo oluşturulmalıdır.

• Her takım birden fazla oyuncu içerebilir.

• Oyuncular bir takıma bağlı olmalı.

• Maçlar, ev sahibi ve deplasman takımlarına bağlı olamlı.

• Hakemler belirli maçlar atanmalı.

• Kartlar veg oller belirli maç ve oyuncularla ilişkilendirilmeli.

• Transferler, oyuncunun eski ve yeni takımları arasında bağlantılı olmalı.

• Lig maçları belirli bir lige atanmalı. Her takım belirli bir lige atanmalı.

• Hakem atamaları, belirli bir lige atanmalı.  ve atanan hakemleri içerçelidir.

• Bir oyuncu aynı transfer penceresinde birden fazla transfer yapmaz.

• Yedek oyuncular, belirli bir maç ve oyuncuyla ilişlendirilmelidir.

• Transfer, eski ve yani takımlar arasında bağlantılı olmalıdır,
böylece bir oyuncunun tranferi izlenebilir.

• Oyuncular, belirli bir maçta hangi koçun yönetiminde olduğlyla ilişkilendirilmelidir.
• Takımların farklı sezon ve ligleri bağlı olarak sıralama tablorına bağlı olmasına sağlamalısı.

• Her oyuncu, kazandığı ödüzlere bağlı olmalı ve bunlar ödül tablosuna kaydedilmelidir.

# İlişkisel Şema:

• Teams(TeamID: SERIAL PRIMARY KEY, Name: VARCHAR(50), City: VARCHAR(50),

Est_year: DATE).

• Person(PersonID: SERIAL PRIMARY KEY, FirstName: VARCHAR(50) NOT NULL,

LastName: VARCHAR(50) NOT NULL, BirthDate: DATE).

• Peron.Players(PersonID: INTEGER, TeamID: INTEGER, Positions: VARCHAR(50),

NationalTeam: VARCHAR(50),

<<PRIMARY KEY (PersonID), FOREIGN KEY (PersonID) REFERENCES Person(PersonID),

FOREIGN KEY (TeamID) REFERENCES Teams(TeamID))>>.

• Coaches(PersonID: INTEGER, TeamID: INTEGER,

<<PRIMARY KEY (PersonID), FOREIGN KEY (PersonID) REFERENCES Person(PersonID),

FOREIGN KEY (TeamID) REFERENCES Teams(TeamID))>>.

• Referees(PersonID: INTEGER, Country: VARCHAR(50),

<<PRIMARY KEY (PersonID), FOREIGN KEY (PersonID) REFERENCES Person(PersonID))>>.

• Leagues(LeagueID: SERIAL PRIMARY KEY, LeagueName: VARCHAR(50), Season: VARCHAR(50),
County: VARCHAR(50), TeamID: INTEGER,

<<FOREIGN KEY (TeamID) REFERENCES Teams(TeamID))>>.

• Standings(StandingsID: SERIAL PRIMARY KEY, TeamID: INTEGER, Wins: INTEGER, Losses: INTEGER,
Draws: INTEGER, <<FOREIGN KEY (TeamID) REFERENCES Teams(TeamID))>>.

• Awards(AwardID: SERIAL PRIMARY KEY, AwardName: VARCHAR(50), PersonID: INTEGER,

<<FOREIGN KEY (PersonID) REFERENCES Person(PersonID))>>.

• Transfers(TransferID: SERIAL PRIMARY KEY, PersonID: INTEGER, OldTeam: VARCHAR(50),

NewTeam: VARCHAR(50), TransferDate: DATE,

<<FOREIGN KEY (PersonID) REFERENCES Person(PersonID))>>.

• TeamAbilities(TeamAbilitiesID: SERIAL PRIMARY KEY, Attack: VARCHAR(50), Mid: VARCHAR(50), Defensive: VARCHAR(50), TeamID: INTEGER,

<<FOREIGN KEY (TeamID) REFERENCES Teams(TeamID))>>.

