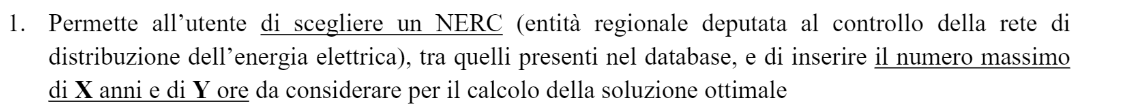
**QUERY SQL**

1. **RICHIESTA:**

****

"""select \* from poweroutages p

where p.nerc\_id = %s

order by p.date\_event\_began """

in questo caso non abbiamo messo i vincoli nella query e abbiamo ordinato per data.

1. **RICHIESTA:**

Questo metodo legge le rotte dal db, aggregando già nella query i voli opposti sulla stessa tratta. Per fare questo, la query è abbastanza complicata, e sfrutta una join di due tabelle temporanee (una il flip

dell'altra). Visto che alcune rotte non hanno voli in entrambe le direzioni, si fanno dei check sui NULL (che non vanno esclusi). COALESCE permette nella sommatoria di considerare i NULL come zero.

"""SELECT T1.ORIGIN\_AIRPORT\_ID as a1, T1.DESTINATION\_AIRPORT\_ID as a2, COALESCE(T1.D, 0) + COALESCE(T2.D, 0) as totDistance, COALESCE(T1.N, 0) + COALESCE(T2.N, 0) as nVoli

FROM

(SELECT f.ORIGIN\_AIRPORT\_ID, f.DESTINATION\_AIRPORT\_ID, SUM(f.DISTANCE) as D, COUNT(\*) as N

FROM flights f

GROUP BY f.ORIGIN\_AIRPORT\_ID, f.DESTINATION\_AIRPORT\_ID) T1

LEFT JOIN

(SELECT f.ORIGIN\_AIRPORT\_ID, f.DESTINATION\_AIRPORT\_ID, SUM(f.DISTANCE) as D, COUNT(\*) as N

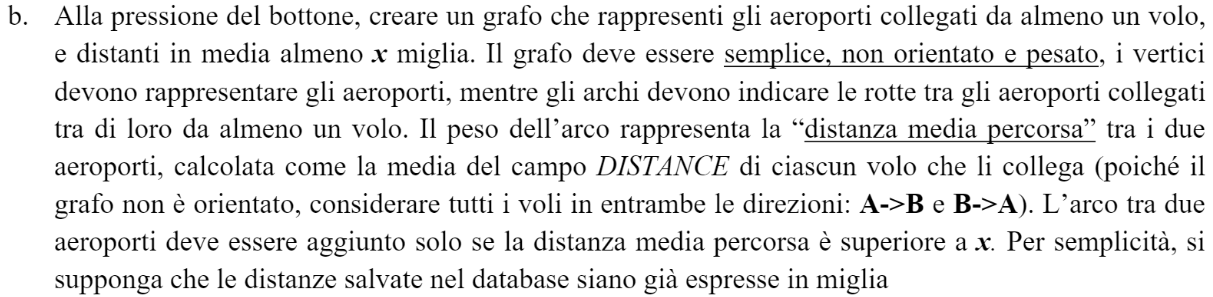
FROM flights f

GROUP BY f.ORIGIN\_AIRPORT\_ID, f.DESTINATION\_AIRPORT\_ID) T2

ON T1.ORIGIN\_AIRPORT\_ID = T2.DESTINATION\_AIRPORT\_ID AND T2.ORIGIN\_AIRPORT\_ID = T1.DESTINATION\_AIRPORT\_ID

WHERE T1.ORIGIN\_AIRPORT\_ID < T2.ORIGIN\_AIRPORT\_ID OR T2.ORIGIN\_AIRPORT\_ID IS NULL OR T2.DESTINATION\_AIRPORT\_ID IS NULL"""

