from database.DB\_connect import DBConnect  
from model.oggetti import Oggetto  
  
class DAO():  
  
 def \_\_init\_\_(self):  
 pass  
  
 @staticmethod  
 def getAllColorsProducts():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT \* from go\_products"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllNodes(year):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
 query = """select distinct least(c.state1no , c.state2no) as state1no,  
 greatest(c.state1no , c.state2no) as state2no  
 from contiguity c   
 where c.`year` <= %s"""  
  
 cursor.execute(query, (year,))  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllConnessioni(year, product1\_code, product2\_code, color):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select COUNT(DISTINCT gds1.Date) as Peso, gds1.Retailer\_code as Retailer\_code,  
 gds1.Product\_number as Product1\_code, gds2.Product\_number as Product2\_code  
 from go\_daily\_sales gds1, go\_daily\_sales gds2, go\_products gp1, go\_products gp2  
 where YEAR(gds1.`Date`) = %s and YEAR(gds2.`Date`) = %s and   
 gds1.Product\_number = %s and gds2.Product\_number = %s and   
 gds1.Retailer\_code = gds2.Retailer\_code and gds1.`Date` = gds2.`Date` and  
 gds1.Product\_number = gp1.Product\_number and gp1.Product\_color = %s and   
 gds2.Product\_number = gp2.Product\_number and gp2.Product\_color = %s"""  
  
 cursor.execute(query, (year, year, product1\_code, product2\_code, color, color,))  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllConnessioni(idMap):  
 conn = DBConnect.get\_connection()  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT c.id\_connessione, c.id\_linea ,c.id\_stazP, c.id\_stazA FROM connessione c"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 stazP = idMap[row["id\_stazP"]]  
 stazA = idMap[row["id\_stazA"]]  
  
 if stazP is not None and stazA is not None:  
 result.append(Oggetto(row["id\_connessione"], row["id\_linea"], stazP, stazA))  
  
 cursor.close()  
 conn.close()  
 return  
  
 @staticmethod  
 def getAllNodes(numMin):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
 query = """select tmp.ID, tmp.IATA\_CODE, count(\*) as N  
 from (  
 SELECT a.ID , a.IATA\_CODE , f.AIRLINE\_ID, count(\*) as n  
 FROM airports a , flights f   
 WHERE a.ID = f.ORIGIN\_AIRPORT\_ID or a.ID = f.DESTINATION\_AIRPORT\_ID   
 group by a.ID , a.IATA\_CODE , f.AIRLINE\_ID  
 ) as tmp  
 group by tmp.ID, tmp.IATA\_CODE  
 having N >= %s"""  
  
 cursor.execute(query, (numMin,))  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllEdges():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
 query = """SELECT f.ORIGIN\_AIRPORT\_ID , f.DESTINATION\_AIRPORT\_ID , count(\*) as N  
 FROM flights f   
 group by f.ORIGIN\_AIRPORT\_ID , f.DESTINATION\_AIRPORT\_ID  
 order by f.ORIGIN\_AIRPORT\_ID , f.DESTINATION\_AIRPORT\_ID """  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllEdges(idMap):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
 query = """select eo1.object\_id as obj1, eo2.object\_id as obj2, count(\*) as peso  
 from exhibition\_objects eo1, exhibition\_objects eo2  
 where eo1.exhibition\_id = eo2.exhibition\_id and eo1.object\_id != eo2.object\_id   
 group by eo1.object\_id , eo2.object\_id """  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(idMap[row["obj1"]], idMap[row["obj2"]], row["peso"]))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllNodes(durata):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select a.\*, sum(t.Milliseconds)/1000/60 as totDurata  
 from track t, album a  
 where t.AlbumId = a.AlbumId   
 group by a.AlbumId  
 having totDurata > %s"""  
  
 cursor.execute(query, (durata,))  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllEdges(year, country):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT least(gr1.Retailer\_code, gr2.Retailer\_code) as Retailer1, greatest(gr1.Retailer\_code, gr2.Retailer\_code) as Retailer2, COUNT(DISTINCT s1.Product\_number) as peso  
 FROM go\_daily\_sales s1, go\_daily\_sales s2, go\_retailers gr1, go\_retailers gr2  
 WHERE YEAR(s1.Date) = YEAR(s2.Date) AND YEAR(s1.Date) = %s  
 and gr1.Country = %s and gr2.Country = %s  
 AND gr1.Retailer\_code > gr2.Retailer\_code  
 AND s1.Product\_number = s2.Product\_number  
 and s1.Retailer\_code = gr1.Retailer\_code and s2.Retailer\_code = gr2.Retailer\_code  
 GROUP BY gr1.Retailer\_code, gr2.Retailer\_code"""  
  
 cursor.execute(query, (year, country, country,))  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
  