• Stadium(StadiumID: SERIAL PRIMARY KEY, StadiumName: VARCHAR(50), TeamID: INTEGER, <<FOREIGN KEY (TeamID) REFERENCES Teams(TeamID))>>.

• Matches(MatchID: SERIAL PRIMARY KEY, HomeTeam: VARCHAR(50), AwayTeam: VARCHAR(50), Score: INTEGER, MatchDate: DATE, StadiumID: INTEGER,

<<FOREIGN KEY (StadiumID) REFERENCES Stadium(StadiumID))>>.

• LeagueMatches(LeagueID: INTEGER, MatchID: INTEGER,

<<PRIMARY KEY (LeagueID, MatchID), FOREIGN KEY (LeagueID) REFERENCES Leagues(LeagueID), FOREIGN KEY (MatchID) REFERENCES Matches(MatchID))>>.

• Goals(GoalID: SERIAL PRIMARY KEY, GoalNumbers: INTEGER).

• TeamsGoals(GoalID: INTEGER, TeamID: INTEGER,

<<PRIMARY KEY (GoalID, TeamID), FOREIGN KEY (GoalID) REFERENCES Goals(GoalID), FOREIGN KEY (TeamID) REFERENCES Teams(TeamID))>>.

# • (Functions):

/// bu function awards(Ödülü) getirir ////

```
CREATE OR REPLACE FUNCTION get_awards_for_player(p_personID INTEGER)
RETURNS TABLE (
   awardID INTEGER,
   awardName VARCHAR(50)
) AS $$
BEGIN
   RETURN QUERY
   SELECT
      "awardID",
      "awardName"
   FROM
      "awards"
   WHERE
      "personID" = p_personID;
END;
$$ LANGUAGE plpgsql;
```

```
-- get_awards_for_player işlevine örnek çağrı:
SELECT * FROM get_awards_for_player(1);
```

==========
==========

/// bu function playerin awards(Ödülü) siler ////

```
CREATE OR REPLACE FUNCTION delete_award_for_player(
    p_personID INTEGER,
    p_awardID INTEGER
)

RETURNS VOID AS $$
BEGIN
    DELETE FROM "awards"
    WHERE "personID" = p_personID AND "awardID" = p_awardID;

END;
$$ LANGUAGE plpgsql;
```

```
-- delete_award_for_player function işlevine örnek çağrı:
SELECT delete_award_for_player(1, 5);
```

==========
==========

/// bu function player getirir ////

```
CREATE OR REPLACE FUNCTION get_player(p_personID INTEGER)
RETURNS TABLE (
   personID INTEGER,
   firstName VARCHAR(50),
   lastName VARCHAR(50),
   birthDate DATE,
   positions VARCHAR(50),
   nationalTeam VARCHAR(50),
   teamID INTEGER
) AS $$BEGIN  RETURN QUERY
   SELECT
      p."personID",
      p."firstName",
      p."lastName",
      p."birthDate",
      pl."positions",
      pl."nationalTeam",
      pl."teamID"
   FROM
      "person"."person" p
   JOIN
      "person"."players" pl ON p."personID" = pl."personID"
   WHERE
      p."personID" = p_personID;
END;
$$ LANGUAGE plpgsql;
```

```
-- get_player işlevine örnek çağrı:
SELECT * FROM get_player(1);
```

==========
==========

/// bu function playerin positions'ine update yapar ////

```
CREATE OR REPLACE FUNCTION update_player_positions(
    p_personID INTEGER,
    p_positions VARCHAR(50)
)

RETURNS VOID AS $$
BEGIN
    UPDATE "person"."players"
    SET "positions" = p_positions
    WHERE "personID" = p_personID;

END;
$$ LANGUAGE plpgsql;
```

```
--  update_player_positions işlevine örnek çağrı:
SELECT update_player_positions(2, 'Defender');
```

