Basi di Dati 2020-21

Progetto “Online Challenge Activity”

Studente: Caputo Luca
Matricola: 4680479
Corso: Informatica
CFU: 12
Parte 3

Indice

11. Elaborazione delle interrogazioni

11.1. Prima della creazione dello schema fisico

Si considerano le interrogazioni contenute nel carico di lavoro e definite nella sezione 4 della parte 1, ma senza la creazione dello schema fisico. Per ogni interrogazione, si riporta la specifica in linguaggio naturale, l'output ottenuto e la descrizione dell'algoritmo e del tempo di esecuzione, tenendo conto che entrambe le tabelle Gioco e Sfida sono state popolate entrambe con 10000 tuple (quindi ben oltre lo spazio richiesto, ma utile per verificare i piani di esecuzione scelti dal sistema).
Inoltre, per ogni interrogazione, si riporterà l'output ottenuto dopo circa 5 o 6 esecuzioni di essa, questo perché il primo output non sempre riflette quello che è il reale tempo di esecuzione (pur sapendo che non è mai esattamente lo stesso), oltre al fatto che si assume che, essendo le interrogazioni molto frequenti, queste possano essere eseguite anche tante volte di seguito.

- Determinare l'identificatore dei giochi che coinvolgono al più quattro squadre e richiedono l'uso di due dadi.

Eseguendo il comando EXPLAIN ANALYZE sull'interrogazione si ottiene il seguente output.

```
Seq Scan on gioco  (cost=0.00..363.00 rows=833 width=5) (actual time=1.589..3.212 rows=834 loops=1)
  Filter: ((max_squadre = 4) AND (dadi_richiesti = 2))
  Rows Removed by Filter: 9166
Planning time: 0.176 ms
Execution time: 3.283 ms
```

Si può notare come il sistema sceglie una semplice scansione sequenziale sulla tabella, filtrata sui due attributi presenti nella clausola di WHERE. Questo porta però a buttare oltre il 90% delle tuple in quanto non soddisfano la condizione, e si ha un tempo di esecuzione di 3.283 ms, con una variabilità di circa 3.5 ms nel caso migliore, ma a volte va oltre i 4 ms.

- Determinare l'identificatore delle sfide relativo a un gioco A di vostra scelta che, in alternativa, hanno avuto luogo a gennaio 2021 e durata massima superiore a 2 ore, o hanno avuto luogo a marzo 2021 e durata massima pari a 30 minuti.

In questo caso l'output ottenuto è il seguente.

```
Bitmap Heap Scan on sfida  (cost=239.29...293.07 rows=1 width=5) (actual time=1.658..1.668 rows=2
loops=1)
  Recheck Cond: (gioco = '6'::numeric)
  Filter: (((data >= '2021-01-01'::date) AND (data <= '2021-01-31'::date) AND (durata_max > 120)) OR
((data >= '2021-03-01'::date) AND (data <= '2021-03-31'::date) AND (durata_max = 30)))
  Rows Removed by Filter: 15
  Heap Blocks: exact=1
  ->  Bitmap Index Scan on sfida_pkey  (cost=0.00..239.29 rows=17 width=0) (actual time=1.623..1.623
rows=17 loops=1)
        Index Cond: (gioco = '6'::numeric)
Planning time: 0.259 ms
Execution time: 1.725 ms
```

In questo caso il sistema utilizza un cammino d'accesso basato su indice, dove a partire dal nodo padre si ordinano i rid inizialmente determinati, e si filtra per data e durata relative al gioco specificato nella condizione, il che porta già a rimuovere tantissime tuple (vista la presenza di 10000 giochi, e tenendo conto che con l'id di questo gioco in particolare dovrebbe restituire due tuple).
Per i nodi figli invece, si usa il gioco come indice e restituisce i rid contenuti a livello foglia che soddisfano la condizione di selezione.
Si ha infine un tempo di esecuzione di 1.725 ms, senza grandi variazioni. In generale quindi, il sistema rende l'esecuzione già piuttosto efficiente anche senza la creazione dello schema fisico, questo proprio per la scelta di usare un cammino d'accesso basato su indice.

- Determinare le sfide, di durata massima superiore a 2 ore, dei giochi che richiedono almeno due dadi. Restituire sia l'identificatore della sfida sia l'identificatore del gioco.