 #COALESCE(expr1, expr2, ..., exprN) restituisce il primo valore NON NULL tra quelli passati.  
 # È una funzione di gestione dei NULL.  
 # Se expr1 è NULL, passa a expr2, e così via fino a trovare un valore.  
 @staticmethod  
 def getAllNodes(year):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT c.customer\_id, c.first\_name, c.last\_name,  
 COALESCE(SUM(oi.quantity), 0) AS totale\_acquistato  
 FROM customers c  
 LEFT JOIN orders o ON c.customer\_id = o.customer\_id  
 LEFT JOIN order\_items oi ON o.order\_id = oi.order\_id  
 GROUP BY c.customer\_id"""  
  
 cursor.execute(query, (year,))  
  
 for row in cursor:  
 result.append(Node(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # QUERY MENU SUL DATABASE FORMULA1 #  
  
 # prendi i piloti che hanno vinto almeno una gara  
 @staticmethod  
 def getMenu():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT d.driverId, d.forename, d.surname  
 FROM results r  
 JOIN drivers d ON r.driverId = d.driverId  
 WHERE r.position = 1  
 ORDER BY d.surname, d.forename"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi le squadre costruttori che hanno partecipato ad una determinata stagione  
 @staticmethod  
 def getMenu2():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT c.constructorId, c.name  
 FROM constructorresults cr  
 JOIN races r ON cr.raceId = r.raceId  
 JOIN constructors c ON cr.constructorId = c.constructorId  
 WHERE r.year = 2000  
 ORDER BY c.name"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi i circuiti successivi ad una determinata stagione  
 @staticmethod  
 def getMenu3():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT c.circuitId, c.name  
 FROM races r  
 JOIN circuits c ON r.circuitId = c.circuitId  
 WHERE r.year >= 1990  
 ORDER BY c.name"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi tutte le nazionalità dei piloti  
 @staticmethod  
 def getMenu4():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT d.nationality  
 FROM results r  
 JOIN drivers d ON r.driverId = d.driverId  
 WHERE r.position IS NOT NULL  
 ORDER BY d.nationality"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi tutti i GP di una determinata stagione  
 @staticmethod  
 def getMenu5():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT raceId, name  
 FROM races  
 WHERE year = 2000  
 ORDER BY round"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi tutti i piloti all-time di una determinata squadra costruttori  
 @staticmethod  
 def getMenu6():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT d.driverId, d.forename, d.surname  
 FROM results r  
 JOIN drivers d ON r.driverId = d.driverId  
 WHERE r.constructorId = ?  
 ORDER BY d.surname, d.forename"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi tutti gli anni a cui ha partecipato un determinato pilota  
 @staticmethod  
 def getMenu7():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT ra.year  
 FROM results r  
 JOIN races ra ON r.raceId = ra.raceId  
 WHERE r.driverId = ?  
 ORDER BY ra.year"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi tutte le gare in cui un pilota ha ottenuto punti  
 @staticmethod  
 def getMenu8():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT r.raceId, ra.name  
 FROM results r  
 JOIN races ra ON r.raceId = ra.raceId  
 WHERE r.driverId = ? AND r.points > 0  
 ORDER BY ra.date"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi gli status che hanno avuto almeno 10 piloti con sto problema  
 @staticmethod  
 def getMenu9():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT s.statusId, s.status  
 FROM results r  
 JOIN status s ON r.statusId = s.statusId  
 WHERE r.position IS NULL  
 GROUP BY s.statusId, s.status  
 HAVING COUNT(\*) >= 10  
 ORDER BY COUNT(\*) DESC"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi le squadre costruttori che hanno vinto almeno una gara  
 @staticmethod  
 def getMenu10():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT c.constructorId, c.name  
 FROM constructorresults cs  
 JOIN constructors c ON cs.constructorId = c.constructorId  
 WHERE cs.position = 1  
 ORDER BY c.name"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi le squadre costruttori che hanno vinto il campionato  
 @staticmethod  
 def getMenu11():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH ultimi\_round AS (  
 SELECT year, MAX(round) AS max\_round  
 FROM races  
 GROUP BY year  
 )  
 SELECT DISTINCT  
 r.year AS anno,  
 cs.constructorId AS constructorId  
 FROM races r  
 JOIN constructorStandings cs  
 ON r.raceId = cs.raceId  
 JOIN ultimi\_round ur  
 ON r.year = ur.year AND r.round = ur.max\_round  
 WHERE cs.position = 1"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # prendi i piloti che hanno vinto almeno una volta un determinato circuito  
 @staticmethod  
 def getMenu11():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT d.driverId, d.forename, d.surname  
 FROM results r  
 JOIN races ra ON r.raceId = ra.raceId  
 JOIN drivers d ON r.driverId = d.driverId  
 WHERE r.position = 1 AND ra.circuitId = %s"""  
  
 cursor.execute(query)  
  
 for row in cursor:  
 result.append(Oggetto(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # QUERY NODI E ARCHI SUL DATABASE FORMULA1 #  
  