==========
==========

# • Triggers:

/// Bir tablo oluşturarak stadyumu yeni bir adla adlandırır ///

```
-- to ensure stadium name is
CREATE OR REPLACE FUNCTION uppercase_stadium_name()
RETURNS TRIGGER AS $$
BEGIN
  NEW."stadiumName" := UPPER(NEW."stadiumName");
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger to activate the function before an INSERT or UPDATE operation on the "stadium" table
CREATE TRIGGER before_insert_update_stadium
BEFORE INSERT OR UPDATE ON "stadium"
FOR EACH ROW EXECUTE FUNCTION uppercase_stadium_name();
```

-- to ensure stadium name is:

```
INSERT INTO "stadium" ("stadiumName", "teamID") VALUES ('yeni stadium', 1);
```

==========

==========

//// bu trigger goalNumbers destirir yerini yeni bir number ekleyebiliriz ////

```
-- Trigger to ensure that goalNumbers is always greater than or equal to 0
CREATE OR REPLACE FUNCTION validate_goal_numbers()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW."goalNumbers" < 0 THEN
    RAISE EXCEPTION 'Goal numbers must be greater than or equal to 0';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger to activate the function before an INSERT or UPDATE operation on the "goals" table
CREATE TRIGGER before_insert_update_goals
BEFORE INSERT OR UPDATE ON "goals"
FOR EACH ROW EXECUTE FUNCTION validate_goal_numbers();


-- This will trigger the validation trigger:
INSERT INTO "goals" ("goalNumbers") VALUES (17);
```

==========

==========

///// Kişinin bilgilerine gider, adını değiştirir ve soyadını birakir. ///////

```
CREATE OR REPLACE FUNCTION set_default_birthdate_on_update()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW."birthDate" IS NULL THEN
    NEW."birthDate" := CURRENT_DATE;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_set_default_birthdate_on_update
BEFORE UPDATE ON "person"."person"
FOR EACH ROW
EXECUTE FUNCTION set_default_birthdate_on_update();
```

```
-- Assume personID is the ID of the person you want to update
UPDATE "person"."person"
SET "firstName" = 'UpdatedFirstName'
WHERE "personID" = 1;  -- Replace 1 with the actual personID
```

==========

==========

//// EĞer (birthdate NULL) ise >> CURRENT_DATE ekler. ////

```
CREATE OR REPLACE FUNCTION set_default_birthdate()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW."birthDate" IS NULL THEN
    NEW."birthDate" := CURRENT_DATE;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_set_default_birthdate
BEFORE INSERT ON "person"."person"
FOR EACH ROW
EXECUTE FUNCTION set_default_birthdate();
```

```
-- Insert a new person without specifying the birthdate:
INSERT INTO "person"."person" ("firstName", "lastName")
VALUES ('John', 'merhabba');
```

==========

==========

//// Bu trigger, bir "person" tablosuna yeni bir satır eklerken, eğer "birthDate" alanı belirtilmemişse (NULL ise), bu alanı özel bir tarih olan 'restgle' olarak ayarlar. Yani,\eğer bir kişi eklenirken doğum tarihi belirtilmemiş ise, bu trigger otomatik olarak doğum tarihini 'restgle' olarak atar. ////

```
CREATE OR REPLACE FUNCTION set_default_birthdate_specific()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW."birthDate" IS NULL THEN
    NEW."birthDate" := '2000-01-01'::DATE;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_set_default_birthdate_specific
BEFORE INSERT ON "person"."person"
FOR EACH ROW
EXECUTE FUNCTION set_default_birthdate_specific();
```

```
-- Insert a new person without specifying the birthdate:
INSERT INTO "person"."person" ("firstName", "lastName")
VALUES ('Hamoud', 'Hasan');

-- Query the inserted record to verify the birthDate
SELECT * FROM "person"."person";
```

