

Pathetic SQL

Un database è un filesystem dotato dell'algebra degli insiem i

La rubrica del mio telefono ha 500 posizioni, la mia biblioteca più di un centinaio di libri, più di duecento LP musicali e i giorni di un anno sono 365. Queste sono le informazioni minime necessarie per gestire un sistema di prestito e restituzione. Il loro prodotto cartesiano $500 \times 300 \times 365$ genera quasi 55 milioni di possibilità che, se gestite abbastanza male, sono più che sufficienti da farvi pensare che SQL sia un linguaggio patetico. Tanto patetico da suggerirvi erroneamente che carta e penna siano ancora i migliori strumenti per la gestione delle vostre informazioni.

Roberto A. Foglietta
me@roberto.foglietta.name

Ultimo aggiornamento: 10 febbraio 2005

```
[roberto@wsraf ~]$ su
Password:
```

```
[root@wsraf roberto]# su - postgres
```

```
-bash-3.00$ createdb 3gcam
CREATE DATABASE
```

```
-bash-3.00$ psql 3gcam
Benvenuto in psql 8.0.0, il terminale interattivo di PostgreSQL.
Digitare: \copyright per le condizioni di distribuzione
          \h per la guida sui comandi SQL
          \? per la guida sui comandi psql
          \g o terminare con punto e virgola per eseguire la query
          \q per uscire
```

```
3gcam=# \timing
Controllo tempo attivato
```

```
3gcam=# CREATE TEMP TABLE cam01 (id serial, testo text);
NOTICE: CREATE TABLE will create implicit sequence "cam01_id_seq" for serial column "cam01.id"
CREATE TABLE
Tempo: 59,288 ms
```

```
3gcam=# COPY cam01 (testo) FROM '/home/roberto/public_html/welleng/3GCAM-001/lastlog.1' DELIMITER '}';
COPY
Tempo: 8961,795 ms
```

```
3gcam=# CREATE INDEX cam01_id_index ON cam01 using btree(id int4_ops);
CREATE INDEX
Tempo: 3406,532 ms
```

```
3gcam=# SELECT id INTO TEMP udid FROM cam01 WHERE testo ~* '##### UPLOADED LOG #####';
SELECT
Tempo: 2779,607 ms
```

```
3gcam=# CREATE INDEX udid_id_index ON udid using btree(id int4_ops);
CREATE INDEX
Tempo: 21,086 ms
3gcam=#
```

Creazione e **connessione** al database.

Creazione di due tabelle temporanee a partire da un file dati di tipo testuale.

Creazione degli indici e delle sequenze.

```
3gcam=# \d cam01
```

```
Tabella "pg_temp_2.cam01"
```

```
Colonna | Tipo | Modificatori
```

```
-----+-----+-----  
id      | integer | not null default nextval('pg_temp_2.cam01_id_seq'::text)  
testo   | text    |
```

```
Indici:
```

```
"cam01_id_index" btree (id)
```

```
SELECT COUNT(*) FROM cam01;
```

```
count
```

```
-----  
465775
```

```
(1 riga)
```

```
Tempo: 870,100 ms
```

```
3gcam=# \d udid
```

```
Tabella "pg_temp_2.udid"
```

```
Colonna | Tipo | Modificatori
```

```
-----+-----+-----  
id      | integer |
```

```
Indici:
```

```
"udid_id_index" btree (id)
```

```
SELECT COUNT(*) FROM udid;
```

```
count
```

```
-----  
1443
```

```
(1 riga)
```

```
Tempo: 21,986 ms
```

Si hanno due **tabelle** entrambe **indicizzate** sull'indice intero. La prima **colonna** della prima tabella è una **sequenza** e funge da **primary key** nel secondo è un intero, che potrebbe essere **referenziato** all'indice della prima tabella per consistenza, che funge da **foreigner key**.

La prima tabella contiene 465mila **righe** di un log file prodotto da una fotocamera digitale controllata con Linux embedded nell'arco di circa due mesi di funzionamento.