 @staticmethod  
 def getAllNodes(year):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select distinct(r.driverId), d.forename, d.surname  
 from results r, races ra, drivers d  
 where r.raceId = ra.raceId and ra.`year` = %s and r.`position` > 0 and d.driverId = r.driverId"""  
  
 cursor.execute(query, (year,))  
  
 for row in cursor:  
 result.append(Node(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 @staticmethod  
 def getAllEdges(year):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select r1.driverId as driverId1, r2.driverId as driverId2, count(\*) as peso  
 from results r1, results r2, races ra  
 where r1.raceId = ra.raceId and r1.raceId = r2.raceId and ra.`year` = %s and r1.driverId != r2.driverId and r1.position > r2.position and r2.position > 0  
 group by r1.driverId, r2.driverId"""  
  
 cursor.execute(query, (year,))  
  
 for row in cursor:  
 result.append(Edge(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se hanno condiviso nella stessa stagione la stessa posizione finale in gara  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges2(year):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select least(r1.driverId, r2.driverId) as driverId1, greatest(r1.driverId, r2.driverId) as driverId2, count(\*) as peso  
 from results r1, results r2, races ra1, races ra2  
 where r1.driverId != r2.driverId and r1.position = r2.position AND r1.position IS NOT null and  
 ra1.raceId = r1.raceId and ra2.raceId = r2.raceId and ra1.year = ra2.year and ra1.year = %s  
 group by driverId1, driverId2"""  
  
 cursor.execute(query, (year,))  
  