==========

==========

# • Kalitim:

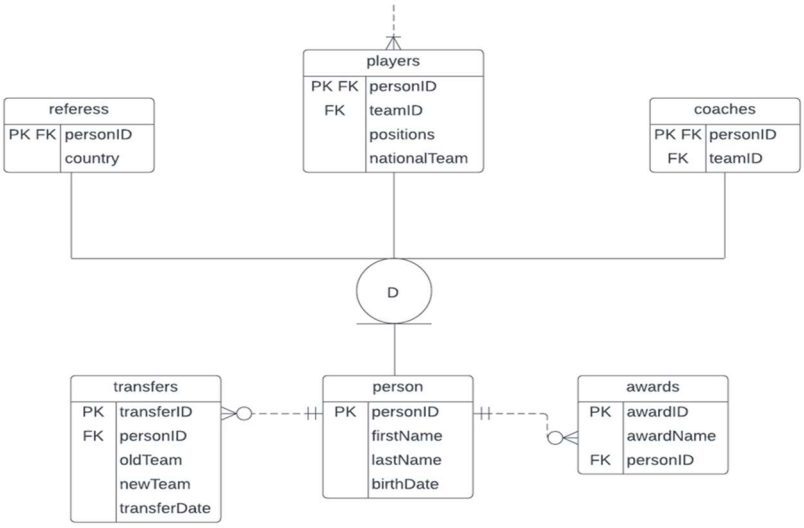
-- Kisi Tablosu (Üst Tablo)

