|  |
| --- |
|  |
|  |  |
|  | ## Description of Tables |
|  |  |
|  | ScoreID Game Player Score |
|  | 1 1 A 11 |
|  | 2 1 B 7 |
|  | 3 2 A 15 |
|  | 4 2 C 13 |
|  | 5 3 B 11 |
|  | 6 3 D 9 |
|  | 7 4 D 11 |
|  | 8 4 A 5 |
|  | 9 5 A 11 |
|  | 10 6 B 11 |
|  | 11 6 C 2 |
|  | 12 6 D 5 |
|  |  |
|  |  |
|  | GameID Winner Date |
|  | 1 A 2017-01-02 |
|  | 2 A 2016-05-06 |
|  | 3 B 2017-12-15 |
|  | 4 D 2016-05-06 |
|  |  |
|  |  |
|  | PlayerID Name LastName |
|  | A Phil Watertank |
|  | B Eva Smith |
|  | C John Wick |
|  | D Bill Bull |
|  | E Lisa Owen |
|  | ############################ |
|  |  |
|  | 1)Table Creation |
|  |  |
|  | #######################SQL Solution################# |
|  |  |
|  | CREATE TABLE Game |
|  | ([ID] int, [Winner] varchar(1), [Date] datetime) |
|  | ; |
|  |  |
|  | INSERT INTO Game |
|  | ([ID], [Winner], [Date]) |
|  | VALUES |
|  | (1, 'A', '2017-01-02 00:00:00'), |
|  | (2, 'A', '2016-05-06 00:00:00'), |
|  | (3, 'B', '2017-12-15 00:00:00'), |
|  | (4, 'D', '2016-05-06 00:00:00') |
|  | ; |
|  |  |
|  |  |
|  | CREATE TABLE Player |
|  | ([ID] varchar(1), [Name] varchar(4), [LastName] varchar(9)) |
|  | ; |
|  |  |
|  | INSERT INTO Player |
|  | ([ID], [Name], [LastName]) |
|  | VALUES |
|  | ('A', 'Phil', 'Watertank'), |
|  | ('B', 'Eva', 'Smith'), |
|  | ('C', 'John', 'Wick'), |
|  | ('D', 'Bill', 'Bull'), |
|  | ('E', 'Lisa', 'Owen') |
|  | ; |
|  |  |
|  |  |
|  | CREATE TABLE Score |
|  | ([ID] int, [Game] int, [Player] varchar(1), [Score] int) |
|  | ; |
|  |  |
|  | INSERT INTO Score |
|  | ([ID], [Game], [Player], [Score]) |
|  | VALUES |
|  | (1, 1, 'A', 11), |
|  | (2, 1, 'B', 7), |
|  | (3, 2, 'A', 15), |
|  | (4, 2, 'C', 13), |
|  | (5, 3, 'B', 11), |
|  | (6, 3, 'D', 9), |
|  | (7, 4, 'D', 11), |
|  | (8, 4, 'A', 5), |
|  | (9, 5, 'A', 11), |
|  | (10, 6, 'B', 11), |
|  | (11, 6, 'C', 2), |
|  | (12, 6, 'D', 5) |
|  | ; |
|  |  |
|  | ######################################################## |
|  |  |
|  | 2)Show the average score of each player, even if they didn't play any games. |
|  | Expected output (Player ID, Name, Average Score) |
|  |  |
|  | ##################SQL Solution############## |
|  |  |
|  | SELECT p.ID, p.LastName AS "Name", |
|  | AVG(s.Score) AS AverageScore |
|  | FROM player p |
|  | LEFT JOIN score s |
|  | ON s.ID = p.PlayerID |
|  | GROUP BY p.ID, p.LastName; |
|  |  |
|  | ############################################# |
|  |  |
|  | 3)The score table is corrupted: a game can only have two players (not more, not less). Write a query that identifies and only shows the valid games and their winner. |
|  | Expected output (Game,Winner) |
|  |  |
|  | Bonus: as an additional challenge, you can also display the winner's score. The condition described above should still apply. |
|  | Expected output (Game, Winner, Winner Score) |
|  |  |
|  | ####################SQL Solution############################# |
|  |  |
|  | SELECT DISTINCT g.Game, g.Winner |
|  | fROM Game g |
|  | JOIN Score s |
|  | ON g.ID = s.Game; |
|  |  |
|  | ###########Bonus: SQL Solution################# |
|  |  |
|  | SELECT DISTINCT g.Game, g.Winner, MAX(s.Score) AS "WinnerScore" |
|  | FROM Game g |
|  | JOIN Score s |
|  | ON g.ID = s.Game |
|  | GROUP BY g.game, g.winner; |
