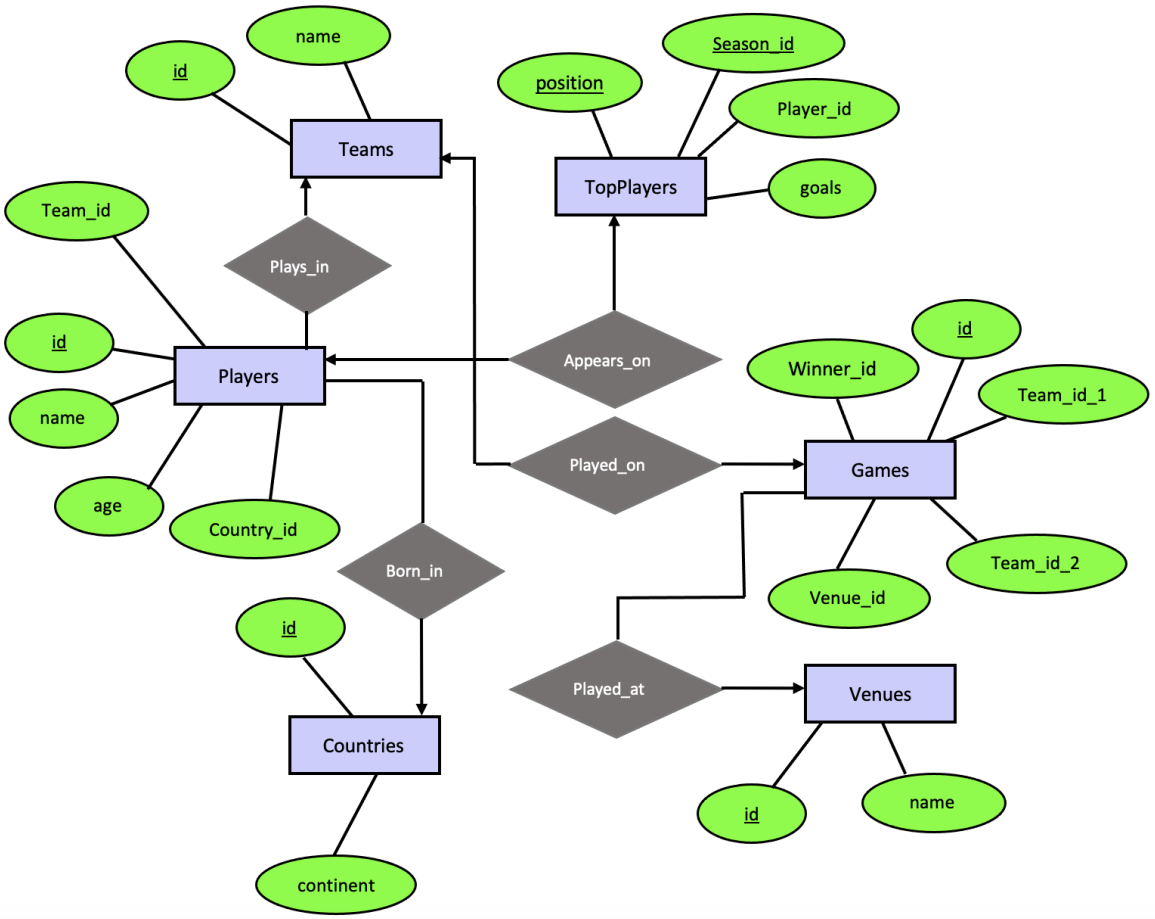
Software Documentation

Ariel ireni - 313914970
Yonatan Voikhansky - 315398339

DB Scheme Structure



Tables:

1. **Players** (id, name, age, country_id, team_id)
   PRIMARY KEY: id
   FOREIGN KEY:
   - country_id from Countries(id)
   - team_id from Teams(id)

2. **Teams** (id, name)
   PRIMARY KEY: id

3. **Countries** (id, name)
   PRIMARY KEY: id

4. **Venues** (id, name)
   PRIMARY KEY: id

5. **Games** (id, venue_id, team_id_1, team_id_2, winner_id)
   PRIMARY KEY: id
   FOREIGN KEY:
   - venue_id from Venues(id)
   - team_id_1 from Teams(id)
   - team_id_2 from Teams(id)
   - winner_id from Teams(id)

6. **TopPlayers** (position, season_id, player_id, goals)
   PRIMARY KEY: position, season_id
   FOREIGN KEY:
   - player_id from Players(id)

DB Optimization Performed:

The database we designed includes the six tables mentioned above: Teams, Venues, Games, Countries, Players, and TopPlayers.
We utilized database optimization techniques to ensure fast query execution and reduce storage space.
Each table has been optimized for performance by creating indices on frequently searched columns and implementing proper data normalization. Specifically, **indices** were added to our database's Players column because they are frequently searched and used in queries. As we've learned in class, without an index, the database would have to scan through the entire table to find the relevant data, which can be time-consuming and slow down the database's performance (especially in our full text query). By adding an index to the Players column, the database can quickly locate the desired data by searching the index rather than the entire table. This results in faster query execution and improved performance of the database overall.

