Progetto Basi di Dati - Social Market
Parte III

Alessio De Vincenzi 4878315
Edoardo Risso 5018707
Federica Tamerisco 4942412
Mattia Cacciatore 4850100

9 Gennaio 2023

Contents

1 9.a - Descrizione e codice SQL interrogazioni scelte sul carico di lavoro

1.1 Interrogazione 1 (con JOIN)

```
SELECT f.CodiceFamiglia, a.DataOraInizio
FROM Cliente c JOIN Famiglia f ON c.CodiceFamiglia = f.CodiceFamiglia
               JOIN Appuntamento a ON c.CodiceCliente = a.CodiceCliente
WHERE a.DataOraInizio > CURRENT_TIMESTAMP
GROUP BY f.CodiceFamiglia, a.DataOraInizio
ORDER BY a.DataOraInizio, f.CodiceFamiglia;
```

Interrogazione che restituisce la lista degli appuntamenti futuri registrati con le famiglie.
Utilizzata molto dai volontari per sapere e organizzare i successivi appuntamenti.

1.2 Interrogazione 2 (condizione complessa)

```
SELECT *
FROM Cliente c
WHERE ((CURRENT_DATE - c.DataInizioAutorizzazione) <= 180)
--WHERE ((date '2021-07-01' - c.DataInizioAutorizzazione) <= 180)
      AND c.CodiceCliente NOT IN (SELECT a.CodiceCliente
                  FROM Appuntamento a
                  WHERE a.CodiceCliente = c.CodiceCliente AND a.DataOraInizio <= CURRENT_TIMESTAMP);
```

Interrogazione che restituisce, per ogni famiglia, la lista dei membri autorizzati che non hanno mai effettuato una spesa.
Utilizzata per individuare i membri autorizzati non attivi per eventuali ricerche e/o revisione delle autorizzazioni.

1.3 Interrogazione 3

```
SELECT NomeProdotto, DataScadenza
FROM Prodotto
WHERE DataScadenza BETWEEN CURRENT_DATE AND (CURRENT_DATE + interval '1' month)
--WHERE DataScadenza BETWEEN CURRENT_DATE AND (CURRENT_DATE + interval '1000' month)
ORDER BY DataScadenza;
```

Interrogazione che restituisce la lista dei prodotti che scadono questo mese con data di scadenza in ordine crescente.
Utilizzata molto dai volontari per poter consigliare ai clienti i prodotti vicino alla scadenza.

N.B. Le interrogazioni presentano codice commentato, che si rende utile decommentare e usare per farsi restituire risultati grossi e significativi in fase di testing.

2 9.b - Progetto fisico con elenco indici

2.1 Interrogazione 1 (con JOIN)

L'idea era quella di aggiungere un indice di tipo B-Tree alla relazione Appuntamento dove avviene la selezione delle tuple che soddisfano la condizione WHERE:

CREATE INDEX IndiceOrariAppuntamento ON Appuntamento(DataOraInizio);
CLUSTER Appuntamento USING IndiceOrariAppuntamento;

Grazie al quale velocizzare la ricerca di DataOra/Timestamp successivi al momento in cui viene eseguita l'interrogazione, contando su un accesso in tempo polinomiale logaritmico, dipendente dalla profondità dell'albero bilanciato, alle tuple d'interesse.
TUTTAVIA nella documentazione[1] ufficiale di PostgresSQL viene specificato che l'aggiunta del vincolo UNIQUE o PRIMARY KEY su un attributo genera automaticamente un indice di tipo B-Tree. Gli indici risultano comunque implementabili anche su altri DBMS che potrebbero non supportare questa funzionalità di PostgresSQL.