 # for row in cursor:  
 # result.append(Edge2(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti partono dalla stessa posizione in gare diverse  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges3():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(r1.driverId, r2.driverId) AS d1,  
 GREATEST(r1.driverId, r2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM results r1  
 JOIN results r2 ON r1.grid = r2.grid AND r1.driverId < r2.driverId  
 WHERE r1.grid IS NOT NULL  
 GROUP BY d1, d2  
 """  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge3(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due costruttori hanno gareggiato nella stessa stagione  
 @staticmethod  
 def getAllEdges4():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT r1.constructorId AS c1, r2.constructorId AS c2  
 FROM results r1  
 JOIN results r2 ON r1.raceId = r2.raceId  
 JOIN races ra ON r1.raceId = ra.raceId  
 WHERE r1.constructorId < r2.constructorId"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge4(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti se hanno stessi millisecondi in best lap nella stessa gara in un determinato anno  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges5():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(l1.driverId, l2.driverId) AS d1,  
 GREATEST(l1.driverId, l2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM races r, laptimes l1  
 JOIN laptimes l2 ON l1.raceId = l2.raceId AND l1.driverId < l2.driverId  
 WHERE l1.milliseconds = l2.milliseconds and r.raceId = l1.raceId and r.year = 2000  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge5(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due costruttori hanno gareggiato nella stessa stagione  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges6():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(r1.constructorId, r2.constructorId) AS c1,  
 GREATEST(r1.constructorId, r2.constructorId) AS c2,  
 COUNT(DISTINCT ra.circuitId) AS peso  
 FROM results r1  
 JOIN results r2 ON r1.raceId = r2.raceId AND r1.constructorId < r2.constructorId  
 JOIN races ra ON r1.raceId = ra.raceId  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge6(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati da quello ha fatto più sorpassi verso chi ne ha subiti in una determinata stagione  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges7():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH sorpassi AS (  
 SELECT r1.driverId AS sorpassante, r2.driverId AS sorpassato  
 FROM races r, results r1  
 JOIN results r2 ON r1.raceId = r2.raceId  
 WHERE r1.grid > r2.grid AND r1.position < r2.position and r.raceId = r1.raceId and r.year = 2000  
 )  
 SELECT sorpassante, sorpassato, COUNT(\*) AS peso  
 FROM sorpassi  
 GROUP BY sorpassante, sorpassato"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge7(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono legati se hanno effettuato pitstop nello stesso giro e nella stessa gara  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges8():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(p1.driverId, p2.driverId) AS d1,  
 GREATEST(p1.driverId, p2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM pitstops p1  
 JOIN pitstops p2 ON p1.raceId = p2.raceId AND p1.lap = p2.lap  
 WHERE p1.driverId < p2.driverId  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge8(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono legati se hanno ricevuto la stessa penalità nella stessa gara  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges9():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(r1.driverId, r2.driverId) AS d1,  
 GREATEST(r1.driverId, r2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM results r1  
 JOIN results r2 ON r1.raceId = r2.raceId AND r1.driverId < r2.driverId  
 WHERE r1.statusId = r2.statusId AND r1.statusId NOT IN (1) -- 1 = "Finished"  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge9(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due costruttori sono legati da chi ha avuto più punti verso chi ne ha avuti meno nella stessa gara  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges10():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 cs1.constructorId AS vincente,  
 cs2.constructorId AS sconfitto,  
 COUNT(\*) AS peso  
 FROM constructorstandings cs1  
 JOIN constructorstandings cs2   
 ON cs1.raceId = cs2.raceId AND cs1.constructorId <> cs2.constructorId  
 WHERE cs1.points > cs2.points  
 GROUP BY vincente, sconfitto"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge10(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due costruttori sono legati se hanno preso lo stesso punteggio nelle stesse gare  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges11():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(cr1.constructorId, cr2.constructorId) AS c1,  
 GREATEST(cr1.constructorId, cr2.constructorId) AS c2,  
 COUNT(\*) AS peso  
 FROM constructorresults cr1  
 JOIN constructorresults cr2   
 ON cr1.raceId = cr2.raceId AND cr1.constructorId < cr2.constructorId  
 WHERE cr1.points = cr2.points  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge11(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono legati se uno ha vinto sull'altro almeno 5 volte in almeno 3 circuiti diversi (quindi non sempre sullo stesso)  
