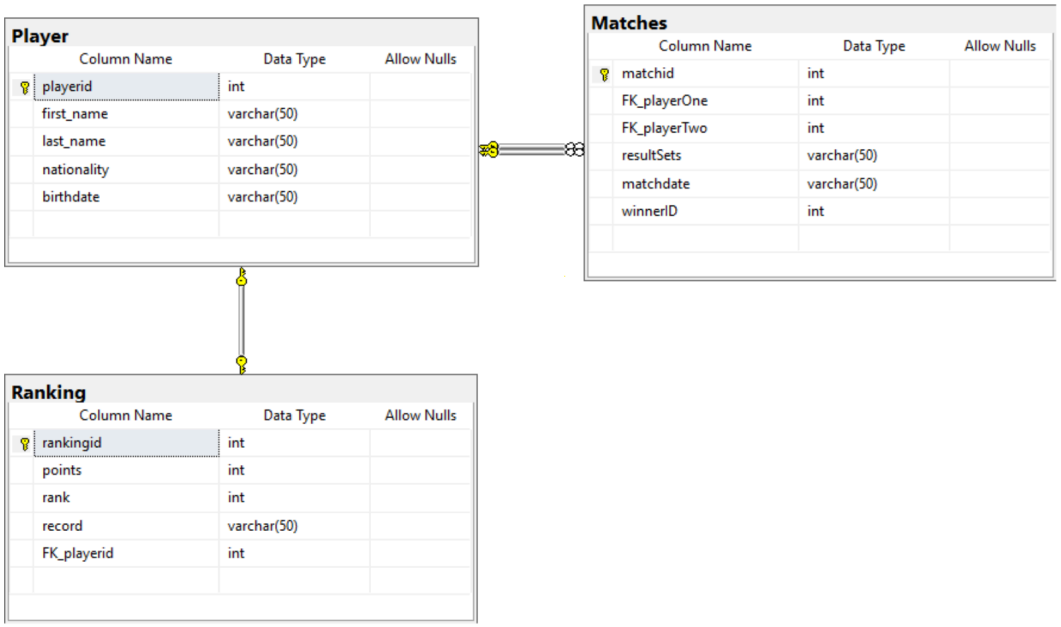
Week 1



TASKS

There are separate activities for the following queris that you have to do. You can find the expected output from each activity separately:

1. Print all players and their information

```
1  SELECT * FROM Player;
```

2. Print all player and ranking information of players in the top 10 ranking

```
1  SELECT * FROM Player, Ranking
2  Where rank <= 10
3  AND rankingid = playerid
```

3. Change all Spanish players to be from Portugal (abbreviation: POR).

```sql
UPDATE Player
SET nationality = 'POR'
WHERE nationality = 'ESP';
SELECT * FROM Player

```

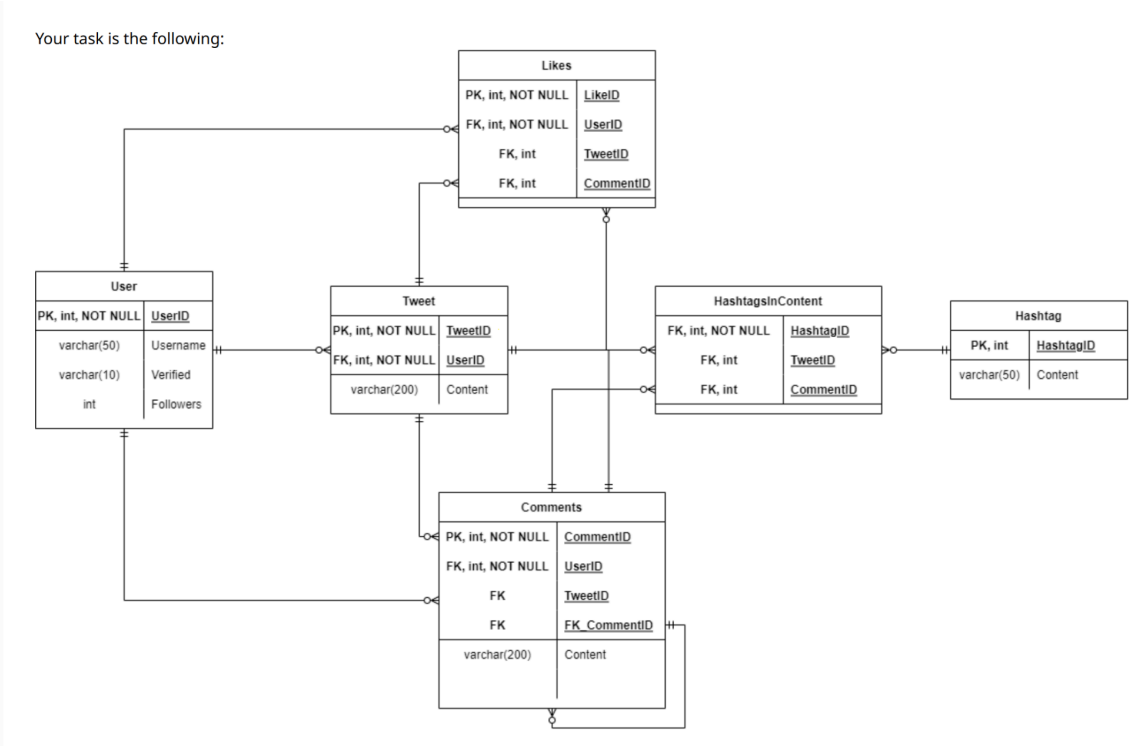
4. Add a new match between the first rank and last rank, where result is "0-0", match date is "unplayed" and winner is 0.

```sql
INSERT INTO Matches (FK_playerOne, FK_playerTwo, resultSets, matchdate, winnerID)
VALUES ('27', '30', '0-0', 'unplayed', '0');
SELECT * FROM Matches
```