Description of our queries

1. Average home win percentage

```
SELECT t.name as team_name,
       (SELECT COUNT(*)
        FROM Games g
        WHERE g.team_id_1 = t.id AND g.winner_id = t.id)
       /
       (SELECT COUNT(*)
        FROM Games g
        WHERE g.team_id_1 = t.id OR g.team_id_2 = t.id) as home_win_percentage
FROM Teams t
WHERE t.name = '{desired_team_name}'
ORDER BY home_win_percentage DESC
```

הסבר - בהינתן שם של קבוצה (desired_team_name) נקבל טבלה בעלת רשומה אחת שתציג את אחוז הניצחונות של קבוצה מסוימת במגרש הביתי שלה.
מבנה ה-DB תומך בשאילתה שכן בטבלת ה-Games יש forgien key ל-team_id של הקבוצות, שמאפשר לנו לבצע את הקישור בין המשחקים לאחוז הנצחונות של טבלה מסוימת.

2. Europeans under the age of

```
SELECT p.name as player_name,
    (SELECT c.name
     FROM arielireni.Countries c
     WHERE c.id = p.country_id
    ) as country_name
FROM arielireni.Players p
WHERE
    (SELECT c.continent
     FROM arielireni.Countries c
     WHERE c.id = p.country_id) = 'Europe'
AND p.age < {age_limit}
```

הסבר - בהינתן גיל מסוים (age_limit) נקבל טבלת שחקנים שמכילה את שמות כל השחקנים האירופאים שהם מתחת לגיל מסוים שנבחר.
מבנה ה-DB תומך בשאילתה שכן בטבלת ה-Players יש forgien key ל-country_id של המדינות, שמאפשר לנו לבצע את הקישור בין שחקנים למדינה הולדתם.

3. most losses in a Venue

```
SELECT t.name as team_name, v.name as venue_name, MAX(losses.num_losses) as number_of_losses
FROM arielireni.Teams t, arielireni.Venues v,
    (SELECT team_id_1 as team_id, venue_id, COUNT(*) as num_losses
     FROM arielireni.Games
     WHERE (team_id_1 = team_id_1 OR team_id_2 = team_id_1) AND winner_id != team_id_1
     GROUP BY team_id_1, venue_id) as losses
WHERE t.id = losses.team_id
AND v.id = (
    SELECT venue_id
    FROM (SELECT team_id_1 as team_id, venue_id, COUNT(*) as num_losses
        FROM arielireni.Games
        WHERE (team_id_1 = team_id_1 OR team_id_2 = team_id_1) AND winner_id != team_id_1
        GROUP BY team_id_1, venue_id) as all_losses
    WHERE all_losses.team_id = losses.team_id
    ORDER BY num_losses DESC
    LIMIT 1)
AND t.name = '{desired_team}'
GROUP BY t.name, v.name
```

הסבר - בהינתן שם של קבוצה מסוימת (desired_team) נחזיר טבלה בעלת רשומה שתציין עבור הקבוצה שנבחרה מה האצטדיון בו הקבוצה הפסידה הכי הרבה פעמים ואת מספר הפעמים.
מבנה ה-DB תומך בשאילתה שכן בטבלת ה-Games יש forgien key ל-venue_id של האצטדיונים שבהם קבוצה שיחקה והפסידה.

4. Teams with top players

```
SELECT t.name as team_name, Count(*) as num_of_players
FROM Teams as t, TopPlayers as top, Players as p
WHERE t.id = p.team_id AND p.id = top.player_id
GROUP BY t.id
ORDER BY num_of_players
DESC LIMIT {desired_limit}
```

הסבר - בהינתן מספר (desired_limit) נקבל טבלה שתציג מידע על הקבוצות בעלות מספר השחקנים הרב ביותר שמופיעים בטבלת השחקנים שכבשו הכי הרבה בכל עונה, כאשר נציג את שם הקבוצה ומספר השחקנים בסדר ממויין יורד, כאשר הקבוצה בעלת מספר השחקנים הרב ביותר תופיע ראשונה ומספר הקבוצות שנציג הוא כמספר שנבחר.
מבנה ה-DB תומך בשאילתה שכן בטבלת ה-TopPlayers יש forgien key ל-player_id של השחקנים, וגם בטבלת ה-Players יש forgien key ל-team_id של טבלת הקבוצות, מה שמאפשר לנו לבצע את הקישור בין השחקנים הכובשים לקבוצות בהן הם משחקים.

