Databases HW2

Taya Harshuk 207944257

Jonathan Josef 203304480

Database Design:

Our DB contains 6 Tables and 9 views. A graph of table's dependencies attached below.

The tables are:

Teams: Contain team IDs

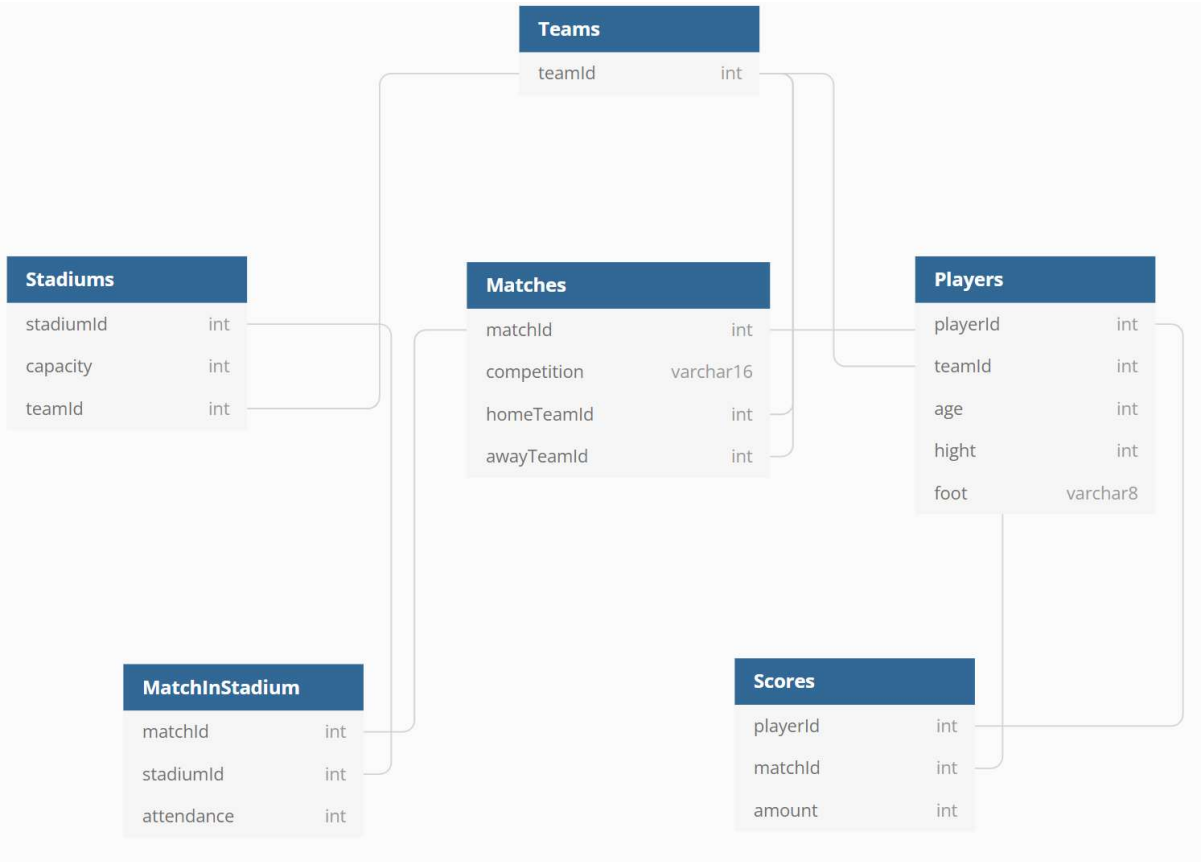
Players: Contain Players (player Id, team id [reference of team Id in Teams], age, height, age, foot)

Matches: Contain Matches (match Id, home & away team ids [references of team Id in Teams], competition)

Stadiums: Contain Stadiums (Stadium Id, team Id [reference of team Id in Teams], capacity)

MatchInStadium: Contain records of matches taking places in stadiums (match Id, stadium Id, attendance)

Scores: Contain records of players score in a match (player Id, match Id, amount)



The views are:

personalStats:

Schema:

PlayerId (for all players)	matchId (a record for every match played by PlayerId), None if didn't play any match	amount (the amount of goals scored by PlayerId in MatchId). 0 if didn't score.

Used for: playerIsWinner

goalsPerMatch:

Schema:

matchId (only for matches that had goals)	Goals scored in that match

Used for: playerIsWinner, goalsPerStadium (another view)

goalsPerPlayer:

Schema:

playerId (for all players)	Goals scored by that player (including 0)

Used for: mostGoalsForTeam

activeTeams: A list of active team IDs. Used for ActiveTallTeam

tallTeams: A list of tall team IDs. Used for ActiveTallTeam

activeTallTeams: A list of active tall team IDs. Intersect between the last two views.

Used for: getActiveTallTeams, getActiveTallRichTeams.

minAttendancePerTeam:

Schema:

teamId (only teams that played at least one home match)	minAttendanceInHomeMatch (self-explanatory). 0 If # attendance doesn't exist for at least 1 home match.