5. Add a new player "Emil Ruusuvuori" from Finland (abbreviation: FIN), born 02/04/1999. Give him a new ranking at position 31 with 0 points and a record of "W: 0 - L: 0"

```sql
INSERT INTO Player (first_name, last_name, nationality, birthdate)
VALUES ('Emil', 'Ruusuvuori', 'FIN', '02/04/1999');

SELECT * FROM Player;

INSERT INTO Ranking (points, rank, record, FK_playerid)
VALUES (0, 31, 'W: 0 - L: 0', 31);

SELECT * FROM Ranking;
```

Week 2:



There are separate activities for the following queris that you have to do. You can find the expected output from each activity separately (if applicable):

Assignments 1-3 build on top of each other so that in the first one, you create a database and test it is created correctly. In the second, you add the insert commands as well to your database and submit everything and finally, add ON DELETE CASCADE commands to the table creation.

1. **(1 %) Create an SQL database using the data model above (NOTE, you have to submit ONLY CREATE TABLE commands. DO NOT add insert commands) BELOW**

```sql
CREATE TABLE User (
    UserID INT PRIMARY KEY NOT NULL,
    Username VARCHAR(50) NOT NULL,
    Verified VARCHAR(10) NOT NULL,
    Followers INT NOT NULL
);

CREATE TABLE Tweet (
    TweetID INT PRIMARY KEY NOT NULL,
    UserID INT NOT NULL,
    Content VARCHAR(200) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
);

CREATE TABLE Comments (
    CommentID INT PRIMARY KEY NOT NULL,
    UserID INT NOT NULL,
    TweetID INT NOT NULL,
    FK_CommentID INT,
    Content VARCHAR(200) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (TweetID) REFERENCES Tweet(TweetID) ON DELETE CASCADE,
    FOREIGN KEY (FK_CommentID) REFERENCES Comments(CommentID) ON DELETE CASCADE
);

CREATE TABLE Likes (
    LikeID INT PRIMARY KEY NOT NULL,
    UserID INT NOT NULL,
    TweetID INT,
    CommentID INT,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (TweetID) REFERENCES Tweet(TweetID) ON DELETE CASCADE,
    FOREIGN KEY (CommentID) REFERENCES Comments(CommentID) ON DELETE CASCADE
);

CREATE TABLE Hashtag (
    HashtagID INT PRIMARY KEY NOT NULL,
    Content VARCHAR(50) NOT NULL
);

CREATE TABLE HashtagsInContent (
    HashtagID INT NOT NULL,
    TweetID INT,
    CommentID INT,
    PRIMARY KEY (HashtagID, TweetID, CommentID),
    FOREIGN KEY (HashtagID) REFERENCES Hashtag(HashtagID) ON DELETE CASCADE,
    FOREIGN KEY (TweetID) REFERENCES Tweet(TweetID) ON DELETE CASCADE,
    FOREIGN KEY (CommentID) REFERENCES Comments(CommentID) ON DELETE CASCADE
);

```

2. **(1 %) Add ON DELETE CASCADE to all foreign key constraints. (NOTE, you have to submit all CREATE TABLE commands. <u>DO NOT</u> add insert commands))**

```sql
CREATE TABLE User (
    UserID INT PRIMARY KEY NOT NULL,
    Username VARCHAR(50) NOT NULL,
    Verified VARCHAR(10) NOT NULL,
    Followers INT NOT NULL
);

CREATE TABLE Tweet (
    TweetID INT PRIMARY KEY NOT NULL,
    UserID INT NOT NULL,
    Content VARCHAR(200) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS Comments (
    CommentID INT PRIMARY KEY NOT NULL,
    UserID INT NOT NULL,
    TweetID INT,
    FK_CommentID INT,
    Content VARCHAR(200) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (TweetID) REFERENCES Tweet(TweetID) ON DELETE CASCADE,
    FOREIGN KEY (FK_CommentID) REFERENCES Comments(CommentID) ON DELETE CASCADE
);

CREATE TABLE Likes (
    LikeID INT PRIMARY KEY NOT NULL,
    UserID INT NOT NULL,
    TweetID INT,
    CommentID INT,
    FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,
    FOREIGN KEY (TweetID) REFERENCES Tweet(TweetID) ON DELETE CASCADE,
    FOREIGN KEY (CommentID) REFERENCES Comments(CommentID) ON DELETE CASCADE
);

CREATE TABLE Hashtag (
    HashtagID INT PRIMARY KEY NOT NULL,
    Content VARCHAR(50) NOT NULL
);

CREATE TABLE HashtagsInContent (
    HashtagID INT NOT NULL,
    TweetID INT,
    CommentID INT,
    PRIMARY KEY (HashtagID, TweetID, CommentID),
    FOREIGN KEY (HashtagID) REFERENCES Hashtag(HashtagID) ON DELETE CASCADE,
    FOREIGN KEY (TweetID) REFERENCES Tweet(TweetID) ON DELETE CASCADE,
    FOREIGN KEY (CommentID) REFERENCES Comments(CommentID) ON DELETE CASCADE
);
```