2.2 Interrogazione 2 (condizione complessa)

Per questa interrogazione la parte computazionale più pesante riguarda l'operatore NOT IN che fa eseguire le scansioni sequenziali sulle tabelle Cliente e Appuntamento, facendo impiegare all'interrogazione diversi secondi per essere terminata, è richiesto quindi un indice di tipo B-Tree che migliori l'efficienza e riduca il numero di accessi disco:

CREATE INDEX IndiceCliente ON Appuntamento(CodiceCliente);
CLUSTER Appuntamento USING IndiceCliente;

2.3 Interrogazione 3

In questa interrogazione, dato che il sistema deve operare su attributi che fanno parte della chiave alternativa, vi è comunque un indice B-Tree pre-esistente fornito da PostgresSQL[1].

L'aggiunta di un indice di tipo B-Tree sul singolo attributo, per la ricerca dell'intervallo dei valori, clusterizzato, produce miglioramenti marginali:

CREATE INDEX IndiceScadenzaProdotto ON Prodotto(DataScadenza);
CLUSTER Prodotto USING IndiceScadenzaProdotto;

3 9.c - Tabella

Dati ottenuti con:

SELECT RelName AS Tabella, RelPages AS NumeroPagine
FROM pg_class
WHERE RelName IN ('appuntamento','cliente','famiglia','prodotto');

e:

SELECT COUNT(*) FROM Appuntamento/Cliente/Famiglia/Prodotto;

Tabella	Numero pagine	Numero tuple coinvolte
Cliente	394	25000
Famiglia	148	20000
Prodotto	95	10000
Appuntamento	155	15000

4 9.d - Descrizione piani di esecuzione scelti dal sistema

4.1 Interrogazione 1 (con JOIN)

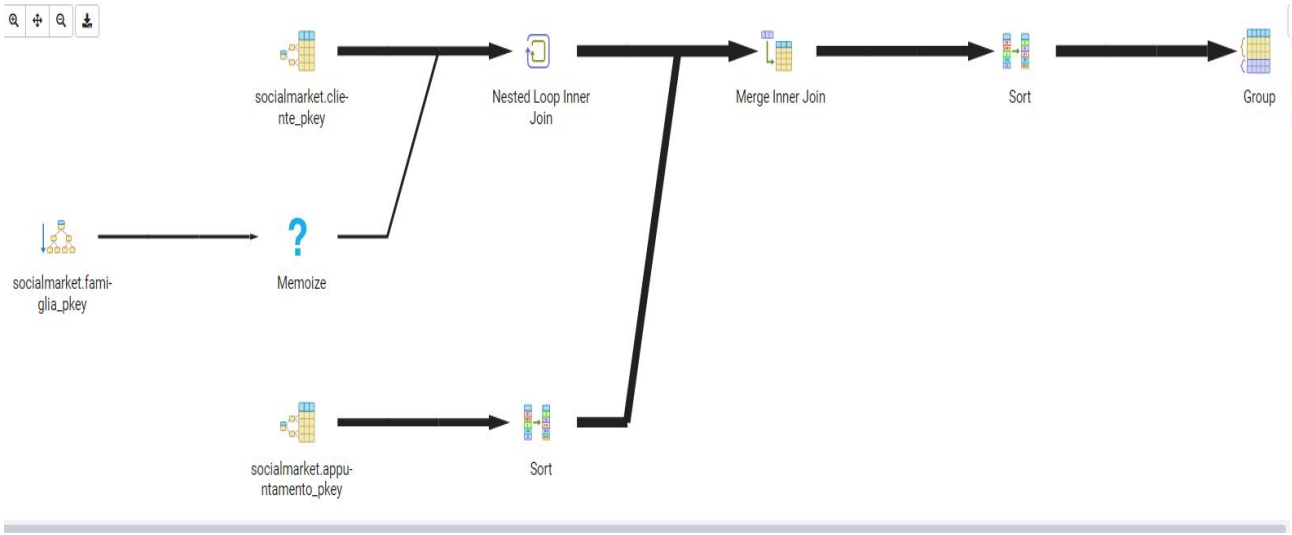
Provando ad eseguire l'interrogazione prima e dopo l'aggiunta dell'indice (aka creazione dello schema fisico), le tempistiche di esecuzione non cambiano:

Graphical | Analysis | Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Group (cost=907.64..945.58 rows=5058 width=16) (actual=2.69..3.355 rows=5055 loops=1)	0.536 ms	3.355 ms	↑ 1.01	5055	5058	1
2.	→ Sort (cost=907.64..920.29 rows=5058 width=16) (actual=2.69..2.82 rows=5055 loops=1)	0.957 ms	2.82 ms	↑ 1.01	5055	5058	1
3.	→ Merge Inner Join (cost=516.56..596.47 rows=5058 width=16) (actual=1.29..1.863 row...	0.398 ms	1.863 ms	↑ 1.01	5055	5058	1
4.	→ Nested Loop Inner Join (cost=0.58..3969.42 rows=25000 width=16) (actual=0.01...	0.038 ms	0.05 ms	↑ 961.54	26	25000	1
5.	→ Index Scan using cliente_pkey on socialmarket.cliente as c (cost=0.29..1056...	0.012 ms	0.012 ms	↑ 961.54	26	25000	1
6.	→ Memoize (cost=0.3..0.35 rows=1 width=8) (actual=0.001..0.001 rows=1 loo... Buckets: Batches: Memory Usage: 1 kB	0.001 ms	0.001 ms	↑ 1	1	1	26
7.	→ Index Only Scan using famiglia_pkey on socialmarket.famiglia as f (cos... Index Cond: (f.codicefamiglia = c.codicefamiglia)	0.001 ms	0.001 ms	↑ 0.04	1	1	8
8.	→ Sort (cost=515.98..528.62 rows=5058 width=16) (actual=1.274..1.415 rows=505...	0.641 ms	1.415 ms	↑ 1.01	5055	5058	1
9.	→ Index Scan using appuntamento_pkey on socialmarket.appuntamento as a (... Index Cond: (a.dataorainizio > CURRENT_TIMESTAMP)	0.774 ms	0.774 ms	↑ 1.01	5055	5058	1

L'aggiunta di altri indici nelle altre tabelle per velocizzare le operazioni di JOIN risulta superflua dato che lavorano sulle chiavi primarie e devono lavorare sulle tabelle dall'inizio alla fine (nota: il sistema esegue un ordinamento con quicksort prima e dopo l'esecuzione delle operazioni di JOIN), alcuni indici sono stati aggiunti allo script SQL, e sono facilmente testabili, ma non alterano il piano fisico scelto dal sistema (se non nella scelta di un indice di tipo B-Tree rispetto ad un altro, ma non ci sono differenze prestazionali e guadagni in termini di efficienza apprezzabili).
Il sistema, dunque, sceglie di usare gli indici di tipo B-Tree presenti di base per le chiavi primarie per accedere più efficientemente alle tuple d'interesse.

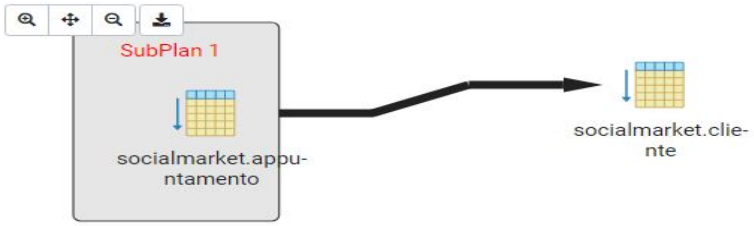
Esso esegue, innanzitutto, un JOIN tra le tabelle Cliente e Famiglia con un (INDEX) NESTED LOOP JOIN potendo contare sul rapido accesso[5] alla singola tupla in Famiglia di cui fa parte un cliente tramite l'indice B-Tree presente sulla chiave primaria e specificato nella condizione di JOIN, ossia il codicefamiglia, e poi memorizza[4] il risultato in una cache evitando così ulteriori e ridondanti accessi a disco o alla relazione se la famiglia cercata fosse una di quelle precedentemente trovate. Tutto questo piano permette di ridurre enormemente il numero di tuple accedute. Nel caso in questione si passa dalle teoriche 25000 a 26.
Il risultato viene poi unito alla tabella Appuntamento, precedentemente ordinata con quicksort, tramite MERGE JOIN perchè le tabelle in input sono ordinate[6] rispetto all'attributo di join, ergo sfruttando questa proprietà e usando un algoritmo di tipo MergeSort, si evitano confronti inutili migliorando l'efficienza.
Il risultato viene riordinato, di nuovo, tramite quicksort, e poi lette e restituite tutte le tuple che soddisfano la condizione, in questo modo l'interrogazione risulta piuttosto rapida e il piano scelto dal sistema, a confronto con il livello di complessità, mostra già un ottimo rendimento.