Used for: popularTeams

goalsPerStadium:

Schema:

StadiumId (only stadiums that hosted at least 1 match)	goals Number of goals scored in the stadium. Including 0.

Used for: getMostAttractiveStadiums

friends:

Schema:

Pid1	Pid2

Description:

A record for each pid1 scored in the same match pid2.

Contain duplicates (both for multiple matches, and ‘reversed’ duplicates for the same match. i.e., the relation is symmetric)

Used for: getClosePlayers

Code Design:

We created a few utils function and two global lists – Tables & Views.

It's important to understand that those lists aren't important for the code logic, only for good coding practices (for example, avoid using explicit table names on create/clean/drop).

Utils function are:

_errorHandling(e) -> ReturnValue:
Catch an exception and convert it to.
Another error has been added: Error # 22001.
This extra error is a special treatment to a case where a string (foot/competition) is not in the defined domain ({'Left', 'Right'} / {'Domestic', 'International'}) but is also out of the defined length (8/16).
With this special treatment we could limit the strings without fearing of DB error instead of BAD_PARAMS.

sendQuery(query) -> collections.namedtuple("QueryResult", ["Status", "RowsAffected","Set"])
Send a query, convert exceptions if gets any, and returns a named tuple with the values of Status (the ReturnValue), RowsAffected, and Set (which is the ResultSet)

_createTable(name, colNames, colTypes, extraProperties, foreignKey=None, checks=None, extraStatements=None)
Returns a table definition for the table generator

_createView(name, query, toMaterialize)
Returns a view definition for the view generator

defineTables()
Define all the tables, and append them to Tables in the right order

defineView()
Define all the views, and append them to Views in the right order

-API implantation

11**CRUD API**

(AddTeam(int teamId- אנו מוסיפים שורה לטבלה Teams שיצרנו עם ערך הteamId שמקבלים.
ערכי החזרה של הפונקציה כפי שמפורטים בגליון.

(addMatch(Match match- בהינתן class match מטיפוס Match, אנו מוציאים את הערכים הרלוונטים לattributes בטבלה שיצרנו- Matches, ומוסיפים את הmatch לטבלה .(הוספת שורה)
ערכי החזרה של הפונקציה כפי שמפורטים בגליון.

(getMatchProfile(int matchId – בהינתן הid של הmatch, אנו מוצאים אותו בטבלה Matches ומוציאים את הערכים הרלוונטים אליו, בעזרתם אנו יוצרים match מטיפוס Match ומחזירים אותו.
במידה ולא קיים משחק בטבלה matches עם matched שקיבלנו, נחזיר ()BadMatch.

(deleteMatch(Match match- נחלץ את ה matchId של match על מנת לחפש אותו בטבלה Matches ולבצע מחיקה אם קיים שם.(מחיקת שורה בטבלה)
ערכי החזרה של הפונקציה כפי שמפורטים בגליון.

(addPlayer(Player player- בהינתן player מטיפוס Player, נוציא ממנו את כל הערכים הרלוונטים לattributes בטבלה שיצרנו- Players ומוסיפים את הplayer לטבלה.(הוספת שורה)
ערכי החזרה של הפונקציה כפי שמפורטים בגליון.

(getPlayerProfile(int playerId- בהינתן playerId, נחפש בטבלה Players את השחקן המתאים לid הנתון. במידה ומצאנו אותו, ניצור player מטיפוס Player עם הערכים המתאים לשורה של השחקן שמצאנו, ונחזיר אותו. אחרת נחזיר ()BadPlayer.

(deletePlayer(Player player- בהינתן player מטיפוס Player, נוציא מהטיפוס את הplayerId ואותו נחפש בטבלה Players. במידה וקיים, נבצע מחיקה של השורה המיוצגת ע"י playerId בטבלה Players ונחזיר OK . אחרת, במידה ולא קיים נחזיר NOT_EXISTS וערך ההחזרה הנוסף הוא לפי הנתון בגליון.

(addStadium(Stadium stadium- בהינתן stadium מטיפוס Stadium, אנו מוציאים את הערכים הרלוונטים לattributes בטבלה שיצרנו- Stadiums, ומוסיפים את הstadium לטבלה .(הוספת שורה)
ערכי החזרה של הפונקציה כפי שמפורטים בגליון.

(getStadiumProfile(int stadiumID- נחפש בטבלה Stadiums באמצעות stadiumID. במידה ונמצא שורה מתאימה בטבלה, נוציא את הattributes של השורה ונבנה stadium מטיפוס Stadium ונחזיר אותו. אחרת, במידה ולא קיים נחזיר NOT_EXISTS וערך ההחזרה הנוסף הוא לפי הנתון בגליון.

(deleteStadium(Stadium stadium- בהינתן stadium מטיפוס Stadium, נוציא מהטיפוס את ערך הstadiumId ונבצע לפיו חיפוש בטבלה Stadiums. במידה ונמצא שורה מתאימה, נמחק אותה מהטבלה ונחזיר OK. אחרת, במידה ולא נמצא שורה מתאימה, נחזיר NOT_EXISTS וערך ההחזרה

הנוסף הוא לפי הנתון בגליון.