//person; player olabilir, coache, referee//

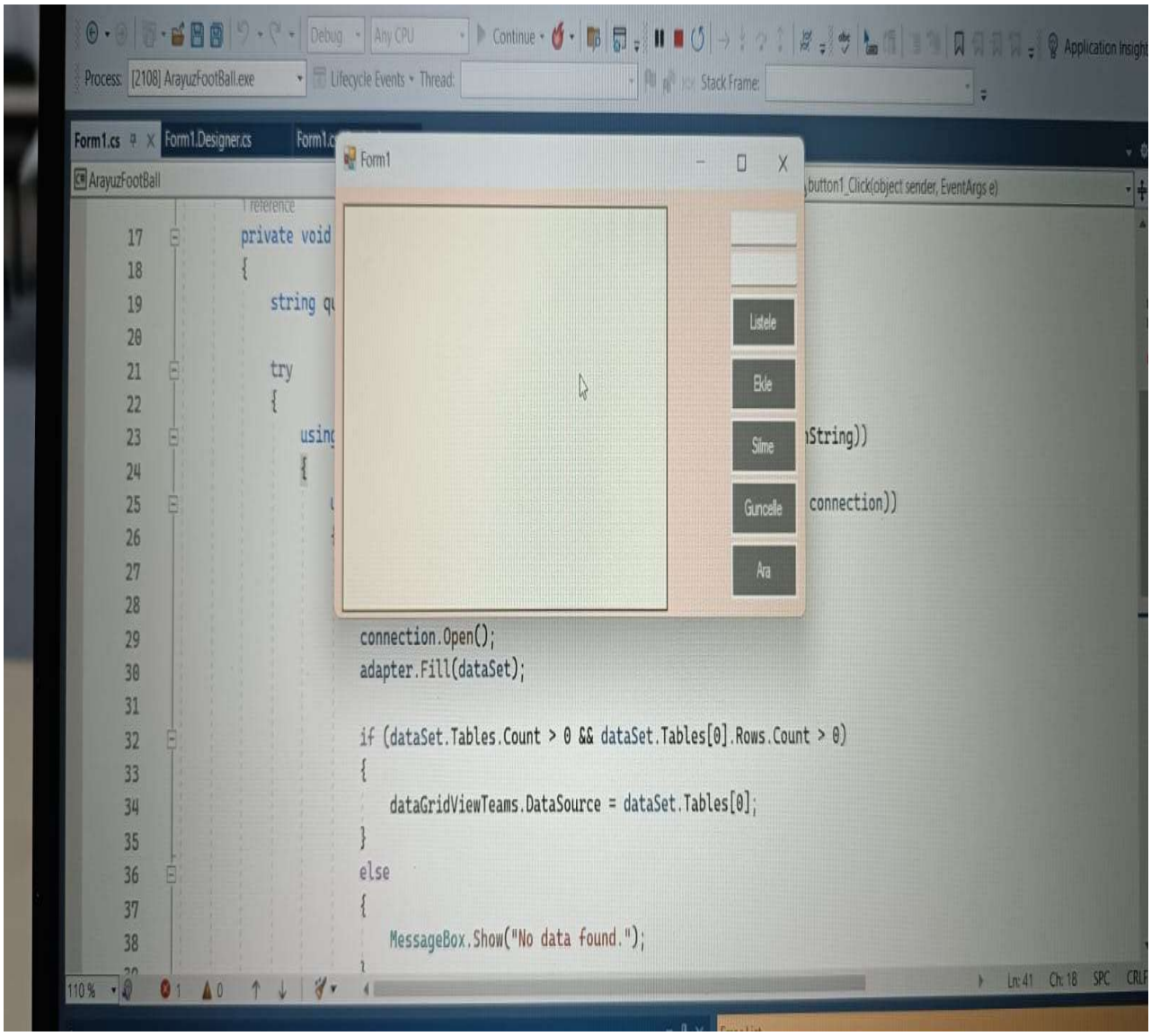
create schema person;