1. **RICHIESTA:**

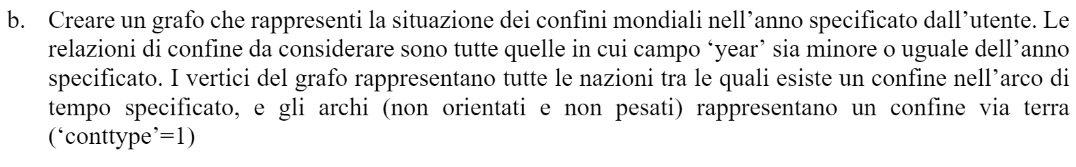
****

"""SELECT f.ORIGIN\_AIRPORT\_ID as a1, f.DESTINATION\_AIRPORT\_ID as a2, SUM(f.DISTANCE) as totDistance, COUNT(\*) as nVoli

FROM flights f

GROUP BY f.ORIGIN\_AIRPORT\_ID, f.DESTINATION\_AIRPORT\_ID"""

1. **RICHIESTA:**

****

"""SELECT co.StateAbb, co.CCode, co.StateNme

from contiguity c, country co

where c.`year` <= %s

and c.state1no = co.CCode

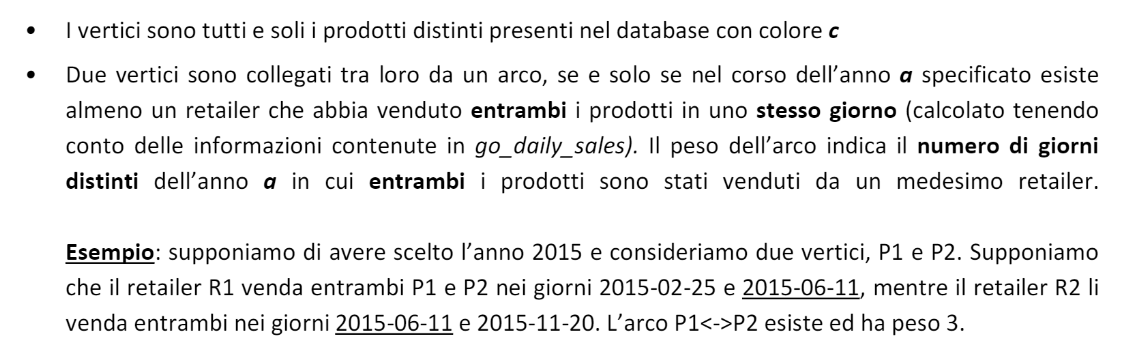
group by c.state1no ORDER BY StateAbb"""

"select state1no, state2no

from contiguity

where contiguity.conttype=1 and contiguity.year <= %s"

1. **RICHIESTA:**

****

'''SELECT COUNT(DISTINCT s1.Date) as N

FROM go\_daily\_sales s1, go\_daily\_sales s2

WHERE s1.Date = s2.Date

AND s1.Retailer\_code = s2.Retailer\_code

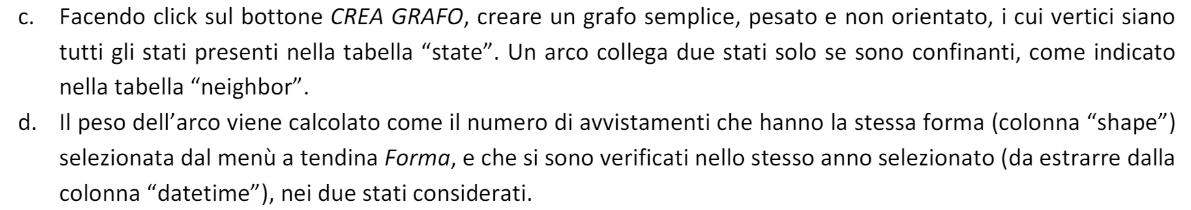
AND s1.Product\_Number = %s

AND s2.Product\_Number = %s

AND YEAR(s1.Date) = %s'''

