Schema fisico

workload

Per stimare un workload, pensando alle operazioni che possono essere fatte piu' spesso sulla base di dati, abbiamo ragionato principalmente sui prodotti, sui donatori (e quindi le donazioni) e sui turni di lavoro.

Principalmente, sui prodotti verranno effettuate molte operazioni di selezione e inserimento, molto raramente aggiornamenti e cancellazioni. Le selezioni verranno effettuate in generale per visualizzare tutti i prodotti, per controllare gli acquisti di un dato cliente, per vedere quali prodotti hanno superato la data di scadenza massima (e quindi sono da scaricae) ecc ecc.

Una condizione che abbiamo notato potrebbe dare problemi a livello di efficienza e' cercare, per esempio, tutti i prodotti da scaricare entro un anno.

```
-- Selezione di tutti i prodotti da scaricare entro un anno da oggi
SELECT *
FROM prodotti
WHERE data_scarico BETWEEN CURRENT_DATE AND CURRENT_DATE + '1 year'::interval;
```

lo schema fisico per questa interrogazione e' la seguente:

```
Seq Scan on prodotti  (cost=0.00..217.04 rows=7024 width=33) (actual time=0.006..1.122 rows=7024 loops=1)
  Filter: ((data_scarico >= CURRENT_DATE) AND (data_scarico <= (CURRENT_DATE + '1 year'::interval)))
Planning Time: 0.066 ms
Execution Time: 1.308 ms
```

Viene ovviamente eseguita una scansione sequenziale, perche' non e' presente nessun indice $I_{data_scarico}(prodotti)$. Eventualmente, sarebbe un indice ad albero, siccome la maggior parte delle selezioni di questo tipo sarebbero in un range.

Come seconda query, vogliamo recuperare la percentuale di prodotti donati per ogni tipologia e marca

```
SELECT s.tipologia, s.marca, round((COUNT(p.id)/(SELECT COUNT(*) FROM prodotti)::decimal(10,2) * 100), 2) 
FROM scorte s
NATURAL JOIN prodotti p
JOIN ingresso_prodotti ip on p.id_ingresso = ip.id
WHERE ip.importo_speso IS NULL
GROUP BY s.tipologia, s.marca
```

Come piano fisico, in questo caso e' il seguente

```
HashAggregate  (cost=675.33..677.08 rows=70 width=1064) (actual time=5.303..5.318 rows=20 loops=1)
"  Group Key: s.tipologia, s.marca"
  Batches: 1  Memory Usage: 24kB
  InitPlan 1 (returns $0)
    ->  Aggregate  (cost=146.80..146.81 rows=1 width=8) (actual time=0.559..0.560 rows=1 loops=1)
          ->  Seq Scan on prodotti  (cost=0.00..129.24 rows=7024 width=0) (actual time=0.015..0.341 rows=7
  ->  Hash Join  (cost=344.61..502.09 rows=3523 width=1036) (actual time=1.892..3.982 rows=3575 loops=1)
        Hash Cond: (p.codice_prodotto = s.codice_prodotto)
        ->  Hash Join  (cost=333.04..480.72 rows=3523 width=8) (actual time=1.862..3.382 rows=3575 loops=1
              Hash Cond: (p.id_ingresso = ip.id)
              ->  Seq Scan on prodotti p  (cost=0.00..129.24 rows=7024 width=12) (actual time=0.004..0.342
              ->  Hash  (cost=239.00..239.00 rows=7523 width=4) (actual time=1.848..1.849 rows=7523 loops=
                    Buckets: 8192  Batches: 1  Memory Usage: 329kB
                    ->  Seq Scan on ingresso_prodotti ip  (cost=0.00..239.00 rows=7523 width=4) (actual ti
                          Filter: (importo_speso IS NULL)
                          Rows Removed by Filter: 7477
        ->  Hash  (cost=10.70..10.70 rows=70 width=1036) (actual time=0.027..0.027 rows=20 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 10kB
              ->  Seq Scan on scorte s  (cost=0.00..10.70 rows=70 width=1036) (actual time=0.009..0.011 ro
Planning Time: 0.276 ms
Execution Time: 5.404 ms
```