3. **(2 %) Create INSERT commands to insert the data given in the excel file within homework section.**
   (NOTE, submit ONLY the INSERT commands, NOT THE CREATE TABLE commands)

```
INSERT INTO User (UserID, Username, Verified, Followers) VALUES
(10001, 'MelonHusk', 'true', 22049),
(10002, 'DonaldDuck', 'true', 149195),
(10003, 'KamKirl', 'false', 207464),
(10004, 'JustForLaughs', 'true', 22019),
(10005, 'TheRock', 'false', 221749),
(10006, 'ReynoldsFan', 'false', 65449),
(10007, 'SkullPoopL', 'false', 6511789),
(10008, 'PingFinity', 'true', 5464198),
(10009, 'HugeAckman', 'false', 1981497),
(10010, 'SnakeShot', 'true', 47896);

INSERT INTO Tweet (TweetID, UserID, Content) VALUES
(20001, 10002, 'Pretty sure that the world is just Duckburg'),
(20002, 10003, 'If you know what is good for you, you should do it.'),
(20003, 10004, 'Why the good die young and the bad go to hell?'),
(20004, 10005, 'Having snow in your shoe is as much fun as having warm beer.'),
(20005, 10007, 'Ice bucket challenge'),
(20006, 10008, 'Because science, right?'),
(20007, 10009, 'Did you know that the number of Nick Cage films correlate with drowning in pool?'),
(20008, 10001, 'Make sure you brush your hair before going to bed.'),
(20009, 10002, 'Bear, beer, beard, bird, turd. Bears are made of poop.'),
(20010, 10010, 'Rock ''n roll all night long with your best friends!');


INSERT INTO Comments (CommentID, UserID, TweetID, FK_CommentID, Content) VALUES
(30001, 10002, 20001, NULL, 'And Scrooge is the richest living being in the world.'),
(30002, 10003, 20002, NULL, 'What if you don''t know what is good for you? Do things to find out?'),
(30003, 10004, 20003, NULL, 'Because hell has to fill the torturer positions first.'),
(30004, 10006, 20004, NULL, 'Or as fun as making out with a pillow.'),
(30005, 10007, NULL, 30002, 'No no no, you ask from others what is good for them.'),
(30006, 10008, NULL, 30003, 'This sounds like the typical corporate ladder, where the first ones become executives and managers.'),
(30007, 10009, NULL, 30001, 'Does Mickey Mouse live in Duckburg or Mouseton?'),
(30008, 10008, NULL, 30002, 'Or never do anything so you don''t accidentally do anything bad.'),
(30009, 10009, 20008, NULL, 'The new way to handle bedhair?'),
(30010, 10010, 20009, NULL, 'I think you dropped the last screw from your brain.');


INSERT INTO Hashtag (HashtagID, Content) VALUES
(40001, '#win'),
(40002, '#friends'),
(40003, '#funny'),
(40004, '#giveaway'),
(40005, '#contest'),
(40006, '#thursdaythoughts'),
(40007, '#traveltuesday'),
(40008, '#science'),
(40009, '#fitness'),
(40010, '#goals');

INSERT INTO HashtagsInContent (HashtagID, TweetID, CommentID) VALUES
(40001, 20003, NULL),
(40002, 20004, NULL),
(40003, 20005, NULL),
(40004, 20006, NULL),
(40005, NULL, 30006),
(40006, NULL, 30007),
(40007, NULL, 30008),
(40008, NULL, 30009),
(40009, NULL, 30010),
(40010, 20002, NULL),
(40003, 20003, NULL),
(40004, NULL, 30004),
(40005, 20010, NULL),
(40006, 20004, NULL),
(40008, NULL, 30003),
(40009, NULL, 30004),
(40010, NULL, 30005);

INSERT INTO Likes (LikeID, UserID, TweetID, CommentID) VALUES
(50001, 10010, 20003, NULL),
(50002, 10008, 20005, NULL),
(50003, 10005, NULL, 30005),
(50004, 10010, NULL, 30007),
(50005, 10007, 20010, NULL),
(50006, 10001, NULL, 30007),
(50007, 10003, NULL, 30003),
(50008, 10005, 20009, NULL),
(50009, 10009, 20010, NULL),
(50010, 10010, 20010, NULL);
```

4.