 # il peso è il numero di occorrenze e il verso va dal vincente verso il perdente  
 @staticmethod  
 def getAllEdges12():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select re1.driverId as driverId1, re2.driverId as driverId2, count(\*) as volte  
 from races r, results re1, results re2  
 where r.raceId = re1.raceId and r.`year` = 2009 and re1.raceId = re2.raceId and re1.driverId != re2.driverId and re1.`position` > re2.`position`   
 group by re1.driverId, re2.driverId  
 having count(\*) > 4 and count(distinct r.circuitId) > 2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge12(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono legati se hanno gareggiato sullo stesso circuito almeno una volta  
 # il peso è il numero di circuiti in comune  
 @staticmethod  
 def getAllEdges13():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select distinct least(re1.driverId,re2.driverId) as driv1, greatest(re1.driverId,re2.driverId) as driv2, count(DISTINCT ra.circuitId) as peso  
 from results re1, results re2, races ra  
 where re1.raceId = ra.raceId and re2.raceId = ra.raceId and re1.driverId <> re2.driverId  
 group by re1.driverId, re2.driverId"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge13(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono legati da chi ha fatto un tempo migliore minore verso quello maggiore nella stessa gara  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges14():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH best\_laps AS (  
 SELECT raceId, driverId, MIN(milliseconds) AS best  
 FROM laptimes  
 GROUP BY raceId, driverId  
 )  
 SELECT   
 b1.driverId AS source,  
 b2.driverId AS target,  
 COUNT(\*) AS peso  
 FROM best\_laps b1  
 JOIN best\_laps b2 ON b1.raceId = b2.raceId  
 WHERE b1.driverId <> b2.driverId AND b1.best < b2.best  
 GROUP BY source, target"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge14(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono legati se hanno fatto parte della stessa squadra in almeno una gara  
 # il peso è il numero di gare nelle stessa squadra  
 @staticmethod  
 def getAllEdges15():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(r1.driverId, r2.driverId) AS d1,  
 GREATEST(r1.driverId, r2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM results r1  
 JOIN results r2 ON r1.raceId = r2.raceId  
 WHERE r1.constructorId = r2.constructorId AND r1.driverId < r2.driverId  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge15(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due circuiti sono legati se hanno il vincitore uguale  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges16():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH vincitori AS (  
 SELECT ra.circuitId, r.driverId  
 FROM results r  
 JOIN races ra ON r.raceId = ra.raceId  
 WHERE r.position = 1  
 )  
 SELECT   
 LEAST(v1.circuitId, v2.circuitId) AS c1,  
 GREATEST(v1.circuitId, v2.circuitId) AS c2,  
 COUNT(\*) AS peso  
 FROM vincitori v1  
 JOIN vincitori v2 ON v1.driverId = v2.driverId AND v1.circuitId < v2.circuitId  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge16(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati da uno verso quello che ha sempre battuto  
 @staticmethod  
 def getAllEdges17():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT r1.driverId AS source, r2.driverId AS target  
 FROM results r1  
 JOIN results r2 ON r1.raceId = r2.raceId  
 WHERE r1.driverId <> r2.driverId AND r1.position < r2.position  
 AND NOT EXISTS (  
 SELECT 1  
 FROM results rx1  
 JOIN results rx2 ON rx1.raceId = rx2.raceId  
 WHERE rx1.driverId = r2.driverId AND rx2.driverId = r1.driverId  
 AND rx1.position < rx2.position)"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge17(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due costruttori sono collegati da uno verso quello che ha battuto per un determinato anno  
 # il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges18():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select c1.constructorId as con1, c2.constructorId as con2, count(\*) as peso  
 from constructorresults cr1, constructorresults cr2, constructors c1, constructors c2, races r  
 where cr1.constructorId = c1.constructorId and cr2.constructorId = c2.constructorId and cr1.raceId = r.raceId and cr1.raceId = cr2.raceId  
 and r.`year` = 2000 and c1.constructorId <> c2.constructorId and cr1.points > cr2.points  
 group by c1.constructorId, c2.constructorId"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge18(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se hanno terminato almeno una gara entrambi durante la stagione  
 # il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges19():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select least(r1.driverId,r2.driverId) as dr1, greatest(r1.driverId,r2.driverId) as dr2, count(\*) as peso  
 from results r1, results r2, races ra  
 where r1.raceId = r2.raceId and r1.driverId > r2.driverId and r1.position IS NOT null and r2.position IS NOT null and r1.raceId = ra.raceId and ra.year = 2000  