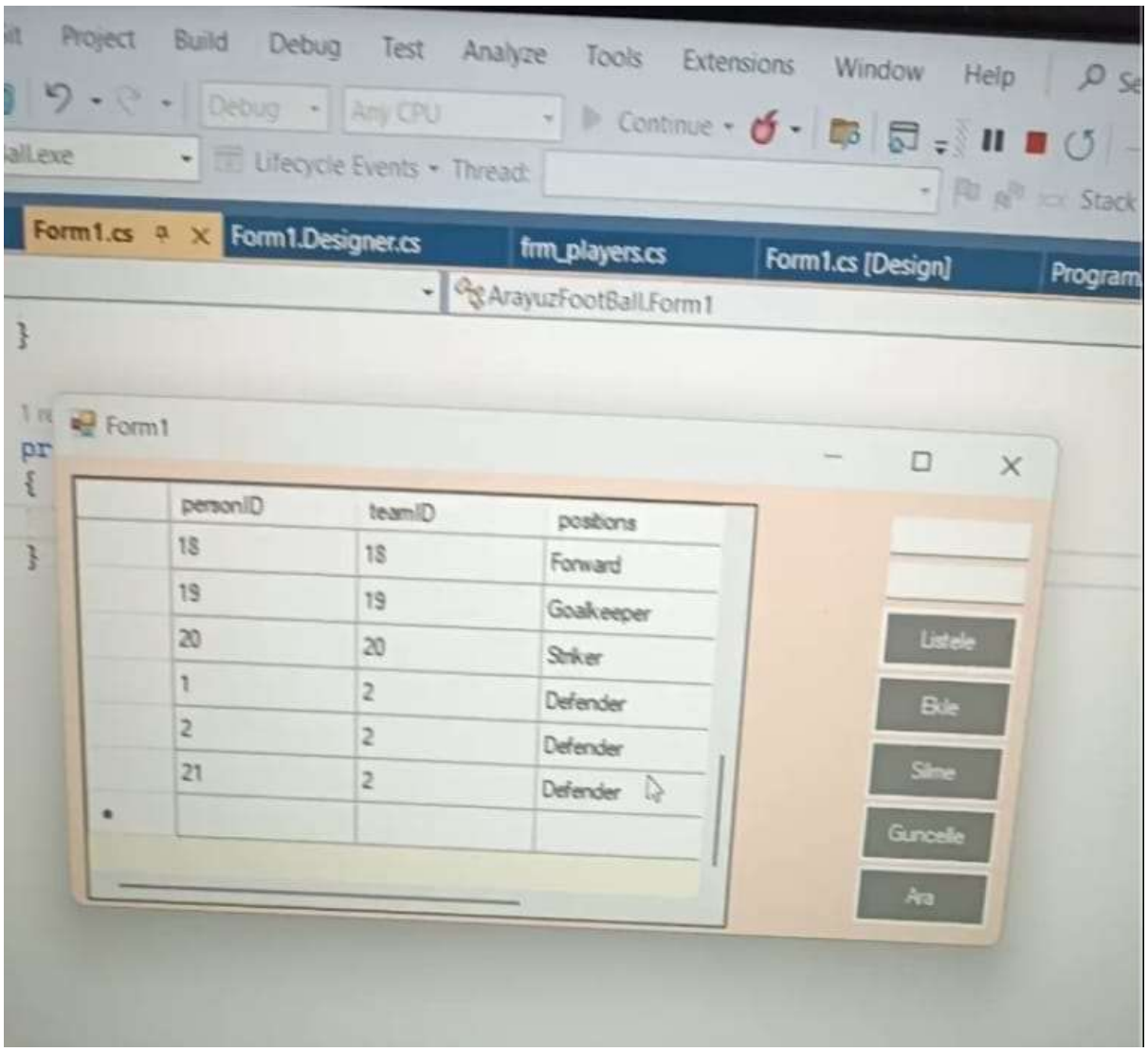
```
CREATE TABLE "person"."person" (
        "personID" SERIAL,
        "firstName" VARCHAR(50) NOT NULL,
        "lastName" VARCHAR(50) NOT NULL,
        "birthDate" DATE,
        CONSTRAINT "personPK" PRIMARY KEY ("personID")
);


CREATE TABLE "person"."players" (
   "personID" INTEGER,
   "teamID" INTEGER,
   "positions" VARCHAR(50),
   "nationalTeam" VARCHAR(50),
   CONSTRAINT "playersPK" PRIMARY KEY ("personID"),
   CONSTRAINT "teamFK" FOREIGN KEY ("teamID") REFERENCES
"teams"("teamID"));
```

```
CREATE TABLE "person"."coaches" (
    "personID" INTEGER,
    "teamID" INTEGER,
    CONSTRAINT "coachesPK" PRIMARY KEY ("personID"),
    CONSTRAINT "teamFK" FOREIGN KEY ("teamID") REFERENCES "teams"("teamID")
);

CREATE TABLE "person"."referees" (
    "personID" INTEGER,
    "country" VARCHAR(50),
    CONSTRAINT "refereesPK" PRIMARY KEY ("personID")
);
```