La seconda contiene 1443 interi che corrispondono ai numeri di righe del log intorno alle quali ci sono informazioni interessanti da analizzare.

EXPLAIN

```
SELECT a.id,a.testo
FROM cam01 AS a, udid AS b
WHERE a.id <= b.id
      AND a.id >= b.id-15;
```

QUERY PLAN

```
-----
Nested Loop (cost=23.87..18563176.39 rows=74952306 width=36)
  Join Filter: (("outer".id <= "inner".id) AND ("outer".id >= ("inner".id - 15)))
  -> Seq Scan on cam01 a (cost=0.00..12456.78 rows=467478 width=36)
  -> Materialize (cost=23.87..38.30 rows=1443 width=4)
      -> Seq Scan on udid b (cost=0.00..22.43 rows=1443 width=4)
(5 righe)
```

Tempo: 0,861 ms

```
SELECT a.id,a.testo
FROM cam01 AS a, udid AS b
WHERE a.id <= b.id
      AND a.id >= b.id-15;
```

Tempo: 217341,501 ms

Con questa select si intende visualizzare le 15 righe precedenti all'evento di upload del porzione di log file. Per fare questo si sceglie l'espressione logico-matematica più semplice possibile, cioè che il numero di riga sia compreso fra $\Psi - 15$ e Ψ stesso, dove Ψ è il numero di riga dell'evento.

Questa select richiede ben 217 secondi (3 minuti e $\frac{1}{2}$) su un P4 a 2.8GHz con 512Mb di RAM DDR nonostante l'esistenza di indici su entrambe le colonne "id" delle due tabelle. Il prodotto cartesiano delle due tabelle è di quasi 675 milioni di righe ma il suo sottoinsieme che viene indagato dalla select è di *soltanto* quasi 75 milioni di righe.

```

EXPLAIN
  SELECT cam01.*
  FROM cam01
  WHERE cam01.id
    BETWEEN (SELECT min(udid.id)
             FROM udid
             WHERE cam01.id >= udid.id
             AND cam01.id-15 <= udid.id)
    AND cam01.id;

```

QUERY PLAN

```

-----
Seq Scan on cam01 (cost=0.00..10991936.48 rows=51942 width=36)
  Filter: ((id <= id) AND (id >= (subplan)))
  SubPlan
    -> Aggregate (cost=23.48..23.48 rows=1 width=4)
        -> Index Scan using udid_id_index on udid (cost=0.00..23.46 rows=8 width=4)
            Index Cond: (($0 >= id) AND (($0 - 15) <= id))

```

(6 righe)

Tempo: 1,188 ms

Lo stesso risultato si può ottenere con una select diversa la quale lavora su uno spazio di sole 52mila righe senza dover implementare il prodotto cartesiano fra le due tabelle e sfruttandone completamente gli indici.

```

SELECT cam01.*
  FROM cam01
  WHERE cam01.id
    BETWEEN (SELECT min(udid.id)
             FROM udid
             WHERE cam01.id >= udid.id
             AND cam01.id-15 <= udid.id)
    AND cam01.id;

```

Tempo: 2424,690 ms

Questa seconda select viene completata in 2 secondi e 1/2, risultando più veloce di 80 volte rispetto alla prima più semplice select.

```

EXPLAIN
  SELECT cam01.*
     FROM cam01
    WHERE cam01.id
          BETWEEN (SELECT min(udid.id)
                   FROM udid
                   WHERE cam01.id >= udid.id
                   AND cam01.id <= udid.id+15)
          AND cam01.id;

```

QUERY PLAN

```

-----
Seq Scan on cam01 (cost=0.00..15748934.95 rows=51942 width=36)
  Filter: ((id <= id) AND (id >= (subplan)))
  SubPlan
    -> Aggregate (cost=33.66..33.66 rows=1 width=4)
         -> Seq Scan on udid (cost=0.00..33.25 rows=161 width=4)
              Filter: (($0 >= id) AND ($0 <= (id + 15)))