4.2 Interrogazione 2 (condizione complessa)

Es. mettendo "DATE '2021-07-01'" per la data di autorizzazione nella clausola WHERE per avere una grossa tabella su cui operare:

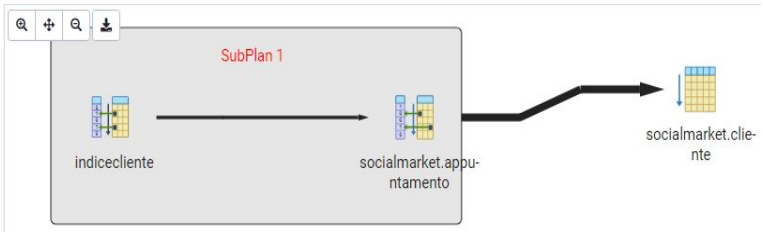
#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Seq Scan on socialmarket.cliente as c (cost=0..5232050.25 rows=4167 width=93) (actu... Filter: ((('2021-07-01'::date - c.datainizioautorizzazione) <= 180) AND (NOT (SubPlan 1))) Rows Removed by Filter: 5185	12888.389 ms	12889.036 ms	↓ 4.76	19815	4167	1
2.	→ Seq Scan on socialmarket.appuntamento as a (cost=0..417.5 rows=399 width=8) (... Filter: ((a.codicecliente = c.codicecliente) AND (a.dataorainizio <= CURRENT_TIMESTAMP)) Rows Removed by Filter: 14994	0.647 ms	0.647 ms	↓ 1	0	399	19832



Si nota subito che, in assenza di indici con cui operare, il sistema non ha altra scelta che scandire sequenzialmente la tabella Appuntamento ogni volta che si trova a verificare un nuovo cliente, il tutto moltiplicato per la quantità di clienti presenti nella tabella Cliente.
Aggiungendo, invece, l'indice clusterizzato di tipo B-Tree nella relazione Appuntamento sull'attributo CodiceCliente, l'operatore NOT IN, ergo il sistema, ne beneficia enormemente potendo contare su una tabella ordinata sui codici clienti ed evitando così di doverla scandire tutta per determinare se un determinato cliente autorizzato ha avuto o meno un appuntamento:

```
CREATE INDEX IndiceCliente ON Appuntamento(CodiceCliente);
CLUSTER Appuntamento USING IndiceCliente;
```

#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Seq Scan on socialmarket.cliente as c (cost=0..2304169 rows=4167 width=93) (actual=0.014..... Filter: ((('2021-07-01'::date - c.datainizioautorizzazione) <= 180) AND (NOT (SubPlan 1))) Rows Removed by Filter: 5185	43.923 ms	43.924 ms	↓ 4.76	19815	4167	1
2.	→ Bitmap Heap Scan on socialmarket.appuntamento as a (cost=8.88..174.38 rows=399 widt... Filter: (a.dataorainizio <= CURRENT_TIMESTAMP) Rows Removed by Filter: 0 Recheck Cond: (a.codicecliente = c.codicecliente) Heap Blocks: exact=17	0.001 ms	0.001 ms	↓ 1	0	399	19832
3.	→ Bitmap Index Scan using indicecliente (cost=0..8.79 rows=600 width=0) (actual=0..0 r... Index Cond: (a.codicecliente = c.codicecliente)	1 ms	1 ms	↑ 0.04	1	600	19832