- AraYüz:


Form1
Listele
Ekle
Sil
Guncelle
Ara
connection.Open();
adapter.Fill(dataSet);
if (dataSet.Tables.Count > 0 && dataSet.Tables[0].Rows.Count > 0)
{
dataGridViewTeams.DataSource = dataSet.Tables[0];
}
else
{
MessageBox.Show("No data found.");


Project
Build
Debug
Test
Analyze
Tools
Extensions
Window
Help
Debug
Any CPU
Continue
Lifecycle Events
Thread:
Stack
Form1.cs
Form1.Designer.cs
frm_players.cs
Form1.cs [Design]
ArayuzFootBallForm1
Form1
personID
teamID
posbions
18
18
Forward
19
19
Goalkeeper
20
20
Striker
1
2
Defender
2
2
Defender
21
2
Defender
Listele
Ekle
Silme
Guncelle
Ara

# Tablolar:

```
CREATE TABLE "teams" (
  "teamID" SERIAL PRIMARY KEY,
  "name" VARCHAR(50),
  "city" VARCHAR(50),
  "est_year" DATE
);
```

```
CREATE TABLE "person"."person" (
        "personID" SERIAL,
        "firstName" VARCHAR(50) NOT NULL,
        "lastName" VARCHAR(50) NOT NULL,
        "birthDate" DATE,
        CONSTRAINT "personPK" PRIMARY KEY ("personID")
);
```

```
CREATE TABLE "person"."players" (
   "personID" INTEGER,
   "teamID" INTEGER,
   "positions" VARCHAR(50),
   "nationalTeam" VARCHAR(50),
   CONSTRAINT "playersPK" PRIMARY KEY ("personID"),
   CONSTRAINT "teamFK" FOREIGN KEY ("teamID") REFERENCES
"teams"("teamID")
);


CREATE TABLE "person"."coaches" (
   "personID" INTEGER,
   "teamID" INTEGER,
   CONSTRAINT "coachesPK" PRIMARY KEY ("personID"),
   CONSTRAINT "teamFK" FOREIGN KEY ("teamID") REFERENCES
"teams"("teamID")
);

CREATE TABLE "person"."referees" (
   "personID" INTEGER,
   "country" VARCHAR(50),
   CONSTRAINT "refereesPK" PRIMARY KEY ("personID")
);
```

```
CREATE TABLE "leagues" (
  "leagueID" SERIAL PRIMARY KEY,
  "leagueName" VARCHAR(50),
  "season" VARCHAR(50),
  "county" VARCHAR(50),
  "teamID" INTEGER,
  FOREIGN KEY ("teamID") REFERENCES "teams"("teamID")
);


CREATE TABLE "standings" (
  "standingsID" SERIAL PRIMARY KEY,
  "teamID" INTEGER,
  "wins" INTEGER,
  "losses" INTEGER,
  "draws" INTEGER,
  FOREIGN KEY ("teamID") REFERENCES "teams"("teamID")
);

CREATE TABLE "awards" (
  "awardID" SERIAL PRIMARY KEY,
  "awardName" VARCHAR(50),
  "personID" INTEGER,
  FOREIGN KEY ("personID") REFERENCES "person"."person"("personID")
);
```

```
CREATE TABLE "transfers" (
  "transferID" SERIAL PRIMARY KEY,
  "personID" INTEGER,
  "oldTeam" VARCHAR(50),
  "newTeam" VARCHAR(50),
  "transferDate" DATE,
  FOREIGN KEY ("personID") REFERENCES "person"."person"("personID")
);


CREATE TABLE "teamAbilities" (
  "teamAbilitiesID" SERIAL PRIMARY KEY,
  "Attack" VARCHAR(50),
  "Mid" VARCHAR(50),
  "Defensive" VARCHAR(50),
  "teamID" INTEGER,
  FOREIGN KEY ("teamID") REFERENCES "teams"("teamID")
);

CREATE TABLE "stadium" (
  "stadiumID" SERIAL PRIMARY KEY,
  "stadiumName" VARCHAR(50),
  "teamID" INTEGER,
  FOREIGN KEY ("teamID") REFERENCES "teams"("teamID")
);
```

```
CREATE TABLE "matches" (
  "matchID" SERIAL PRIMARY KEY,
  "homeTeam" VARCHAR(50),
  "awayTeam" VARCHAR(50),
  "score" INTEGER,
  "matchDate" DATE,
  "stadiumID" INTEGER,
  FOREIGN KEY ("stadiumID") REFERENCES "stadium"("stadiumID")
);

CREATE TABLE "leagueMatches" (
  "leagueID" INTEGER,
  "matchID" INTEGER,
  PRIMARY KEY ("leagueID", "matchID"),
  FOREIGN KEY ("leagueID") REFERENCES "leagues"("leagueID"),
  FOREIGN KEY ("matchID") REFERENCES "matches"("matchID")
);

CREATE TABLE "goals" (
  "goalID" SERIAL PRIMARY KEY,
  "goalNumbers" INTEGER
);
```

```
CREATE TABLE "teamsGoals" (
  "goalID" INTEGER,
  "teamID" INTEGER,
  PRIMARY KEY ("goalID", "teamID"),
  FOREIGN KEY ("goalID") REFERENCES "goals"("goalID"),
  FOREIGN KEY ("teamID") REFERENCES "teams"("teamID")
);
```