5. Top Players from certain country under age

```
SELECT p.name, top.goals
FROM Players as p, TopPlayers as top, Countries as c
WHERE p.id = top.player_id
AND p.country_id = c.id
AND c.name = '{desired_country_name}'
AND p.age < {desired_age_limit}
ORDER BY top.goals DESC
```

הסבר - בהינתן גיל מסוים (desired_age_limit) ומדינה מסוימת (desired_country_name) נחזיר טבלה המכילה את כל השחקנים מתחת לגיל הנבחר שנולדו במדינה המסוימת שנבחרה שהופיעו אי פעם בטבלת כובשי השערים.
מבנה ה-DB תומך בשאילתה שכן בטבלת ה-TopPlayers יש forgien key ל-player_id של השחקנים, וגם בטבלת ה-Players יש forgien key ל-country_id של טבלת המדינות, מה שמאפשר לנו לבצע את הקישור בין השחקנים הכובשים למדינת הולדתם.

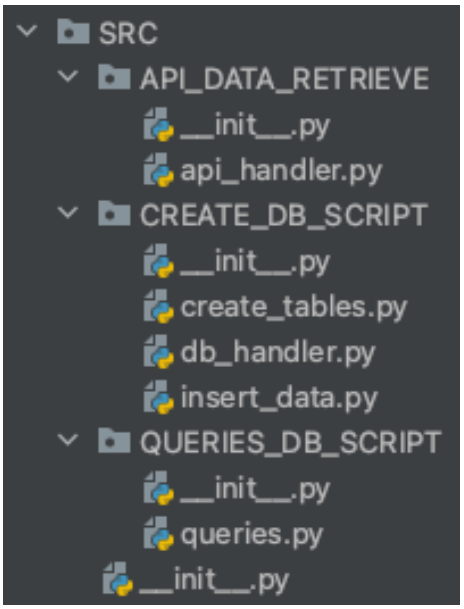
6. Player data

```
SELECT p.name as player_name , c.name as country_name, t.name as team_name
FROM Players as p, Countries as c, Teams as t
WHERE p.name LIKE '%{user_keyword}%'
AND p.country_id = c.id
AND p.team_id = t.id
```

הסבר - בהינתן מילת מפתח אותה מספר המשתמש (user_keyword) נחזיר מידע על כל השחקנים (שם שחקן, ארץ הולדתו והקבוצה בה הוא משחק) אשר שמם מכיל את מילת המפתח לפיה ביצענו את החיפוש
מבנה ה-DB תומך בשאילתה שכן בטבלת ה-Players יש forgien key ל-country_id של המדינות, וגם בטבלת ה-Players יש forgien key ל-team_id של טבלת הקבוצות, מה שמאפשר לנו להציג מידע נרחב מכל הטבלאות על השחקן. נציין בנוסף כי השימוש באינדקסים בטבלת ה-Players מאפשר להריץ חיפוש יעיל שכן החיפוש בטבלה מתבצע לפי שם השחקן, אותו בחרנו לייצג באינדקסים.

Code Structure

The SRC dictionary of our project was built in the following structure:



- /API_DATA_RETRIEVE
  - /api_handler.py
    The python script we used to get the data from the API we chose.
    We implemented a class called APIHandler, and called its functions from the db_handler.py file, in order to store the data we fetched in our DB.
- /CREATE_DB_SCRIPT
  - /db_handler.py
    The python script we used to store the data from the API in our DB.
    We implemented a class called DBHandler, and called its functions from the create_tables.py and insert_data.py files, in order to create the tables and insert data into them.
  - /create_table.py
    The python script we ran to create all the tables.
  - /insert_data.py
    The python script we ran to insert all the data.
    Note: since we had API request limitations we had to split the script into a few parts and run for several hours.
- /QUERIES_DB_SCRIPT
  - /queries.py
    The python script we used to run our queries.

Description of our API

ה- api בו השתמשנו הוא האתר my sportmonks המכיל מידע רב על על קבוצות כדורגל רבות בליגות שונות ושחקנים שונים. לינק לאתר - https://my.sportmonks.com

The general flow of the application

1. בתור התחלה עלינו להריץ את הסקריפט create_tables שמופיע תחת CREATE_DB_SCRIPT על מנת לייצר את הטבלאות הרלוונטיות ב- DB.
2. נריץ את insert_data שמופיע תחת CREATE_DB_SCRIPT על מנת להזין את המידע ל- DB.
   a. כל פונקציית insert תקרא לפונקציה המתאימה ב- api_handler שנמצא תחת התיקייה API_DATE_RETRIEVE על מנת לייבא את המידע מהמקורות השונים, ואז תכניס אותו לטבלה הרלוונטית.
3. לבסוף נוכל להריץ את הסקריפט queries.py (מכיל main מומלץ לנסות!) שמכיל בתוכו את 6 השאילתות השונות ולקבל את המידע הרלוונטי.