|  |  |
|  | #################################### |
|  |  |
|  | 4) Show the score of players in games they he lost. |
|  | Expected output (Game ID, Player Name, Player LastName, Score) |
|  |  |
|  | ###################SQL Solution###################################### |
|  |  |
|  | SELECT a.GameId, p.Name AS "PlayerName", p.LastName AS "PlayerLastname" , a.Score |
|  | FROM Player p |
|  | JOIN (SELECT s.Player, g.GameId , MIN(S.Score) AS "Score" |
|  | FROM Score s |
|  | JOIN GAME g |
|  | ON s.Game=g.GameID |
|  | GROUP BY Game) a |
|  | ON a.Player=p.PlayerId; |
|  |  |
|  | ###################################################### |
|  |  |
|  | 5)Show the players which have not played any games? |
|  | Expected output (Player.ID, Player.Name, Player.LastName) |
|  |  |
|  | ######################SQL Solution##################### |
|  |  |
|  | SELECT DISTINCT Player.ID, Player.Name, Player.LastName |
|  | fROM Player p |
|  | LEFT JOIN Score s |
|  | on s.Player = p.ID |
|  | WHERE s.Player is null; |
|  |  |
|  | ##################################### |
|  |  |
|  | 6)Show the list of player combinations who have never played together. |
|  | Expected Output (Player1, Player2). Reverse duplicates are authorized. |
|  |  |
|  | ####################Python Solution############## |
|  |  |
|  | #######Table creation########## |
|  | import pandas as pd |
|  | score = pd.DataFrame([[1,1,'A',11],[2,1,'B',7],[3,2,'A',15],[4,2,'C',13],[5,3,'B',11],[6,3,'D',9],[7,4,'D',11],[8,4,'A',5]], |
|  | columns=['score\_id', 'game', 'player', 'score']) |
|  | current\_pairs = [] |
|  | ####nunique functions like COUNT DISTINCT in SQL############ |
|  | #############score['game'].nunique(), calculates the number of distinct games which in this question equals 4 ########## |
|  | ###########and 1 is added to it since using range, the number itself is not produced######## |
|  | ############the range is set to start from 1, since 'game'values start from 1 |
|  | for i in range(score['game'].nunique()+1)[1:]: |
|  | game\_pairs=[] |
|  | mod\_score= score.loc[score['game']==i,['player']] |
|  | ####mod\_score is a table#### |
|  | for j in range(2): |
|  | game\_pairs.append(mod\_score.iat[j,0]) |
|  | pairs=set(game\_pairs) |
|  | ####sets are used since order of players is not important####### |
|  | current\_pairs.append(pairs) |
|  | ######Player table creation########## |
|  | player = pd.DataFrame([['A','Phil','Watertank'],['B','Eva','Smith'],['C','John','Wick'],['D','Bill','Bull'],['E','Lisa','Owen']], |
|  | columns=['id', 'name', 'lastname']) |
|  | from itertools import combinations |
|  | potential\_players=[] |
|  | for i in range(player['id'].count()): |
|  | potential\_players.append(player.iat[i,0]) |
|  | comb = combinations(potential\_players,2) |
|  | ####comb itself is list of tuples, but since it should be compared with a list of sets, the below list is created.######### |
|  | potential\_pairs=[] |
|  | for j in comb: |
|  | j=set(j) |
|  | potential\_pairs.append(j) |
|  | unpaired\_players=[] |
|  | ####Comparison between current players and potential players#### |
|  | for i in potential\_pairs: |
|  | if i not in current\_pairs: |
|  | unpaired\_players.append(i) |
|  | print(unpaired\_players) |