---------- After this point, you do not need to (nor should you) submit the CREATE TABLE or INSERT commands ---------

(2 %)Create a view Comments_of_comments that shows only comments associated with other comments in the following format and order the results by username:
|User|Comment| Commented on |

```sql
CREATE VIEW Comments_of_comments AS
SELECT
    U.Username AS User,
    C.Content AS Comment,
    Parent.CommentID AS "Commented on"
FROM
    Comments C
JOIN
    Comments Parent ON C.FK_CommentID = Parent.CommentID
JOIN
    User U ON C.UserID = U.UserID
ORDER BY
    U.Username;
```

5. **(2 %) Create a trigger hashtag_not_allowed that triggers before data is inserted into Hashtag- table. The trigger should raise an abort printing "Mayonnaise detected!" when hashtag content has the word mayonnaise**

```sql
CREATE TRIGGER hashtag_not_allowed
BEFORE INSERT ON Hashtag
FOR EACH ROW
BEGIN

    SELECT CASE
        WHEN NEW.Content LIKE '%mayonnaise%' THEN
            RAISE(ABORT, 'Mayonnaise detected!')
    END;
END;
```

6. **(2 %) Create a view Tweets_and_tags that shows all hashtags with the associated Tweet in the following format and order the results by username:**

```sql
CREATE VIEW Tweets_and_tags AS
SELECT
    U.Username AS User,
    T.Content AS Tweet,
    GROUP_CONCAT(H.Content, '') AS Hashtag
FROM
    HashtagsInContent HC
JOIN
    Tweet T ON HC.TweetID = T.TweetID
JOIN
    Hashtag H ON HC.HashtagID = H.HashtagID
JOIN
    User U ON T.UserID = U.UserID
GROUP BY
    T.TweetID
ORDER BY
    U.Username;

```

Week 3:

1.

This is the tennis database but with more data in it. Your task is to do the following:

Pair up two players who have the same birthdate and print the p1 last name, birthdate and p2 last_name. Order the results first by descending birthdate and then default p1.playerid. Finally, limit the results to 300 rows.

```sql
SELECT
    p1.last_name AS p1lastname,
    p1.birthdate AS birthdate,
    p2.last_name AS p2lastname
FROM Player p1
JOIN Player p2
    ON p1.birthdate = p2.birthdate
    AND  p1.playerid != p2.playerid

ORDER BY
    p1.birthdate DESC,
    p1.playerid
LIMIT 300;
```

2.Your task is to do the following:Create a trigger that automatically creates the ranking information for new players and sets their rank to be the last one in ranking. This is the updated version of the homework 1.5 where you did this manually so similarly as in that homework, give the new players 0 points and a record of "W: 0 - L: 0" as well as automatically place them at the last rank. Note, that you should not add the rankingid as that is automatically given by the database.

```sql

CREATE TRIGGER Automaticrankingtrigger
AFTER INSERT ON Player
FOR EACH ROW
BEGIN

    INSERT INTO Ranking (points, rank, record, FK_playerid)
    VALUES (0, NEW.playerid, 'W: 0 - L: 0', NEW.playerid);
END;
```

Your task is to do the following:

Print player information (without birthdate) but change the nationality to continent information. So instead of having for example ESP, it should be Europe, or USA should be America and so on. Most nationalities are from the Europe so the other continents will only have a couple of nationalities (and players).

Expected output:

First_name|Last_name|Continent
Novak|Djokovic|Europe
Roger|Federer|Europe
Rafael|Nadal|Europe
Stanislas|Wawrinka|Europe
Kei|Nishikori|Asia
Andy|Murray|Europe
...And so on...