 group by r1.driverId, r2.driverId"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge19(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati dal pilota con la migliore posizione in qualifica verso quello peggiore nella stessa gara per la stagione di un anno  
 # il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges20():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select q1.driverId as dr1, q2.driverId as dr2, count(\*) as peso  
 from qualifying q1, qualifying q2, races r  
 where q1.raceId = q2.raceId and q1.`position` < q2.`position` and q1.driverId <> q2.driverId and r.raceId = q1.raceId and r.`year` = 2000  
 group by q1.driverId, q2.driverId"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge20(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati dal pilota con la migliore media pitstop verso quello peggiore nella stessa gara  
 @staticmethod  
 def getAllEdges21():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT p1.driverId AS vincente, p2.driverId AS sconfitto  
 FROM (  
 SELECT raceId, driverId, AVG(milliseconds) AS tempo\_totale  
 FROM pitstops  
 GROUP BY raceId, driverId  
 ) p1  
 JOIN (  
 SELECT raceId, driverId, AVG(milliseconds) AS tempo\_totale  
 FROM pitstops  
 GROUP BY raceId, driverId  
 ) p2 ON p1.raceId = p2.raceId and p1.raceId = 841 AND p1.driverId <> p2.driverId  
 WHERE p1.tempo\_totale < p2.tempo\_totale"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge21(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se sono arrivati entrambi al podio nella stessa gara  
 # il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges22():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select r1.driverId as dr1, r2.driverId as dr2, count(\*) as peso  
 from results r1, results r2  
 where r1.raceId = r2.raceId and r1.driverId < r2.driverId and r1.position < 4 and r2.position < 4  
 group by r1.driverId, r2.driverId"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge22(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due circuiti sono collegati da circuito A verso circuito B indica che un certo pilota ha ottenuto risultati migliori.  
 # il peso è la differenza di successi  
 # una volta presi i risultati vanno ciclati per tutte le coppie possibili per vedere gli archi da creare  
 @staticmethod  
 def getAllEdges23():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT ra.circuitId, COUNT(\*) AS vittorie  
 FROM results r  
 JOIN races ra ON r.raceId = ra.raceId  
 WHERE r.driverId = 27 AND r.position = 1  
 GROUP BY ra.circuitId"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge23(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due nazionalità sono collegate se almeno un pilota per ciascuna ha corso nella stessa gara  
 # Il peso è la somma dei punti guadagnati da piloti delle due nazionalità in tutte le gare in comune  
 @staticmethod  
 def getAllEdges24():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH punti\_per\_pilota AS (  
 SELECT r.raceId, d.driverId, d.nationality, r.points  
 FROM results r  
 JOIN drivers d ON r.driverId = d.driverId  
 WHERE r.points IS NOT NULL),  
 naz\_per\_gara AS (  
 SELECT p1.raceId, p1.nationality AS naz1, p2.nationality AS naz2,  
 SUM(p1.points) + SUM(p2.points) AS peso  
 FROM punti\_per\_pilota p1  
 JOIN punti\_per\_pilota p2   
 ON p1.raceId = p2.raceId AND p1.nationality < p2.nationality  
 GROUP BY p1.raceId, p1.nationality, p2.nationality),  
 grafo\_finale AS (  
 SELECT naz1, naz2, SUM(peso) AS peso\_totale  
 FROM naz\_per\_gara  
 GROUP BY naz1, naz2)  
 SELECT \* FROM grafo\_finale  
 ORDER BY peso\_totale DESC"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge24(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se si sono ritirati nello stesso modo in almeno 2 gare consecutive  
 # Il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges25():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH ritiri AS (  
 SELECT r.driverId, ra.round, ra.year, r.statusId  
 FROM results r  
 JOIN races ra ON r.raceId = ra.raceId  
 WHERE r.position IS NULL)  
 SELECT   
 LEAST(r1.driverId, r2.driverId) AS d1,  
 GREATEST(r1.driverId, r2.driverId) AS d2  
 FROM ritiri r1  
 JOIN ritiri r2 ON r1.year = r2.year and r1.year = 2000  
 AND ABS(r1.round - r2.round) = 1   
 AND r1.driverId < r2.driverId   
 AND r1.statusId = r2.statusId  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge25(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due squadre costruttori sono collegati se si sono scambiati posizioni nella stessa stagione  
 # A batte B in alcune gare, ma B ha battuto A in altre. Mostra le coppie “equilibrate” con bilanci vicini a zero.  
 # Il peso è il numero delle occorrenze in un determinato anno  
 @staticmethod  
 def getAllEdges26():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH confronti AS (  
 SELECT   
 LEAST(c1.constructorId, c2.constructorId) AS c1,  
 GREATEST(c1.constructorId, c2.constructorId) AS c2,  
 SUM(CASE WHEN c1.points > c2.points THEN 1 WHEN c2.points > c1.points THEN -1 ELSE 0 END) AS score  
 FROM races r, constructorresults c1  
 JOIN constructorresults c2   
 ON c1.raceId = c2.raceId AND c1.constructorId < c2.constructorId  
 where c1.raceId = r.raceId and r.year = 1988  
 GROUP BY c1, c2  
 )  
 SELECT \* FROM confronti  