-BASIC API

-playerScoredInMatch(Match match, Player player, int Amount) נחלץ מmatch ומplayer את matchID ו-playerID בהתאמה ונכניס לטבלה scores שורה חדשה עם הערכים amount,playerID,matchId. ערכי החזרה כפי שמפורטים בגליון.

-playerDidntScoreInMatch(Match match, Player player) בהינתן match וplayer נחלץ מהם את matchID ואת playerID בהתאמה ונמחק את השורה בscores בה מופיעים ערכים אלו תחת העמודות המתאימות. ערכי החזרה כפי שמפורטים בגליון.

matchInStadium(Match match, Stadium stadium, int attendance) – נחלץ מmatch ומstadium את matchID ואת stadiumID בהתאמה. כעת נוסיף שורה חדשה לטבלה matchInStadium עם הערכים match,stadium,attendance. ערכי החזרה כפי שמפורטים בגליון.

matchNotInStadium(Match match, Stadium stadium) – נחלץ מmatch ומstadium את matchID ואת stadiumID בהתאמה. כעת נחפש את השורה המתאימים לערכים אלו בטבלה matchInStadium ובמידה ונמצא, נמחק אותה. ערך החזרה של כל מקרה אחר- לפי המפורט בגליון.

-averageAttendanceInStadium(int stadiumID) בהינתן stadiumID נפעיל SELECT על הטבלה matchInStadium כך שהשורות שיישארו יהיו השורות של stadiumID בלבד. לאחר מכן נפעיל AVG על עמודת attendance ונחזיר את הערך המתקבל. ערכי החזרה כפי שמפורטים בגליון.

-stadiumTotalGoals(int stadiumID) השתמשנו בטבלאות matchInStadium ובטבלה scores על מנת להפיק עמודה ושורה אחת שמכילה את המספר הנדרש להחזיר- כמות הגולים שהובקעו באצטדיון עם stadiumID. במידה והstasiumId לא קיים , נחזיר 0 ובמקרה של error נחזיר 1-.

-playerIsWinner(int playerID, int matchID) השתמשנו בטבלאות view (עליהן הרחבנו תחילה) personalStats ו- goalsPerMatch. ביצענו עליהן union all על מנת לא לאבד כפילויות והחזרנו true במידה ומתקיים $personalAmount * 2 \leq totalAmount$.

-getActiveTallTeams() השתמשנו בactiveTallTeams (טבלת view עליה הסברנו בהתחלה) והחזרנו ממנה רשימה בסדר יורד באורך 5 כנדרש. במקרה שactiveTallTeams ריקה נחזיר רשימה ריקה.

-getActiveTallRichTeams() החזרנו רשימה בסדר יורד ובאורך של עד 5, של הid של הקבוצות שנמצאות בחיתוך של activeTallTeams (טבלת view) עם טבלת stadium שסיננו בה את השורות בהם $capacity \leq 55000$. במידה ואין בחיתוך אף קבוצה, נחזיר רשימה ריקה.

-popularTeams() אנחנו משתמשים בטבלאות Teams ו-minAttendancePerTeam(טבלת view אשר פירטנו עליה). מטילים את minAttendancePerTeam על Teams לקבלת מספר הצפיות המינימלי או null- במקרה ולא שיחקו משחקים באצטדיון. נזכור כי ב minAttendancePerTeam כבר מעניקים מס' צופים = 0 לאצטדיונים ששיחקו בהם משחקים אשר לא דווח בהם מספר הצופים. נסנן שורות עם $\geq$ 40000 צופים, נמיין , נגביל ל10 ונחזיר את הרשימה.

Advanced API:

getMostAttractiveStadiums:

Sources: Stadiums, goalsPerStadium (view)

We joined goals per stadium on stadiums to get goaled scored per stadium including the 0 for unreported stadiums. We ordered as expected and returned the list.

mostGoalsForTeam:

Sources: goalsPerPlayer(view)

We filtered the players by the required team, sorted, limited and returned.

getClosePlayers:

Sources: friends (view), Players, black magic.

This one was tricky.

We couldn't use too many views because the parameter (playerId) needed to be taking care of soon to have an efficient query.

We used 3 nested queries.

N1: Use friends, to filter by pid2=playerId and count by pid1. This way, we've got a relation that holds for every player that scored with playerId, how many matches they scored together. Another interesting row we've got, is how many matches playerId scored with itself (with no duplications) => How many matches playerId scored in.

N2: We joined N1 on Players to get (pid, scoredTogether) for every player existed. scoredTogether will be 0 if the player didn't score in any mutual match.

N3: We used our wild row on N2 – the self PlayerId row. We used it and the fact that the player scored with itself the most. Then, using N2 and MAX, we had a relation with a single constant col – the number of matches playerId scored in.

The query: We used N2 and N3 to get a relation with on col – pid. We filtered with "WHERE n2.pid <> {playerId} AND n2.playedTogether * 2 >= n3.max" sorted, limited, and returned.