Il sistema sceglie di usare l'indice perchè la tabella è più grande di una pagina e il numero di accessi disco mediante indice è minore rispetto a quello con la scansione sequenziale.
La condizione è più selettiva, ergo il sistema ha bisogno di accedere ad un numero inferiore di tuple e lo fa seguendo i puntatori delle foglie del B-Tree, eseguendo meno accessi disco.
Notare che il sistema non usa solamente l'indice dato che i fattori di carico non sono troppo piccoli, ma una struttura

ausiliaria ossia una bitmap per rendere l'interrogazione più efficiente.
In questo test l'interrogazione ha impiegato solamente 43,923ms anzichè 12888,389ms con un guadagno d'efficienza di 30000 volte, decisamente notevole.

4.3 Interrogazione 3

Es. mettendo "interval '1000' month" per la data di scadenza nella clausola WHERE per avere una grossa tabella su cui operare:

#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Index Only Scan using prodotto_datascadenza_nomeprodotto_key on socialmarket.prodotto as pr... Index Cond: ((prodotto.datascadenza >= CURRENT_DATE) AND (prodotto.datascadenza <= (CURRENT_DATE + '83 years 4 mons'::interval month)))	1.34 ms	1.34 ms	↑ 1	9862	9862	1

All'inizio il sistema decide di sfruttare l'indice tipo B-Tree automaticamente generato sulla chiave alternativa formata dai 2 attributi NomeProdotto e DataScadenza riuscendo a restituire il risultato in 1,34 ms.

#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Index Scan using indicescadenzaprodotto on socialmarket.prodotto as prodotto (cost=0.29..414.5... Index Cond: ((prodotto.datascadenza >= CURRENT_DATE) AND (prodotto.datascadenza <= (CURRENT_DATE + '83 years 4 mons'::interval month)))	1.128 ms	1.128 ms	↑ 1	9862	9862	1

Tuttavia, aggiungendo un altro indice di tipo B-Tree e clusterizzandolo, il sistema cambia il suo piano di esecuzione e seleziona il nuovo indice appena creato riuscendo ad essere leggermente più efficiente restituendo il risultato in 1,128 ms potendo contare su tuple ordinate rispetto alla data di scadenza (sono state fatte più prove, nonostante i valori oscillino un po' c'è sempre uno scarto seppur piccolo).

5 10 - Transazione

```
START TRANSACTION;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT *
FROM Cliente;

SELECT *
FROM Famiglia;

INSERT INTO Autorizzatore VALUES(15,'centro ascolto','Sun','Rome','17 Street', 9);

COMMIT;
```

Come transazione è stata scelta questa, una lettura della tabella Clienti per visionare i clienti del socialmarket, un'altra lettura sulla tabella Famiglia per visionare le famiglie dei clienti e una scrittura per aggiungerne un nuovo autorizzatore.
Nel contesto di questa specifica transazione è stato scelto il livello di isolamento intermedio, ossia il **READ COMMITTED**.
Le motivazioni dietro questa scelta risiedono nel voler assicurare l'assenza di anomalie quali **LOST UPDATE**, per cui l'inserimento del nuovo autorizzatore deve realizzarsi e **DIRTY READ**, per evitare le letture sbagliate/sporche delle tabelle per modifiche da parte di altre transazioni.
Non è stato ritenuto necessario, oltre che per non inficiare troppo sulle performance della transazione e sul sistema, scegliere un alto livello di isolamento e permettere la presenza potenziale di anomalie come **UNREPEATABLE READ**, dato che si fanno solo singole letture, e **PHANTOM ROW** dato che le letture non sono direttamente collegate all'operazione di scrittura/inserimento.

