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.

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 scaricare) ecc ecc.

La prima query recupera tutti i prodotti "vicini" alla scadenza reale (a 2 settimane dalla data) la cui quantita' e' sotto le 50 unita' (e quindi bisognera' rifornire l'inventario a breve)

NOTA: siccome il database non ospita prodotti la cui scadenza reale e' nelle prossime due settimane, solamente a scopo didattico, controlliamo i prodotti la cui scadenza reale e' nel prossimo anno e la quantita' e' minore di 250.

```
SELECT p.id, s.tipologia, s.marca, p.scadenza_reale, p.data_scarico, s.qta
FROM prodotti p
NATURAL JOIN scorte s
WHERE p.scadenza_reale <= CURRENT_DATE + '1 year'::interval AND s.qta <= 250;
```

La seconda query riguarda i volontari, con i vari turni. Elenca, per ogni volontario, il numero di colli che questo trasporteta' 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;
```

Infine, abbiamo lavorato sulla parte riguardante le donazioni. Vogliamo ottenere una lista dei top donatori privati, ovvero tutti i donatori privati ordinati in base alla somma totale degli importi che hanno donato nell'ultimo anno

```
SELECT d.id AS id_donatore, SUM(importo) as tot_donazione
FROM donatori d
JOIN donazioni d2 on d.id = d2.donatore
WHERE importo IS NOT NULL AND d.id IN (SELECT id FROM donatori_privati) AND d2.data <= CURRENT_DATE - '1 year'::
GROUP BY d.id
ORDER BY tot_donazione DESC;
```

Definizione schema fisico

Per poter osservare dei cambiamenti significativi nell'efficienza delle query, e' necessario che le tabelle implicate siano occupate da una dimensione significativa di record, nel nostro caso, solamente la tabella scorte occupa 0 pagine perche' contiene solamente una quindicina di record

```
+---------------------+-------+--------------+--------+
|nome_relazione       |n_tuple|spazio_occupato|n_pagine|
+---------------------+-------+--------------+--------+
|volontari            |15000  |1680          |210     |
|scorte               |-1     |0             |0       |
|prodotti             |7024   |472           |59      |
|turni                |7500   |584           |73      |
|turni_trasporti      |7500   |624           |78      |
|donatori             |15000  |1504          |188     |
|donatori_privati     |4960   |376           |47      |
|donazioni            |15000  |1000          |125     |
+---------------------+-------+--------------+--------+
```

Schema fisico prima delle modifiche

Query 1:

```
Hash Join  (cost=11.16..212.63 rows=2308 width=1048) (actual time=0.023..1.597 rows=1149 loops=1)
  Hash Cond: (p.codice_prodotto = s.codice_prodotto)
  ->  Seq Scan on prodotti p  (cost=0.00..181.92 rows=7024 width=16) (actual time=0.009..0.985 rows=7024 loops=1
        Filter: (scadenza_reale <= (CURRENT_DATE + '1 year'::interval))
  ->  Hash  (cost=10.88..10.88 rows=23 width=1040) (actual time=0.009..0.010 rows=7 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 9kB
        ->  Seq Scan on scorte s  (cost=0.00..10.88 rows=23 width=1040) (actual time=0.004..0.006 rows=7 loops=1
              Filter: (qta <= 250)
              Rows Removed by Filter: 13
Planning Time: 0.128 ms
Execution Time: 1.650 ms
```

Query 2:

```
HashAggregate  (cost=1075.39..1150.39 rows=7500 width=12) (actual time=17.322..18.041 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=9.912..15.500 rows=7500 loops=1)
        Hash Cond: (tt.volontario = v.id)
        ->  Hash Join  (cost=246.75..470.70 rows=7500 width=8) (actual time=3.189..7.154 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=0.059..2.467 rows=7500
                    Filter: (data <= (CURRENT_DATE + '2 mons'::interval))
              ->  Hash  (cost=153.00..153.00 rows=7500 width=12) (actual time=3.109..3.109 rows=7500 loops=1)
                    Buckets: 8192  Batches: 1  Memory Usage: 387kB
                    ->  Seq Scan on turni_trasporti tt  (cost=0.00..153.00 rows=7500 width=12) (actual time=0.02
        ->  Hash  (cost=360.00..360.00 rows=15000 width=4) (actual time=6.679..6.679 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=0.038..5.114 rows
Planning Time: 10.439 ms
Execution Time: 18.533 ms
```

Query 3:

```
Sort  (cost=1210.44..1216.52 rows=2430 width=12) (actual time=7.706..7.763 rows=1916 loops=1)
  Sort Key: (sum(d2.importo)) DESC
  Sort Method: quicksort  Memory: 138kB
  ->  HashAggregate  (cost=1049.49..1073.79 rows=2430 width=12) (actual time=7.003..7.214 rows=1916 loops=1)
        Group Key: d.id
        Batches: 1  Memory Usage: 369kB
        ->  Hash Join  (cost=597.99..1037.34 rows=2430 width=12) (actual time=3.731..6.538 rows=2398 loops=1)
              Hash Cond: (d2.donatore = d.id)
              ->  Seq Scan on donazioni d2  (cost=0.00..387.50 rows=7349 width=12) (actual time=0.009..1.876 row
                    Filter: ((importo IS NOT NULL) AND (data <= (CURRENT_DATE - '1 year'::interval)))
                    Rows Removed by Filter: 7654
              ->  Hash  (cost=535.99..535.99 rows=4960 width=8) (actual time=3.702..3.704 rows=4960 loops=1)
                    Buckets: 8192  Batches: 1  Memory Usage: 258kB
                    ->  Hash Join  (cost=158.60..535.99 rows=4960 width=8) (actual time=0.817..3.212 rows=4960 l
                          Hash Cond: (d.id = donatori_privati.id)
                          ->  Seq Scan on donatori d  (cost=0.00..338.00 rows=15000 width=4) (actual time=0.004.
                          ->  Hash  (cost=96.60..96.60 rows=4960 width=4) (actual time=0.795..0.796 rows=4960 lo
                                Buckets: 8192  Batches: 1  Memory Usage: 239kB
                                ->  Seq Scan on donatori_privati  (cost=0.00..96.60 rows=4960 width=4) (actual t
Planning Time: 0.377 ms
Execution Time: 7.961 ms
```

Progettazione schema fisico

Sulla base del workload misurato, notiamo che i punti un po' piu' critici ruguardano:

- **prodotti e inventario**

Dal piano fisico scelto, notiamo che la ricerca sulla data di scadenza in un range e' effettuata con una scansione sequenziale, questo perche' non abbiano alcun indice su *scadenza_reale*. Un' idea potrebbe essere di inserire un indice ordinato su data_scadenza, scadenza_reale e data_scarico, poiche' sara' molto frequente una ricerca per range in base a questi 3 attributi. Notiamo che anche la quantita' nella tabella scorte viene cercata con una scansione sequenziale, pero' in questo caso non migliorerebbe inserire un indice, anzi peggiorerebbe le operazioni. Questo perche' la tabella scorte e' molto piccola (alla peggio qualche centianio di record), e inoltre viene modificata spesso per tenere l'inventario aggiornato (ogni volta che viene inserito un prodotto viene aumentata la quantita').

Riteniamo inoltre utile clusterizzare rispetto alle varie date, poiche' questo aiuterebbe maggiormente le ricerche per range (che riteniamo essere molte).

- **volontari e turni**

Anche in questo caso, per torvare i turni in un range rispetto alla data, il sistema esegue una scansione sequenziale. Prevediamo che le ricerche nei turni per range di date saranno frequenti, quindi conviene aggiungere un indice. Anche in questo caso conviene clusterizzare rispetto alla data del turno

- **donatori e donazioni**

Anche in questo caso, conviene un indice sulla data della donazione. Anche su importo ci verrebbe comodo mettere un indice, perche' potremmo voler cercare tutte le donazioni in un range. Anche in questo caso conviene clusterizzare rispetto alla data della donazione

Si potrebbero inserire altri indici in altre tabelle (e clusterizzare a loro volta, se necessario), pero' si tratta di tabelle piccole o comunque non frequentemente utilizzate come quelle presenti nel carico di lavoro.

Per esempio, tutti i codici fiscali presenti nel DB sono unique, di conseguenza sono chiave di ricerca di un indice ordinato. Non verranno mai effettuate, pero', ricerche per range sui codici fiscali, quindi per migliorarne l'efficienza si potrebbero sostituire con degli indici hash.