homework34.sql

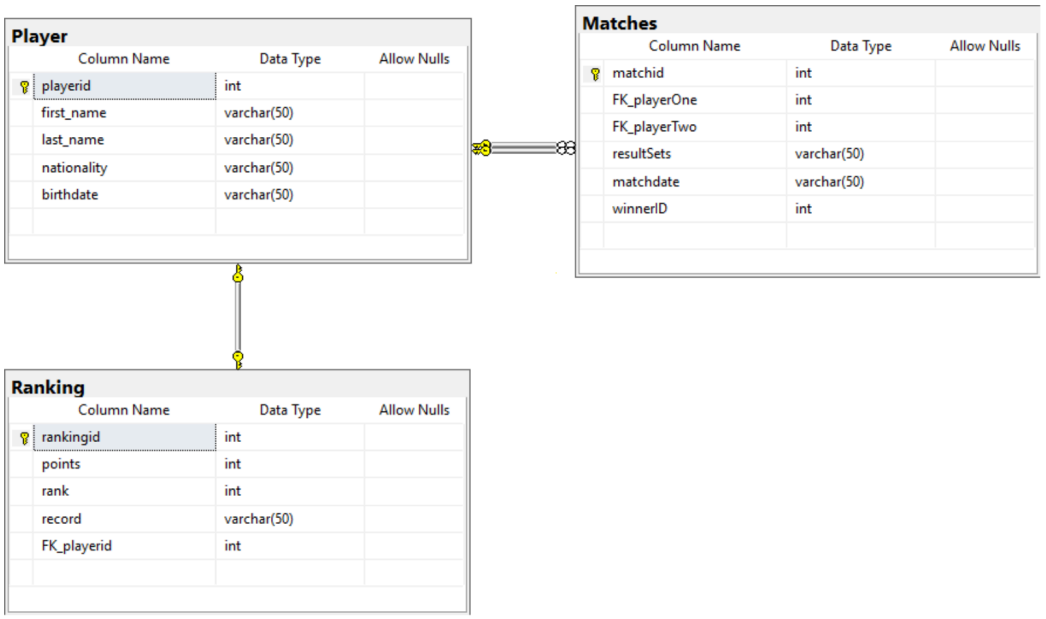
```sql
SELECT
    first_name,
    last_name,
CASE
    WHEN nationality = 'SRB' THEN 'Europe'
    WHEN nationality = 'SUI' THEN 'Europe'
    WHEN nationality = 'ESP' THEN 'Europe'
    WHEN nationality = 'JPN' THEN 'Asia'
    WHEN nationality = 'GBR' THEN 'Europe'
    WHEN nationality = 'CZE' THEN 'Europe'
    WHEN nationality = 'CAN' THEN 'America'
    WHEN nationality = 'CRO' THEN 'Europe'
    WHEN nationality = 'BUL' THEN 'Europe'
    WHEN nationality = 'LAT' THEN 'Europe'
    WHEN nationality = 'RSA' THEN 'Africa'
    WHEN nationality = 'FRA' THEN 'Europe'
    WHEN nationality = 'USA' THEN 'America'
    WHEN nationality = 'ITA' THEN 'Europe'
    WHEN nationality = 'BEL' THEN 'Europe'
    WHEN nationality = 'UKR' THEN 'Europe'
    WHEN nationality = 'GER' THEN 'Europe'
    WHEN nationality = 'ARG' THEN 'America'
    WHEN nationality = 'URU' THEN 'America'
    WHEN nationality = 'COL' THEN 'Europe'
    WHEN nationality = 'SVK' THEN 'Europe'
    WHEN nationality = 'AUT' THEN 'Europe'
    WHEN nationality = 'POL' THEN 'Europe'
    WHEN nationality = 'LUX' THEN 'Europe'
    WHEN nationality = 'RUS' THEN 'Europe'
    WHEN nationality = 'UZB' THEN 'Europe'
    WHEN nationality = 'AUS' THEN 'Oceania'
    WHEN nationality = 'POR' THEN 'Europe'
    WHEN nationality = 'BRA' THEN 'Europe'
    WHEN nationality = 'KAZ' THEN 'Europe'
    WHEN nationality = 'FIN' THEN 'Europe'
    WHEN nationality = 'TUN' THEN 'Europe'
    WHEN nationality = 'DOM' THEN 'Europe'
    WHEN nationality = 'SLO' THEN 'Europe'
    WHEN nationality = 'NED' THEN 'Europe'
    WHEN nationality = 'CYP' THEN 'Europe'
    WHEN nationality = 'LTU' THEN 'Europe'
    WHEN nationality = 'ISR' THEN 'Europe'
    WHEN nationality = 'TUR' THEN 'Europe'
    WHEN nationality = 'BIH' THEN 'Europe'
    WHEN nationality = 'ROU' THEN 'Europe'
    WHEN nationality = 'TPE' THEN 'Europe'
    WHEN nationality = 'IND' THEN 'Europe'
    WHEN nationality = 'HUN' THEN 'Europe'
```

```
    WHEN nationality = 'MDA' THEN 'Europe'
    WHEN nationality = 'KOR' THEN 'Asia'
    WHEN nationality = 'CHN' THEN 'Asia'
    WHEN nationality = 'EST' THEN 'Europe'
    WHEN nationality = 'IRL' THEN 'Europe'
    WHEN nationality = 'BLR' THEN 'Europe'
    WHEN nationality = 'CHI' THEN 'America'
    WHEN nationality = 'SWE' THEN 'Europe'
    WHEN nationality = 'BOL' THEN 'America'
    WHEN nationality = 'EGY' THEN 'Africa'
    WHEN nationality = 'VEN' THEN 'America'
    WHEN nationality = 'BAR' THEN 'Europe'
    WHEN nationality = 'ECU' THEN 'Europe'
    WHEN nationality = 'ESA' THEN 'Europe'
    WHEN nationality = 'DEN' THEN 'Europe'
    WHEN nationality = 'THA' THEN 'Europe'
    WHEN nationality = 'ZIM' THEN 'Africa'
    WHEN nationality = 'NZL' THEN 'Oceania'
    WHEN nationality = 'PER' THEN 'Europe'
    WHEN nationality = 'MON' THEN 'Europe'
    WHEN nationality = 'MEX' THEN 'America'
    WHEN nationality = 'GEO' THEN 'Europe'
    WHEN nationality = 'GRE' THEN 'Europe'
    WHEN nationality = 'MKD' THEN 'Europe'
    WHEN nationality = 'ALG' THEN 'Africa'
    WHEN nationality = 'LBA' THEN 'Europe'
    WHEN nationality = 'INA' THEN 'Europe'
    WHEN nationality = 'BEN' THEN 'Europe'
    WHEN nationality = 'MAR' THEN 'Europe'
    WHEN nationality = 'MNE' THEN 'Europe'
    WHEN nationality = 'NOR' THEN 'Europe'
    WHEN nationality = 'KGZ' THEN 'Europe'
    WHEN nationality = 'BDI' THEN 'Europe'
    WHEN nationality = 'PHI' THEN 'Europe'
    WHEN nationality = 'PUR' THEN 'Europe'
    WHEN nationality = 'IRI' THEN 'Europe'
    WHEN nationality = 'HAI' THEN 'Europe'
    WHEN nationality = 'MAD' THEN 'Europe'
    WHEN nationality = 'KUW' THEN 'Europe'
    WHEN nationality = 'SEN' THEN 'Europe'
    WHEN nationality = 'BRN' THEN 'Europe'
    WHEN nationality = 'KEN' THEN 'Europe'
    WHEN nationality = 'UAE' THEN 'Europe'
    WHEN nationality = 'CUB' THEN 'Europe'
    WHEN nationality = 'NAM' THEN 'Europe'
    WHEN nationality = 'BOT' THEN 'Europe'
    WHEN nationality = 'MLI' THEN 'Europe'
    WHEN nationality = 'CIV' THEN 'Europe'
    WHEN nationality = 'COD' THEN 'Europe'
END as continent
FROM Player;
```

```
--def luku(nimi):
    --lista = []
    --tiedosto = open(nimi, "r", encoding="UTF-8")
    --for rivi in tiedosto:
       -- monni = rivi.rstrip()
       -- if monni not in lista:
          --lista.append(monni)

    --return lista


--def tulostus(lista):
    ---amerikka = ['USA', 'ARG', 'CAN']
    --aasia = ['JPN', 'CHN', 'KOR']
   --oceania = ['AUS', 'NZL']
   --for maa in lista:
       -- --if maa in amerikka:
          --  print(f"WHEN nationality = '{maa}' THEN 'America'")
       -- elif maa in aasia:
       --     print(f"WHEN nationality = '{maa}' THEN 'Asia'")
      -- elif maa in oceania:
         --  print(f"WHEN nationality = '{maa}' THEN 'Oceania'")
       -- else:
          -- print(f"WHEN nationality = '{maa}' THEN 'Europe'")
--def paaohjelma():
    --nimi = "sonni.txt"
   -- sonni = luku(nimi)
   --tulostus(sonni)


--paaohjelma()
```