**QUERY DI POSSIBILI DB D’ESAME**

**UFO SIGHTINGS – RICHIESTA DA SIMULAZIONE**

****

**Esempio DAO UFO**

from database.DB\_connect import DBConnect

from model.States import State

from model.Sighting import Sighting

class DAO():

def \_\_init\_\_(self):

pass

@staticmethod

def getAllStates():

conn = DBConnect.get\_connection()

result = []

cursor = conn.cursor(dictionary=True)

query = """select \* from state s """

cursor.execute(query)

for row in cursor:

result.append(State(row["id"], row["Name"], row["Capital"], row["Lat"], row["Lng"], row["Area"], row["Population"], row["Neighbors"]))

cursor.close()

conn.close()

return result

@staticmethod

def getAllSighting():

conn = DBConnect.get\_connection()

result = []

cursor = conn.cursor(dictionary=True)

query = """select \* from sighting s order by `datetime` asc """

cursor.execute(query)

for row in cursor:

result.append(Sighting(\*\*row))

cursor.close()

conn.close()

return result

@staticmethod

def getAllShapes():

conn = DBConnect.get\_connection()

result = []

cursor = conn.cursor(dictionary=True)

query = """select distinct shape from sighting s

where shape != "" """

cursor.execute(query)

for row in cursor:

result.append(row['shape'])

cursor.close()

conn.close()

return result

@staticmethod

def getAllWeightedNeigh(year,shape):

conn = DBConnect.get\_connection()

result = []

cursor = conn.cursor(dictionary=True)

query = **"""SELECT n.state1, n.state2 , count(\*) as N**

**FROM sighting s , neighbor n**

**where year(s.`datetime`) = %s**

**and s.shape = %s**

**and (s.state = n.state1 or s.state = n.state2 )**

**and n.state1 < n.state2**

**group by n.state1 , n.state2 """**

cursor.execute(query, (year,shape))

for row in cursor:

result.append((row['state1'],row['state2'], row["N"]))

cursor.close()

conn.close()

return result

**QUERY UFO**

**select** s1.city , s2.city

**from** sighting s1, sighting s2

**where** s1.`datetime` = s2.`datetime`

**and** s1.city > s2.city

**and** s1.state = s2.state

**and** s1.shape = s2.shape

ARCHI- > Seleziono tutti gli avvistamenti nella stessa esatta data, giorno mese anno, nello stesso stato, con la stessa forma e città diverse.

**select** **count**(\*)

**from** sighting s

**where** s.state = "ca"

**and** s.shape = 'circle'

**and** **year**(s.`datetime` ) = 1999

Il peso dell’arco viene calcolato come la somma

del numero di avvistamenti verificatisi nei due

stati considerati, purché che si siano verificati al

massimo ad una distanza di xG giorni nell’anno selezionato (da estrarre dalla colonna “datetime”) da un

avvistamento nell’altro stato.

Utilizzare la funzione DATEDIFF di mysql per calcolare la differenza in giorni tra campi datetime.

Es. DATEDIFF("1964-06-15 13:00:00", "1964-04-15 22:00:00")

**select** **count**(**distinct** s1.state , s2.state) as peso

**from** sighting s1 , sighting s2

**where** **year**(s1.`datetime`) = 2010 **and** **year**(s2.`datetime`) = 2010

**and** **datediff**(s1.`datetime`, s2.`datetime`) <= 100

**and** s1.state < s2.state

**and** s1.state = 'NY' **and** s2.state = 'PA'

**RICORSIONE GENERICA**

def ricorsione(self, parziale):

if self.puoEssereSol(parziale): #ho raggiunto condizione terminale: non ho piu vicini validi da inserire

if self.soluzioneMigliore(parziale): #controllo se è la migliore

self.bestPath = copy.deepcopy(parziale) #copio

self.bestScore = self.calcolaPeso(parziale) #salvo il peso

return #vado alla pop per scrivere un'altra soluzione

for n in self.grafo.neighbors(parziale[-1]): #il prossimo deve essere collegato all'ultimo del parziale (è un cammimo)

if self.nodoValido(parziale,n): #controllo se il nodo rispetta i vincoli

parziale.append(n) #aggiunto

self.ricorsione(parziale) #richiamo

parziale.pop() #vado qui dopo le return