 WHERE ABS(score) <= 2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge26(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se si sono ritirati nello stesso modo in almeno 2 gare consecutive  
 # Il peso è il numero delle occorrenze in un determinato anno  
 @staticmethod  
 def getAllEdges27():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH ritiri AS (  
 SELECT r.driverId, ra.round, ra.year, r.statusId  
 FROM results r  
 JOIN races ra ON r.raceId = ra.raceId  
 WHERE r.position IS NULL  
 )  
 SELECT   
 LEAST(r1.driverId, r2.driverId) AS d1,  
 GREATEST(r1.driverId, r2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM ritiri r1  
 JOIN ritiri r2 ON r1.year = r2.year   
 and r1.year = 2000  
 AND r1.round = r2.round   
 AND r1.driverId < r2.driverId   
 AND r1.statusId = r2.statusId  
 GROUP BY d1, d2  
 HAVING peso >= 2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge27(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se hanno lo stesso numero di posizioni guadagnate dalla partenza nella stessa gara  
 # Il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges28():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH guadagni AS (  
 SELECT r.driverId, r.raceId, (r.grid - r.position) AS delta  
 FROM results r  
 WHERE r.grid IS NOT NULL AND r.position IS NOT NULL  
 )  
 SELECT   
 LEAST(g1.driverId, g2.driverId) AS d1,  
 GREATEST(g1.driverId, g2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM guadagni g1  
 JOIN guadagni g2   
 ON g1.raceId = g2.raceId   
 AND g1.driverId < g2.driverId   
 AND g1.delta = g2.delta  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge28(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se sono usciti nella stessa gara di qualificazione nello stesso Q (q1, q2, q3)  
 # Il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges29():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(q1.driverId, q2.driverId) AS d1,  
 GREATEST(q1.driverId, q2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM qualifying q1  
 JOIN qualifying q2   
 ON q1.raceId = q2.raceId   
 AND q1.driverId < q2.driverId  
 WHERE (  
 q1.q1 IS NOT NULL AND q2.q1 IS NOT NULL AND q1.q1 = q2.q1) OR (  
 q1.q2 IS NOT NULL AND q2.q2 IS NOT NULL AND q1.q2 = q2.q2) OR (  
 q1.q3 IS NOT NULL AND q2.q3 IS NOT NULL AND q1.q3 = q2.q3)  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge29(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due gare sono collegate se hanno tagliato il traguardo lo stesso numero di piloti  
 # Il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges30():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH arrivi AS (  
 SELECT raceId, COUNT(\*) AS num\_arrivati  
 FROM results  
 WHERE position IS NOT NULL  
 GROUP BY raceId  
 )  
 SELECT   
 LEAST(a1.raceId, a2.raceId) AS r1,  
 GREATEST(a1.raceId, a2.raceId) AS r2,  
 1 AS peso  
 FROM arrivi a1  
 JOIN arrivi a2   
 ON a1.raceId < a2.raceId AND a1.num\_arrivati = a2.num\_arrivati"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge30(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due costruttori sono collegati se hanno effettuato lo stesso numero di pitstop totali nella stessa gara  
 # Il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges31():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH pit AS (  
 SELECT r.raceId, r.constructorId, COUNT(\*) AS num\_pit  
 FROM results r  
 JOIN pitstops p ON r.driverId = p.driverId AND r.raceId = p.raceId  
 GROUP BY r.raceId, r.constructorId  
 )  
 SELECT   
 LEAST(p1.constructorId, p2.constructorId) AS c1,  
 GREATEST(p1.constructorId, p2.constructorId) AS c2,  
 COUNT(\*) AS peso  
 FROM pit p1  
 JOIN pit p2   
 ON p1.raceId = p2.raceId   
 AND p1.constructorId < p2.constructorId   
 AND p1.num\_pit = p2.num\_pit  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge31(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Due piloti sono collegati se hanno registrato il miglior tempo nello stesso giro in gara  
 # Il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges32():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """WITH best\_laps AS (  
 SELECT driverId, raceId, MIN(milliseconds) AS best\_time, lap  
 FROM laptimes  
 GROUP BY driverId, raceId  
 )  
 SELECT   
 LEAST(b1.driverId, b2.driverId) AS d1,  
 GREATEST(b1.driverId, b2.driverId) AS d2,  
 COUNT(\*) AS peso  
 FROM best\_laps b1  
 JOIN best\_laps b2   
 ON b1.raceId = b2.raceId   
 AND b1.driverId < b2.driverId  
 AND b1.lap = b2.lap  
 GROUP BY d1, d2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge32(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # Un pilota è collegato verso un altro se ha avuto posizione migliore sia in griglia che a fine gara  
 # Il peso è il numero delle occorrenze  
 @staticmethod  
 def getAllEdges33():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 r1.driverId AS source,  
 r2.driverId AS target,  
 COUNT(\*) AS peso  
 FROM results r1  
 JOIN results r2   
 ON r1.raceId = r2.raceId   
 AND r1.driverId <> r2.driverId  
 WHERE   
 r1.grid < r2.grid AND  
 r1.position < r2.position AND  
 r1.grid IS NOT NULL AND r2.grid IS NOT NULL AND  
 r1.position IS NOT NULL AND r2.position IS NOT NULL  
 GROUP BY source, target"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge33(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
  