Week 4

Beneath you will find a database diagram for the database you are accessing.



Return your assignment as .py file. If you use the built-in testing ground, make sure the file you name has .py at the end.

Your task is to do the following:

You are given a Python template in this homework section to fill that has the following functions:

1. **def printPlayers()**
   Print player information row by row.
2. **def printRanking()**
   Print ranking information row by row.
3. **def printMatches()**
   Print match information row by row.
4. **def searchPlayer()**
   The search is done based on player surname and as an output, you should print the player's data on separate rows (print rows are included in the template).
5. **def moveMatch()**
   The match date can also be set to NULL. If this is done, it means that the match has not been played so you need to NULL the winner and score as well.
6. **def deletePlayer()**
   When deleting the player, make sure the ranking table is also modified (remove the player information) and modify the match information (set NULL the player ID in all instances where the removed player appears).¨ NOTE! When you set something to NULL in Python, it will print as None, not an empty string or null.

Your task is to fill the functions with the appropriate Python and SQL commands.

Hint: For the first three, you can find an example from the lecture slides.

```
##################################################
############## Do not touch this part ##############
import sqlite3
db = sqlite3.connect('hw5tennis.db')
cur = db.cursor()
def initializeDB():
    try:
        f = open("sqlcommands.sql", "r")
        commandstring = ""
        for line in f.readlines():
            commandstring+=line
        cur.executescript(commandstring)
    except sqlite3.OperationalError:
        print("Database exists, skip initialization")
    except:
        print("No SQL file to be used for initialization")


def main():
    initializeDB()
    userInput = -1
    while(userInput != "0"):
        print("\nMenu options:")
        print("1: Print Players")
        print("2: Print Ranking")
        print("3: Print Matches")
        print("4: Search for one player")
        print("5: Move matchdate")
        print("6: Delete player")
        print("0: Quit")
        userInput = input("What do you want to do? ")
        print(userInput)
        if userInput == "1":
            printPlayers()
        if userInput == "2":
            printRanking()
        if userInput == "3":
            printMatches()
        if userInput == "4":
            searchPlayer()
        if userInput == "5":
            moveMatch()
        if userInput == "6":
            deletePlayer()
        if userInput == "0":
            print("Ending software...")
    db.close()
    return

############## Do not touch part ends ##############
##################################################