```

(6 righe)

Tempo: 1,293 ms

Attenzione però perché è sufficiente scrivere la stessa espressione matematica in un modo leggermente diverso per perdere tutti i benefici della precedente select.

```

SELECT cam01.*
   FROM cam01
  WHERE cam01.id
        BETWEEN (SELECT min(udid.id)
                 FROM udid
                 WHERE cam01.id >= udid.id
                 AND cam01.id <= udid.id+15)
        AND cam01.id;

```

Tempo: 393025,402 ms

Lo spazio di indagine è immutato ma si è persa la possibilità di usare gli indici. Il tempo necessario sale ad oltre 6½ minuti che è maggiore persino della prima select.

Il problema da risolvere è la giunzione dei log generati da un sistema embedded. Essi vengono trasmessi a pezzi e si sovrappongono per una parte (circa 10 righe).

```
SELECT cam01.*
  INTO TEMP dups
  FROM cam01
  WHERE cam01.id
        BETWEEN (SELECT min(udid.id)
                  FROM udid
                  WHERE cam01.id > udid.id
                  AND cam01.id-15 <= udid.id)
        AND cam01.id;
```

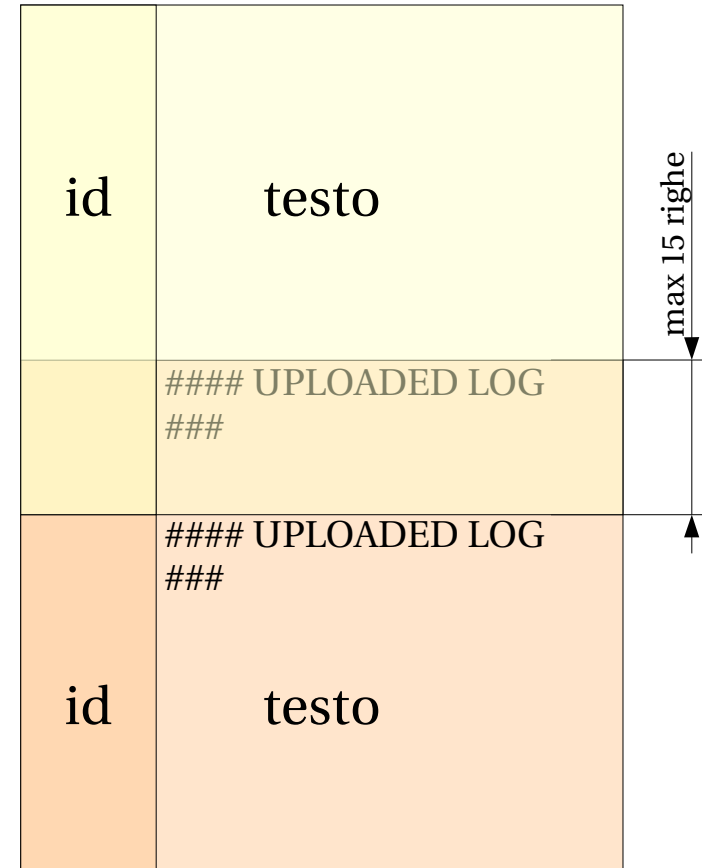
Tempo: 4120,096 ms

```
3gcam=# \d dups
          Tabella "pg_temp_2.dups"
  Colonna | Tipo      | Modificatori
-----+-----+-----
   id     | integer   |
  testo   | text      |
```

```
3gcam=# SELECT COUNT(*) FROM dups;
 REATE
 count
-----
 17340
(1 riga)
```

Tempo: 130,307 ms

giuntura {



L'ora e la data potrebbero non essere affidabili e quindi non si può fare giunture e ordinamento basandosi su questi dati. Occorre perciò un ID progressivo (numero di riga) che va mantenuto per poter disporre della progressione temporale esatta.

```
SELECT id,testo INTO log01 FROM cam01 EXCEPT SELECT id,testo FROM dups;  
SELECT  
Tempo: 18211,295 ms
```

```
ALTER TABLE log01 ADD ncam int4;  
ALTER TABLE  
Tempo: 3,610 ms
```

```
UPDATE log01 SET ncam=1;  
UPDATE 450138  
Tempo: 18913,952 ms
```

L'elaborazione mediante tabelle temporanee è generalmente più veloce. In questo particolare esempio più del doppio: 16 contro 39 secondi.