L'output senza schema è il seguente.

```
Merge Join  (cost=367.81..491.39 rows=390 width=10) (actual time=4.176..6.594 rows=400 loops=1)
  Merge Cond: (gioco.id = sfida.gioco)
  ->  Index Scan using gioco_pkey on gioco  (cost=0.29..1145.06 rows=4998 width=5) (actual
time=0.032..1.554 rows=499 loops=1)
        Filter: (dadi_richiesti >= 2)
        Rows Removed by Filter: 504
  ->  Sort  (cost=367.52..369.48 rows=781 width=10) (actual time=4.130..4.293 rows=781 loops=1)
        Sort Key: sfida.gioco
        Sort Method: quicksort  Memory: 61kB
        ->  Seq Scan on sfida  (cost=0.00..330.00 rows=781 width=10) (actual time=0.062..3.704 rows=781
loops=1)
              Filter: (durata_max > 120)
              Rows Removed by Filter: 9219
Planning time: 0.739 ms
Execution time: 6.727 ms
```

In questo caso viene utilizzato un partizionamento basato su ordinamento, con l'algoritmo di merge join. Con condizione quella specificata nella clausola di JOIN. Dopodiché usando un indice, per ogni gioco si filtra per dadi richiesti, si riordinano le sfide usando il quicksort, e le si filtra per durata massima con una scansione sequenziale.
Il tempo di esecuzione totale è di 6.727 ms, ma talvolta può essere più basso (nell'ordine di 6.5 ms) o addirittura più alto (nell'ordine di 7.1 ms!), per cui la creazione dello schema fisico potrebbe portare dei miglioramenti nelle prestazioni.

11.2. Dopo la creazione dello schema fisico

Come specificato nella sezione 4 della parte 1, si è deciso, sulla base dell'analisi del carico di lavoro, di creare i seguenti indici ad albero:

- Un indice multiattributo su Gioco, definito sugli attributi max_squadre e dadi_richiesti, non clusterizzato (nello script SQL index_gioco1)
- Un indice a singolo attributo su Gioco, definito sull'attributo dadi_richiesti, clusterizzato (nello script SQL index_gioco2)

- Un indice multiattributo su Sfida, definito sugli attributi gioco, data e durata_max, clusterizzato (nello script SQL **index_sfida1**)
- Un indice a singolo attributo su Sfida, definito sull'attributo durata_max, non clusterizzato (nello script SQL **index_sfida2**)

Si riporta quindi il piano di esecuzione scelto dal sistema dopo la creazione dello schema sulle stesse interrogazioni definite sopra. Anche in questo caso si riporta l'output dopo circa 5 o 6 esecuzioni di essa, proprio per avere una stima precisa di quello che sarà il nuovo tempo di esecuzione.

- Determinare l'identificatore dei giochi che coinvolgono al più quattro squadre e richiedono l'uso di due dadi.

Il sistema ora utilizza il seguente piano.

```
Index Scan using index_gioco2 on gioco  (cost=0.29..139.27 rows=833 width=5) (actual time=0.040..1.092 rows=834 loops=1)
  Index Cond: (dadi_richiesti = 2)
  Filter: (max_squadre = 4)
  Rows Removed by Filter: 1665
Planning time: 0.252 ms
Execution time: 1.174 ms
```

Viene effettuata una scansione basata su indice utilizzando **index_gioco2**, sul quale la relazione è stata clusterizzata e che quindi accede alle tuple con dadi_richiesti = 2 per poi filtrare il numero di squadre. Ora il numero di tuple accedute è molto minore, e il tempo di esecuzione è di 1.174 ms, in alcuni casi leggermente più alto (max 1.5 ms). Il tempo di esecuzione si riduce quindi del 64,2%.

- Determinare l'identificatore delle sfide relativo a un gioco A di vostra scelta che, in alternativa, hanno avuto luogo a gennaio 2021 e durata massima superiore a 2 ore, o hanno avuto luogo a marzo 2021 e durata massima pari a 30 minuti.

In questo caso l'output ottenuto è il seguente.

```
Bitmap Heap Scan on sfida  (cost=8.60..12.63 rows=1 width=5) (actual time=0.045..0.046 rows=2 loops=1)
  Recheck Cond: (((gioco = '6'::numeric) AND (data >= '2021-01-01'::date) AND (data <= '2021-01-31'::date) AND (durata_max > 120)) OR ((gioco = '6'::numeric) AND (data >= '2021-03-01'::date) AND (data <= '2021-03-31'::date) AND (durata_max = 30)))
  Heap Blocks: exact=1
  ->  BitmapOr  (cost=8.60..8.60 rows=1 width=0) (actual time=0.034..0.034 rows=0 loops=1)
        ->  Bitmap Index Scan on index_sfida1  (cost=0.00..4.30 rows=1 width=0) (actual time=0.028..0.028 rows=1 loops=1)
              Index Cond: ((gioco = '6'::numeric) AND (data >= '2021-01-01'::date) AND (data <= '2021-01-31'::date) AND (durata_max > 120))
        ->  Bitmap Index Scan on index_sfida1  (cost=0.00..4.30 rows=1 width=0) (actual time=0.005..0.005 rows=1 loops=1)
              Index Cond: ((gioco = '6'::numeric) AND (data >= '2021-03-01'::date) AND (data <= '2021-03-31'::date) AND (durata_max = 30))
Planning time: 0.399 ms
```

```
Execution time: 0.129 ms
```