e, infine, come ultima query, abbiamo ragionato sui turni e i volontari. Vogliamo vedere, per ogni volontario, il numero di colli totali che trasportera' (se esegue turni trasporti) nei prossimi 2 mesi

```
SELECT v.id, SUM(n_colli) as n_colli_trasportati
FROM volontari v
JOIN turni_trasporti tt on v.id = tt.volontario
JOIN turni t ON tt.id = t.id
WHERE t.data <= CURRENT_DATE + '2 months'::interval
GROUP BY v.id
```

piano fisico:

```
HashAggregate  (cost=1075.39..1150.39 rows=7500 width=12) (actual time=10.433..11.253 rows=5886 loops=1)
  Group Key: v.id
  Batches: 1  Memory Usage: 913kB
  ->  Hash Join  (cost=794.25..1037.89 rows=7500 width=8) (actual time=4.058..8.693 rows=7500 loops=1)
        Hash Cond: (tt.volontario = v.id)
        ->  Hash Join  (cost=246.75..470.70 rows=7500 width=8) (actual time=1.559..4.361 rows=7500 loops=1
              Hash Cond: (t.id = tt.id)
              ->  Seq Scan on turni t  (cost=0.00..204.25 rows=7500 width=4) (actual time=0.015..1.170 rows
                    Filter: (data <= (CURRENT_DATE + '2 mons'::interval))
              ->  Hash  (cost=153.00..153.00 rows=7500 width=12) (actual time=1.511..1.512 rows=7500 loops=
                    Buckets: 8192  Batches: 1  Memory Usage: 387kB
                    ->  Seq Scan on turni_trasporti tt  (cost=0.00..153.00 rows=7500 width=12) (actual time
        ->  Hash  (cost=360.00..360.00 rows=15000 width=4) (actual time=2.460..2.461 rows=15000 loops=1)
              Buckets: 16384  Batches: 1  Memory Usage: 656kB
              ->  Seq Scan on volontari v  (cost=0.00..360.00 rows=15000 width=4) (actual time=0.014..1.10
Planning Time: 0.293 ms
Execution Time: 11.716 ms
```

Da come possiamo vedere, i punti critici sono soprattutto sul confronto (per range) nelle varie date presenti nelle varie tabelle, in generale i join vengono eseguiti in maniera abbastanza efficiente (sono praticamente tutti hash join), perche' vengono fatti su attributi chiave, quindi gia' presenti in un indice. Abbiamo inoltre notato che, nelle tabelle clienti e familiari, sono gia' presenti degli indici per i codici fiscali (essendo in entrambe le tabelle chiave), pero' ad albero. Siccome pensiamo che in nessun caso verra' effettuata una ricerca per range su un codice fiscale, e' buono sostituire l'indice ad albero con un indice hash.

NOTE:

- l'indice $I_{CF}(clienti)$ potrebbe essere tranquillamente ad hash

```
-- Per visualizzare gli indici
SELECT C.oid, relname, indexrelid, relam as tipo_indice, indnatts, indisunique, indisprimary, indiscluster
FROM (pg_namespace N JOIN pg_class C ON N.oid = C.relnamespace) JOIN pg_index ON C.oid = indexrelid
WHERE N.nspname = 'social_market'
```

- Inserire indice $I_{data_scarico}(prodotti)$ ad albero

```
Seq Scan on prodotti  (cost=0.00..169.94 rows=6568 width=33) (actual time=0.014..4.023 rows=6568 loops=1)
  Filter: (scadenza <= CURRENT_DATE)
Planning Time: 0.117 ms
Execution Time: 4.160 ms

-- f_selettivita: 1/6568
```

```
// numero di pagine occupate
+------------+--------+
|tabella     |n_pagine|
+------------+--------+
|clienti     |215     |
|familiari   |942     |
|telefoni    |192     |
|email       |288     |
|fasce_orarie|0       |
|appuntamenti|146     |
|volontari   |210     |
|scarichi    |96      |
|scorte      |0       |
|prodotti    |59      |
+------------+--------+
```