La velocità si perde nel caso che l'elaborazione non sia troppo complessa e/o che il prodotto debba comunque essere salvato in una tabella persistente (24 secondi).

Un vantaggio che comunque rimane è, ad esempio, che “log_01” non risulta più essere ordinata per “id” dopo l'update della colonna “ncam” mentre con l'inserimento in una tabella persistente si può imporre l'ordinamento desiderato con ORDER BY.

```
SELECT id,testo INTO TEMP log_01 FROM cam01 EXCEPT SELECT id,testo FROM dups;  
SELECT  
Tempo: 10785,343 ms
```

```
ALTER TABLE log_01 ADD ncam int4;  
ALTER TABLE  
Tempo: 5,207 ms
```

```
UPDATE log_01 SET ncam=1;  
UPDATE 450138  
Tempo: 5012,483 ms
```

Questo significa che una SELECT su un sistema in produzione dovrebbe sempre riportare per esplicito le colonne da visualizzare e anche l'ordinamento.

L'uso delle VIEW in questo contesto può diventare realmente interessante per staccare più nettamente i due livelli: application-level e database administration.

```
3gcam=# INSERT INTO rawlog(id,cam,testo) SELECT id,ncam,testo FROM log_01 ORDER BY id;  
INSERT 0 450138  
Tempo: 24238,271 ms
```

```
3gcam=# SELECT COUNT(*)
        FROM rawlog
        WHERE testo ~* 'error';

count
-----
      842
(1 riga)
```

Tempo: 1707,168 ms

```
3gcam=# SELECT COUNT(*)
        FROM rawlog
        WHERE testo ~* 'error' AND id < 20000;

count
-----
      2
(1 riga)
```

Tempo: 1823,794 ms

```
3gcam=# SELECT COUNT(*)
        FROM rawlog
        WHERE id < 20000 AND testo ~* 'error';

count
-----
      2
(1 riga)
```

Tempo: 411,234 ms

Ritorno ancora sulla questione delle prestazioni ma questa volta usando un esempio più semplice: desidero contare quante righe d'errore ci sono in tutto il log e solo nelle prime 20 mila righe.

Si noti che le prime due query necessitano pressoché il medesimo tempo per essere eseguite mentre la terza che è matematicamente equivalente alla seconda richiede meno del 25% del tempo.

La regola generale è quella di mettere le condizioni più semplici da verificare per prime e a parità di semplicità quelle più restrittive in maniera che le successive siano applicate sul sottoinsieme più piccolo possibile. Ovviamente si tratta di una regola euristica e il calcolo esatto non è per nulla facile da fare.

Condizioni più semplici in ordine: binarie, numeriche, testuali e polinomiali (sqrt... exp)

```

SELECT date(date),count(*),' '::text
      INTO graph
      FROM logcam001
      WHERE date(date) IN (SELECT date FROM days)
            AND log ~* 'error'
            GROUP BY date(date)
            ORDER BY date;

```

```

UPDATE graph
      SET text = rpad('', floor(10*log(count))::int4, '#');

```

```

SELECT max(count) FROM graph;
max = 287

```

```

SELECT sum(count) FROM graph;
sum = 865

```

```

test=# SELECT * FROM graph;