 # QUERY SUL DATABASE BIKE\_STORE\_FULL #  
  
 @staticmethod  
 def getAllEdges(giorni, store\_id):  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """select o1.order\_id as order\_id\_1, o2.order\_id as order\_id\_2,  
 (count(oi1.order\_id) + count(oi2.order\_id)) as peso, DATEDIFF(o1.order\_date, o2.order\_date) as verso  
 from orders o1, orders o2, order\_items oi1, order\_items oi2   
 where abs(datediff(o1.order\_date , o2.order\_date)) < %s and o1.store\_id = o2.store\_id and o1.order\_id != o2.order\_id and o1.store\_id =%s  
 and oi1.order\_id = o1.order\_id and oi2.order\_id = o2.order\_id  
 group by oi1.order\_id, oi2.order\_id  
 having verso > 0"""  
  
 cursor.execute(query, (giorni, store\_id,))  
  
 for row in cursor:  
 result.append(Edge(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono i clienti e sono collegati se hanno almeno un ordine nello stesso mese  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges2():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT  
 LEAST(o1.customer\_id, o2.customer\_id) AS c1,  
 GREATEST(o1.customer\_id, o2.customer\_id) AS c2,  
 COUNT(\*) AS peso  
 FROM orders o1  
 JOIN orders o2 ON o1.customer\_id < o2.customer\_id  
 WHERE MONTH(o1.order\_date) = MONTH(o2.order\_date)  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge2(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono i prodotti e sono collegati se sono nello stesso ordine  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges3():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT  
 LEAST(oi1.product\_id, oi2.product\_id) AS p1,  
 GREATEST(oi1.product\_id, oi2.product\_id) AS p2,  
 COUNT(\*) AS peso  
 FROM order\_items oi1  
 JOIN order\_items oi2 ON oi1.order\_id = oi2.order\_id  
 WHERE oi1.product\_id < oi2.product\_id  
 GROUP BY p1, p2  
 """  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge3(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono le persone dello staff e sono collegati se hanno ordini nello stesso giorno  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges4():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT DISTINCT  
 LEAST(o1.staff\_id, o2.staff\_id) AS s1,  
 GREATEST(o1.staff\_id, o2.staff\_id) AS s2  
 FROM orders o1  
 JOIN orders o2 ON o1.staff\_id < o2.staff\_id  
 WHERE o1.order\_date = o2.order\_date"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge4(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono le categorie e sono collegate se sono presenti nello stesso ordine  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges6():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT  
 LEAST(p1.category\_id, p2.category\_id) AS c1,  
 GREATEST(p1.category\_id, p2.category\_id) AS c2,  
 COUNT(DISTINCT oi1.order\_id) AS peso  
 FROM order\_items oi1  
 JOIN order\_items oi2 ON oi1.order\_id = oi2.order\_id  
 JOIN products p1 ON oi1.product\_id = p1.product\_id  
 JOIN products p2 ON oi2.product\_id = p2.product\_id  
 WHERE p1.category\_id < p2.category\_id  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge6(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono gli ordini di uno stesso cliente e sono collegati se uno dopo l'altro  
 # il peso è il numero di giorni che intercorre tra i due  
 @staticmethod  
 def getAllEdges7():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT o1.order\_id AS source, o2.order\_id AS target,  
 DATEDIFF(o2.order\_date, o1.order\_date) AS peso  
 FROM orders o1  
 JOIN orders o2 ON o1.customer\_id = o2.customer\_id AND o1.order\_date < o2.order\_date"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge7(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono i prodotti e sono collegati se sono stati ordinati almeno una volta nello stesso store  
 # il peso è il numero di store in cui accade  
 @staticmethod  
 def getAllEdges8():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(oi1.product\_id, oi2.product\_id) AS p1,  
 GREATEST(oi1.product\_id, oi2.product\_id) AS p2,  
 COUNT(DISTINCT o1.store\_id) AS peso  
 FROM order\_items oi1  
 JOIN orders o1 ON oi1.order\_id = o1.order\_id  
 JOIN order\_items oi2 ON o1.order\_id = oi2.order\_id  
 WHERE oi1.product\_id < oi2.product\_id  
 GROUP BY p1, p2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge8(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono i clienti e sono collegati se hanno acquistato gli stessi prodotti  
 # il peso è il numero di prodotti uguali ordinati  
 @staticmethod  
 def getAllEdges9():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(o1.customer\_id, o2.customer\_id) AS c1,  
 GREATEST(o1.customer\_id, o2.customer\_id) AS c2,  
 COUNT(DISTINCT oi1.product\_id) AS peso  
 FROM orders o1  
 JOIN order\_items oi1 ON o1.order\_id = oi1.order\_id  
 JOIN orders o2 ON o1.customer\_id < o2.customer\_id  
 JOIN order\_items oi2 ON o2.order\_id = oi2.order\_id  
 WHERE oi1.product\_id = oi2.product\_id  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge9(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono le categorie e sono collegate se sono state ordinate insieme in ordini diversi  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges10():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(p1.category\_id, p2.category\_id) AS c1,  
 GREATEST(p1.category\_id, p2.category\_id) AS c2,  
 COUNT(\*) AS peso  
 FROM order\_items oi1  
 JOIN order\_items oi2 ON oi1.order\_id = oi2.order\_id AND oi1.item\_id < oi2.item\_id  
 JOIN products p1 ON oi1.product\_id = p1.product\_id  
 JOIN products p2 ON oi2.product\_id = p2.product\_id  
 WHERE p1.category\_id <> p2.category\_id  
 GROUP BY c1, c2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge10(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono i prodotti e sono collegati se sono stati ordinati insieme in ordini diversi  
 # il peso è il numero di occorrenze che deve essere di base almeno 10 e meno di 150  
 @staticmethod  
 def getAllEdges11():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(p1.category\_id, p2.category\_id) AS c1,  
 GREATEST(p1.category\_id, p2.category\_id) AS c2,  
 COUNT(\*) AS peso  
 FROM order\_items oi1  
 JOIN order\_items oi2 ON oi1.order\_id = oi2.order\_id AND oi1.item\_id < oi2.item\_id  
 JOIN products p1 ON oi1.product\_id = p1.product\_id  
 JOIN products p2 ON oi2.product\_id = p2.product\_id  
 WHERE p1.category\_id <> p2.category\_id  
 GROUP BY c1, c2  
 HAVING COUNT(\*) > 10 and COUNT(\*) < 150"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge11(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono i clienti e sono collegati se sono hanno fatto ordini nello stesso giorno  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges12():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT distinct LEAST(o1.customer\_id, o2.customer\_id) AS s1, GREATEST(o1.customer\_id, o2.customer\_id) AS s2, count(distinct o1.order\_date) as peso  
 FROM orders o1, orders o2  
 WHERE o1.order\_date = o2.order\_date and o1.customer\_id < o2.customer\_id and o1.order\_id <> o2.order\_id  
 group by o1.customer\_id, o2.customer\_id"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge12(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result  
  
 # I nodi sono lo staff e sono collegati se hanno servito gli stessi clienti  
 # il peso è il numero di occorrenze  
 @staticmethod  
 def getAllEdges13():  
 conn = DBConnect.get\_connection()  
  
 result = []  
  
 cursor = conn.cursor(dictionary=True)  
  
 query = """SELECT   
 LEAST(o1.staff\_id, o2.staff\_id) AS s1,  
 GREATEST(o1.staff\_id, o2.staff\_id) AS s2,  
 COUNT(DISTINCT o1.customer\_id) AS peso  
 FROM orders o1  
 JOIN orders o2 ON o1.customer\_id = o2.customer\_id AND o1.staff\_id < o2.staff\_id  
 GROUP BY s1, s2"""  
  
 cursor.execute(query)  
  
 # for row in cursor:  
 # result.append(Edge13(\*\*row))  
  
 cursor.close()  
 conn.close()  
 return result