QUERY DB BIKE

1. trovare membri dello staff ATTIVI che lavorano in uno store scelto

```
SELECT sta.first_name, sta.last_name, sta.active as
staff_status, sto.store_name
FROM staffs as sta, stores as sto
WHERE sta.store_id = sto.store_id  and sta.active = 1 and
sto.store_name="Santa Cruz Bikes"
```

2. conta il numero di prodotti di ogni categoria (ogni categoria ha tot prodotti)

```
SELECT c.category_name as category_name, COUNT(*) as
num_product
FROM categories as c, products as p
WHERE p.category_id = c.category_id
GROUP BY category_name
```

3. calcola il NUMERO DI ORDINI fatti da ciascun costumer

```
SELECT  c.customer_id, c.first_name, c.last_name,
COUNT(*) as num_orders
FROM customers as c, orders as o
WHERE o.customer_id = c.customer_id
GROUP BY o.customer_id
```

4. calcola il total sale PER OGNI PRODOTTO (considerando quantità e sconti)
vendite totali = somma di (prezzo scontato × quantità) = SUM(list_price × (1 - discount) × quantity)

```
SELECT oi.product_id, p.product_name, SUM(oi.quantity) AS
total_quantity, SUM(oi.list_price * (1 - oi.discount) *
oi.quantity) AS total_sales_per_product
FROM order_items AS oi, products AS p
WHERE oi.product_id = p.product_id
GROUP BY oi.product_id;
```

5. Per ogni tipologia di order status (pending/completed ecc) voglio sapere quanti ordini ho all'interno

```
SELECT  order_status, COUNT(*) as num_orders
FROM orders
GROUP BY order_status
```

6. scrivere una query che mostri i costumers che hanno fatto ALMENO un ordine

```
SELECT c.customer_id, c.first_name, c.last_name, COUNT(*)
as num_order
FROM customers as c, orders as o
WHERE c.customer_id = o.customer_id
GROUP BY c.customer_id
```

7. query che trova la quantità totale PER OGNI PRODOTTO disponibile in ogni store

```
SELECT s.product_id, p.product_name, SUM(s.quantity) AS
total_q_in_all_stores
FROM stocks AS s, products AS p
WHERE s.product_id = p.product_id
GROUP BY s.product_id
```

8. Total gross revenue per ogni negozio

```
 SELECT st.store_id, st.store_name, st.state,
SUM(oi.list_price * (1 - oi.discount) * oi.quantity) AS
total_sales_per_store
FROM order_items AS oi,orders AS o, stores AS st
WHERE oi.order_id = o.order_id AND o.store_id =
st.store_id
GROUP BY st.store_id
```

9. trova i top 5 clienti che hanno speso più soldi

```
SELECT c.customer_id, c.first_name, c.last_name,
    SUM(oi.list_price * (1 - oi.discount) * oi.quantity)
AS total_spending
FROM customers c, orders o, order_items oi
WHERE c.customer_id = o.customer_id AND o.order_id =
oi.order_id
GROUP BY  c.customer_id
ORDER BY total_spending DESC
LIMIT 5;
```

10. numero di stocks per ogni categoria in ogni negozio

```
SELECT s.store_name, c.category_name, SUM(st.quantity) AS
sum_qty
FROM categories c, products p, stocks st, stores s
WHERE
    c.category_id = p.category_id
    AND p.product_id = st.product_id
    AND st.store_id = s.store_id
GROUP BY
    s.store_name, c.category_name;
```

11. quantità di items ordinati per ogni categoria nei vari store

```
SELECT store_name, category_name, SUM(quantity) AS
sum_qty
FROM categories c,products p, order_items oi, orders o
WHERE c.category_id = p.category_id AND p.product_id =
oi.product_id and oi.order_id = o.order_id
and  o.store_id = s.store_id
GROUP BY store_name, category_name
```

12. top 10 best-selling products

```
select p.product_name, count(*) as total_counts
(ot.quantity * ot.list_price) as revenue
from products p, order_items oi
where p.product_id= oi.product_id
group by p.product_name
order by total_counts desc, revenue desc
```