```

date	count	text
2004-11-04	2	###
2004-11-05	1	
2004-11-06	1	
2004-11-08	15	#####
2004-11-09	1	
2004-11-15	2	###
2004-11-16	13	#####
2004-11-23	3	###
2004-11-25	20	#####
2004-11-26	26	#####
2004-11-27	287	#####
2004-11-28	51	#####
2004-11-29	57	#####
2004-11-30	49	#####
2004-12-01	1	
2004-12-12	1	
2004-12-19	5	#####
2004-12-22	1	
2004-12-24	2	###
2004-12-26	5	#####
2004-12-27	2	###
2004-12-28	17	#####
2004-12-30	11	#####
2004-12-31	2	###
2005-01-01	21	#####
2005-01-02	6	#####
2005-01-03	1	
2005-01-04	2	###
2005-01-06	13	#####
2005-01-07	1	
2005-01-08	1	
2005-01-09	1	
2005-01-11	67	#####
2005-01-12	99	#####
2005-01-14	2	###
2005-01-15	1	



Capitasse mai di dover fare con urgenza un grafico logaritmico avendo sottomano solamente una console testuale e senza fogli di calcolo...

...altrimenti l'esercizio è buono per imparare a raggruppare il risultato di una select per un serie di valori, gli ordinamenti e alcune funzioni matematiche e di manipolazione di stringhe.

2004-11-09	1	
2004-11-15	2	###
2004-11-16	13	#####
2004-11-23	3	###
2004-11-25	20	#####
2004-11-26	26	#####
2004-11-27	287	#####
2004-11-28	51	#####
2004-11-29	57	#####
2004-11-30	49	#####
2004-12-01	1	

```
SELECT DISTINCT date(date)
  FROM logcam001
 WHERE log ~* 'error';
```

Tempo: 1921,324 ms

```
SELECT date(date)
  INTO TEMP days
  FROM logcam001
 WHERE log ~* 'error'
 GROUP BY date(date);
```

Tempo: 1530,890 ms

```
SELECT date(date),count(*)
  FROM logcam001
 WHERE log ~* 'error'
 GROUP BY date(date);
```

Tempo: 1577,528 ms

Alcuni esempi sulla convenienza di usare DISTINCT invece del GROUP BY che oltretutto permette maggiore flessibilità oltre che velocità.

Le prestazioni però in termini di tempo sono sempre da riferirsi a query specifiche su basi dati specifiche e sono soggette ad una diverse variabili per le quali la comparazione non è sempre possibile eccetto casi eclatanti come quelli prima mostrati soprattutto dovuti a cattivo uso di indici.

RINGRAZIAMENTI:

Christopher Mair <christopher.mair@sad.it> per aver sempre raccolto le mie sfide e le mie provocazioni su SQL in maniera propositiva.

Stefano Martinelli <stefano.martinelli@sad.it> più comodo del *man* e più veloce di *google* nel suggerirmi quale comando adoperare.

Tutti i lettori, in particolare i più attenti che mi hanno segnalato errori, omissioni e quant'altro di fallace imputabile alla mia umana natura!

© 2005, Roberto A. Foglietta <me@roberto.foglietta.name>
rilasciato sotto Creative Common License (vedere pagina successiva)



Attribuzione-StessaLicenza 2.0 Italia

Tu sei libero:

- di distribuire, comunicare al pubblico, rappresentare o esporre in pubblico l'*opera*
- di creare *opere derivate*
- di utilizzare l'*opera* per scopi commerciali.

Alle seguenti condizioni:



Attribuzione. Devi riconoscere la paternità dell'*opera* all'*autore originario*.



Condividi sotto la stessa licenza. Se alteri, trasformi o sviluppi quest'*opera*, puoi distribuire l'*opera* risultante solo per mezzo di una licenza identica a questa.

- In occasione di ogni atto di riutilizzo o distribuzione, devi chiarire agli altri i termini della licenza di quest'*opera*.
- Se ottieni il permesso dal titolare del diritto d'autore, è possibile rinunciare a ciascuna di queste condizioni.

Le tue utilizzazioni libere e gli altri diritti non sono in nessun modo limitati da quanto sopra.