```

```
############## Please modify the following ##############
def printPlayers():


    print("Printing players")

    for row in cur.execute('SELECT * FROM Player'):
        print(row)

    #Start your modifications after this comment
    #You should print the data noe row at a time.


    return

def printRanking():
    print("Printing ranking")
    """
    Insert the correct Python and SQL commands
    to print all ranking information
    """
    cur.execute("SELECT * FROM Ranking;")

    results = cur.fetchall()

    for row in results:
        print(row)

    #Start your modifications after this comment
    #You should print the data noe row at a time.

    return

def printMatches():
    print("Printing matches")
    """
    Insert the correct Python and SQL commands
    to print all ranking information
    """

    cur.execute('SELECT * FROM Matches')
    results = cur.fetchall()

    for row in results:
        print(row)
    #Start your modifications after this comment
    #You should print the data one row at a time.



    return
```

```
def searchPlayer():
    playerName = input("What is the player's surname? ")
    """
    Insert the correct Python and SQL commands to find the player
    using the given surname
    """
    #Start your modifications after this comment
    #You are given the print statements, now you need to add the fetched data to the five prints.

    cur.execute("SELECT * FROM Player WHERE last_name='{}'".format(playerName))
    oneROW = cur.fetchone()
    print("ID: " + str(oneROW[0]))
    print("First name: " + str(oneROW[1]))
    print("Last name: " + str(oneROW[2]))
    print("Birthdate: " + str(oneROW[4]))
    print("Nationality: " + str(oneROW[3]))


    return

def moveMatch():
    matchID = input("What is the matchID of the match you want to move? ")
    newMatchDate = input ("What is the new matchdate you want to set?")



    """
    Using the correct Python and SQL comands:
    Change the match date based on the given matchID and new matchdate
    IF a new matchdate is set to NULL, set the winner and result to NULL as well
    """
    #Start your modifications after this comment

    cur.execute("UPDATE Matches SET matchdate = '{}' WHERE matchid='{}'".format(newMatchDate, matchID))



    return

def deletePlayer():
    playerID = input("What is the player's PlayerID? ")
    """
    Using the correct Python and SQL comands:
    Delete the Player and his Ranking information
    Additionally, set the playerid to NULL in ALL match-data it is found
    """
    #Start your modifications after this comment

    cur.execute("DELETE FROM Ranking WHERE matchid ='{}'".format(playerID))
    cur.execute("UPDATE matches SET player_id = NULL WHERE player_id = '{}'".format(playerID))
    cur.execute("DELETE FROM players WHERE id = '{}'".format(playerID))
main()
```

week 5

You are given a database that has one table of different data related to bands, albums, tracks etc.

```
CREATE TABLE musicrecords
    band VARCHAR (50)
    band_member VARCHAR (50)
    member_instrument VARCHAR (50),
    track VARCHAR (50),
    track_duration VARCHAR (50)
    album VARCHAR (50),
    releaseYear INTEGER
);
```

Normalize this table by creating the following views:
View_1: Band data, should include the band, its members and their instruments. **Data order**: Band, band member.
View_2: Album data, should include the band, album and its release year. **Data order**: Band, release year.
View_3: Track data, should include the band, album, its tracks and their duration. **Data order**: Band, album, track duration

Use the view names **View_1, View_2, View_3**
Your views should **not have** rows containing **NULL values**.
You do NOT have to create anything else than the views.

1.

homework51.sql

```sql
CREATE VIEW View_1 AS
SELECT DISTINCT band, band_member, member_instrument
FROM musicrecords
WHERE band IS NOT NULL
  AND band_member IS NOT NULL
  AND member_instrument IS NOT NULL
ORDER BY band, band_member;


CREATE VIEW View_2 AS
SELECT DISTINCT band, album, releaseYear
FROM musicrecords
WHERE band IS NOT NULL
  AND album IS NOT NULL
  AND releaseYear IS NOT NULL
ORDER BY band, releaseYear;


CREATE VIEW View_3 AS
SELECT band, album, track, track_duration
FROM musicrecords
WHERE band IS NOT NULL
  AND album IS NOT NULL
  AND track IS NOT NULL
  AND track_duration IS NOT NULL
ORDER BY band, album, track, track_duration;
```

2. This is the tennis database. Your task is to do the following:

You should create a new table called player_updated, that contains the player data but instead of first name and last name, there should only be one column for the full_name of players (otherwise identical table). After creating the table, you can copy data from the old table to the new one.

homework52.sql

```sql
CREATE TABLE player_updated (
    playerid INTEGER PRIMARY KEY,
    full_name TEXT NOT NULL,
    nationality TEXT NOT NULL,
    birthdate TEXT NOT NULL
);


INSERT INTO player_updated (playerid, full_name, nationality, birthdate)
SELECT playerid, first_name || ' ' || last_name AS full_name, nationality, birthdate
FROM player;
```

This is the tennis database but with more data in it. Your task is to do the following:

Print out the nationalities, the number of players in the nation and the difference to the average number of players in all countries that has been rounded to the nearest full number. Order the results by descending order of the number of players.

```sql
SELECT
    nationality,
    COUNT(*) AS num_players,
    ROUND(COUNT(*) - (SELECT AVG(player_count)
                      FROM (SELECT nationality, COUNT(*) AS player_count
                            FROM Player
                            GROUP BY nationality) AS avg_table)) AS diff_from_avg
FROM Player
GROUP BY nationality
ORDER BY num_players DESC;
```

4.