6 11 - Controllo accessi

TABELLA PRIVILEGI	alice	roberto
Appuntamento	ALL	SELECT, INSERT
Autorizzatore	ALL	SELECT
Cliente	ALL	SELECT
Disponibilità	ALL	SELECT
Donatore	ALL	SELECT, INSERT
Famiglia	ALL	SELECT
FasciaOraria	ALL	SELECT
FornireServizio	ALL	SELECT
IngressoMerce	ALL	SELECT, INSERT
Inventario	ALL	SELECT. INSERT
Prezzario	ALL	SELECT, INSERT
Prodotto	ALL	SELECT, INSERT, UPDATE
Scarica	ALL	SELECT, INSERT
Servizio	ALL	SELECT
Spesa	ALL	SELECT, INSERT
Trasporto	ALL	SELECT
Turnazione	ALL	SELECT
Turno	ALL	SELECT
Veicolo	ALL	SELECT
Volontario	ALL	SELECT

N.B. ALL = SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER.

La scelta di questa politica segue la filosofia del minimo privilegio.
Al gestore del socialmarket, ossia **alice**, vanno ovviamente concessi tutti i privilegi sullo schema del social market dato il suo ruolo.
La scelta di limitare fortemente i privilegi di **roberto**, principalmente, alle letture e qualche scrittura, è dettata da tre fattori, il primo riguarda la mancanza di specifica che non stabilisce chiaramente cosa un volontario possa fare o non possa fare all'interno del DBMS, il secondo guarda all'integrità della base dati stessa, per cui si parte già con il NON fornire privilegi insensati ad un volontario come la possibilità di eliminare donatori, ingressi merci registrati, prezzi e via dicendo e il terzo riguarda il ruolo e i servizi legati al volontario.
Sul discorso errore umano si rimanda la responsabilità al gestore per il ripristino di stati precedenti a inserimenti o aggiornamenti errati. Meglio avere, in linea di principio, un carico di lavoro maggiore (nel caso di alice gestore) piuttosto che maggiore libertà con enormi rischi sulla sicurezza e integrità della base dati e dello schema (volontari che cancellano o modificano volontariamente o meno le liste clienti, donatori, famiglie e autorizzatori, rischiando di provocare anche danni economici).
I privilegi di scrittura per il volontario **roberto** si limitano strettamente alle attività direttamente collegate a quelle dei volontari, ossia:
-accoglienza e assistenza negli acquisti (APPUNTAMENTO, SPESA, PRODOTTO)
-riordino dei prodotti (PRODOTTO, SCARICA, PREZZARIO)

-trasporto al market dei prodotti offerti dai supermercati (TRASPORTO, INGRESSO MERCE, INVENTARIO)
-collaborazione a eventi/raccolte (INGRESSO MERCE, DONATORE)
per cui gli appuntamenti vengono fissati dai volontari, essi si occupano anche di ricevere i prodotti e registrarli nel socialmarket con le tabelle IngressoMerce, ove si includono anche le operazioni di cassa, Inventario, Prodotto (che viene costantemente aggiornato) e Prezzario. I volontari si occupano anche di registrare le spese effettuate dai clienti e di scaricare i prodotti scaduti in Spesa e Scarica (per esempio tramite una procedura, ecco perchè in Prodotto viene concesso anche il privilegio di UPDATE).

7 Fonti e riferimenti

1) https://www.postgresql.org/docs/9.6/ddl-constraints.html
2) https://www.postgresql.org/docs/9.6/indexes-index-only-scans.html
3) https://www.postgresql.org/docs/9.6/using-explain.html
4) https://www.postgresql.org/docs/9.6/runtime-config-query.html
5) Slides corso - "Ottimizzazione Fisica" pag.52
6) Slides corso - "Ottimizzazione Fisica" pag.55