13. quale categoria genera le piu alte entrate?

```
select c.category_name, (oi.quantity * oi.list_price) as
revneue
from categories c, products p, order_items oi
where p.category_id=c.category_id
and oi. product_id= p.product_id
group by c.category_name
order by revneue desc
```

14. quale store ha le piu alte sales?

```
select store_name, (ot.quantity * ot.list_price) as
revneue
from stores s, orders o , order_items oi
where s.store_id=o.order and o.order_id=oi.order_id
group by store_name
order by revenue desc
```

```
SELECT  c.category_name as name # per riempire dropdown di categorie
FROM categories c
WHERE c.category_name !=""
group by c.category_name

SELECT p.* # nodi di quella categoria, in result.append fare ogni riga e mettere a 0 l'ultimo valore
FROM products p, categories c
WHERE p.category_id = c.category_id and
c.category_name = 'Cruisers Bicycles'
group by p.product_id

SELECT count(*) as numero # numero da salvare nella dataclass per volume di vendita (salvo in dataclass un valore)
FROM order_items oi
WHERE oi.product_id = 20
GROUP by oi.product_id  # restituisce il volume di vendta di quel nodo

# nel modello ciclo su ogni nodo che ho costruito e associo il valore mancante al valore che ho calcolato qui sopra

SELECT oi.product_id as id1, oi2.product_id as id2
FROM order_items oi, order_items oi2, orders o,
orders o2
WHERE oi.product_id > oi2.product_id and oi.order_id
= o.order_id and oi2.order_id = o2.order_id
and o2.order_date BETWEEN '2016/01/01' and
'2018/12/28' and o.order_date BETWEEN '2016/01/01'
and '2018/12/28'
GROUP by oi.product_id, oi2.product_id

#tutti gli archi compresi
```

```
def calcolaPuntiNodo(self, nodo):
    cntVittorie=0
    cntSconfitte=0
    for arco in self.grafo.out_edges(nodo, data=True):
        cntVittorie += arco[2]['weight']
```

```
    for arco2 in self.grafo.in_edges(nodo, data=True):
        cntSconfitte+=arco2[2]['weight']
    puntiNodo = cntVittorie-cntSconfitte
    return puntiNodo

def calcolaPuntiTutti(self):
    mappaNodoPunti={}
    for nodo in self.grafo.nodes():
        punteggio = self.calcolaPuntiNodo(nodo)
        mappaNodoPunti[nodo]=punteggio

    mappaSorted = dict(sorted(mappaNodoPunti.items(), key=lambda x: x[1], reverse=True))
    return next(iter(mappaSorted)), self.calcolaPuntiNodo(next(iter(mappaSorted)))
```

```
def getBestPath(self,nodoSourceId,nodoDestId,maxLen):
        self.bestPath = []
        self._costoMax = 0
        nodoSource=self.idMap[nodoSourceId]
        nodoDestId= self.idMap[nodoDestId]


        parziale=[nodoSource]
        self.ricorsione(parziale,maxLen,nodoDestId)
        return self.bestPath, self._costoMax



def ricorsione(self,parziale,maxLen, nodoDest):
    if (len(parziale)==maxLen and
parziale[-1]==nodoDest):

        if
self.calcolaCosto(parziale)>self._costoMax:

self._costoMax=self.calcolaCosto(parziale)
                self.bestPath=copy.deepcopy(parziale)
            return

vicini=self.prendiVicini(parziale[-1])
```

```
for nodo in vicini:
 if nodo not in parziale:
                parziale.append(nodo)
                self.ricorsione(parziale,maxLen,
nodoDest)
                parziale.pop()



def calcolaCosto(self, parziale):
     costo=0
     for i in range(0, len(parziale) - 1):
             costo += self._grafo[parziale[i]]
[parziale[i + 1]][
                "weight"]  # prendo il peso dell'arco
che connette i e i+1
        return costo



def prendiVicini(self, ultimoNodo):
        return
list(self._grafo.successors(ultimoNodo))
```