This is the tennis database but with more data in it. Your task is to do the following:

Print out the last_name, points, nat_average and nationalities of all players who have points over their national average. Round the average points to 1 decimal and limit the results to 300 rows.
Update on 20.02.2025: And order the results by playerid.

```sql
SELECT
    p.last_name,
    r.points,
    ROUND(n.avgerage_points, 1) AS national_average,
    p.nationality
FROM Player p
JOIN Ranking r ON p.playerid = r.FK_playerid
JOIN (
    SELECT p.nationality, ROUND(AVG(r.points), 1) AS avgerage_points
    FROM Player p
    JOIN Ranking r ON p.playerid = r.FK_playerid
    GROUP BY p.nationality
) n ON p.nationality = n.nationality
WHERE r.points > n.avgerage_points
ORDER BY p.playerid
LIMIT 300;
```

Week 6

Using the Tennis DB from Homework 2 (but much larger), you are given the following query:

```
SELECT * FROM Player
        INNER JOIN Ranking ON Player.playerid = Ranking.FK_playerid
        WHERE (SELECT COUNT (*) FROM Matches
                WHERE (Player.playerid = Matches.FK_playerOne OR
Player.playerid = Matches.FK_playerTwo) )/2 =
                (SELECT COUNT (*) FROM Matches WHERE (Player.playerid =
Matches.winnerID));
```

Create indices that allows the query to run in less than 4secs in Moodle.
You only need to submit the INDEX creation statements.

```
1  CREATE INDEX indexranking_fk_playerid ON Ranking(FK_playerid);
2
3  CREATE INDEX indexmatches_fk_playerOne ON Matches(FK_playerOne);
4  CREATE INDEX indexmatches_fk_playerTwo ON Matches(FK_playerTwo);
5  CREATE INDEX indexmatches_winnerID ON Matches(winnerID);
6
7  CREATE INDEX indexmatches_players ON Matches(FK_playerOne, FK_playerTwo);
8
9  CREATE INDEX indexmatches_all ON Matches(FK_playerOne, FK_playerTwo, winnerID);
```

2. Using the Tennis DB from Homework 2, perform the following:

Print all matches where the same players have faced off more than once against each other as well as the name of the winner of each match. Order the results by the winner AND then by match id (in case you got correct matches but they are still in slightly different order).
Use the following example columns and their respective data:
| *Player one* | *Player two* | *Matchdate* | *Winner* |
Add the following command to the beginning of your SQL Statement: .headers on
This enables SQLite to show the headers of your query.
"Player one" and "Player two" columns should contain first and last name of each player.

```
.headers on

SELECT
    p1.first_name || ' ' || p1.last_name AS "Player one",
    p2.first_name || ' ' || p2.last_name AS "Player two",
    m.matchdate AS "Matchdate",
    pw.first_name || ' ' || pw.last_name AS "Winner"
FROM Matches m
JOIN Player p1 ON m.FK_playerOne = p1.playerid
JOIN Player p2 ON m.FK_playerTwo = p2.playerid
JOIN Player pw ON m.winnerID = pw.playerid
WHERE EXISTS (
    SELECT 1 FROM Matches m2 WHERE
        ((m2.FK_playerOne = m.FK_playerOne AND m2.FK_playerTwo = m.FK_playerTwo) OR (m2.FK_playerOne = m.FK_playerTwo AND m2.FK_playerTwo = m.FK_playerOne)) AND m2.matc
ORDER BY "Winner", m.matchid;
```

3. Using the Tennis DB from Homework 2, perform the following:

Print all the matches the top 5 ranked players have lost in the following format and order the results by winner rank:

| *Winner name* | *Winner rank* | *Loser name* | *Loser rank* | *Matchdate* |

Add the following command to the beginning of your SQL Statement: .headers on

This enables SQLite to show the headers of your query.

```
.headers on

SELECT
    champ.first_name || ' ' || champ.last_name AS "Winner name",
    rank_champ.rank AS "Winner rank",
    runner_up.first_name || ' ' || runner_up.last_name AS "Loser name",
    rank_runner_up.rank AS "Loser rank",
    game.matchdate AS "Matchdate"
FROM Matches game
JOIN Player champ ON game.winnerID = champ.playerid
JOIN Player runner_up ON (game.FK_playerOne = runner_up.playerid OR game.FK_playerTwo = runner_up.playerid) AND runner_up.playerid <> game.winnerID
JOIN Ranking rank_champ ON champ.playerid = rank_champ.FK_playerid
JOIN Ranking rank_runner_up ON runner_up.playerid = rank_runner_up.FK_playerid
WHERE rank_runner_up.rank IS NOT NULL AND rank_runner_up.rank <= 5 ORDER BY rank_champ.rank;
```