Anche in questo caso, come per la versione senza schema fisico, viene utilizzato un cammino d'accesso basato su indice e in particolare un algoritmo simile al precedente, con la differenza che non filtra solo su gioco ma anche su data e durata massima. Dopodiché vengono uniti i rid ottenuti dai nodi figli accesso a indice e, tramite **index_sfida1**, li si accede con una scansione basata su indice, prima quelli che soddisfano la prima condizione, poi quella alternativa.
In questo caso il tempo di esecuzione è di 0.399 ms e quindi una riduzione del 92,5% rispetto al tempo del piano senza schema, per cui l'implementazione risulta molto efficiente.

- Determinare le sfide, di durata massima superiore a 2 ore, dei giochi che richiedono almeno due dadi. Restituire sia l'identificatore della sfida sia l'identificatore del gioco.

Da cui si ottiene il seguente output.

```
Merge Join  (cost=270.91..394.49 rows=390 width=10) (actual time=1.139..3.011 rows=400 loops=1)
  Merge Cond: (gioco.id = sfida.gioco)
  ->  Index Scan using gioco_pkey on gioco  (cost=0.29..1145.06 rows=4998 width=5) (actual
time=0.021..1.172 rows=499 loops=1)
        Filter: (dadi_richiesti >= 2)
        Rows Removed by Filter: 504
  ->  Sort  (cost=270.62..272.58 rows=781 width=10) (actual time=1.108..1.251 rows=781 loops=1)
        Sort Key: sfida.gioco
        Sort Method: quicksort  Memory: 61kB
        ->  Bitmap Heap Scan on sfida  (cost=18.34..233.10 rows=781 width=10) (actual time=0.171..0.704
rows=784 loops=1)
              Recheck Cond: (durata_max > 120)
              Heap Blocks: exact=201
              ->  Bitmap Index Scan on index_sfida2  (cost=0.00..18.14 rows=781 width=0) (actual
time=0.133..0.134 rows=781 loops=1)
                    Index Cond: (durata_max > 120)
Planning time: 0.848 ms
Execution time: 3.173 ms
```

Il piano di esecuzione scelto dal sistema è, fino alla scelta dell'algoritmo di ordinamento, uguale a quello senza schema fisico, per poi, tramite un cammino di accesso basato su indice, riordinare le tuple in base alla durata massima e, infine, accedere tramite **index_sfida2** a quelle che soddisfano la condizione definita nell'indice.
Il tempo di esecuzione è di 3.173 ms (in alcuni casi un po' più alto, fino a 3.4 ms circa), quindi con una riduzione del 52,8% rispetto al tempo di esecuzione originale.

11.3. Conclusioni

In generale si è riuscito a ridurre almeno del 50% il tempo di esecuzione di tutto il carico di lavoro, il che è un buon risultato, tuttavia si poteva fare di più relativamente alla prima interrogazione, ma considerato che si è preferito ottimizzare l'interrogazione con il join (la più difficile considerando le precedenti, e il risultato è stato soddisfacente), si può comunque accettare il risultato ottenuto. Molto bene invece la seconda interrogazione.

Tuttavia, si può notare che l'indice index_gioco1 (non clusterizzato), non è mai stato preso in considerazione dal sistema per scegliere un piano di esecuzione, quindi nelle attuali condizioni può anch'essere eliminato, considerato anche che la tabella Gioco è stata clusterizzata su index_gioco2.

12. Controllo dell'accesso

12.1. Analisi dei ruoli nel dominio

La specifica prevede un controllo dell'accesso basato sui seguenti ruoli:

- utente
- giocatore
- gameadmin
- gamecreator

Si assume per semplicità che i ruoli siano definiti in ordine gerarchico, per cui un utente avrà determinati privilegi, il giocatore i suoi oltre a quelli degli utenti e così via, fino ad arrivare al gamecreator che avrà pressoché tutti i privilegi nella base di dati definita.

Si ipotizza inoltre che i ruoli siano dinamici, ossia che un utente può ricoprire per diverse operazioni ruoli diversi: si parte quindi dal presupposto che nella piattaforma siano quasi tutti semplici utenti, per poi, ad esempio, diventare giocatori durante lo svolgimento di un gioco, e infine tornare utenti, per cui non potranno mantenere il ruolo oltre il tempo che serve per effettuare una determinata operazione.

In particolare, un **utente** è il ruolo più semplice di tutti, e si ipotizza sia colui che si registra alla piattaforma e ne può navigare i contenuti, senza però vedere i dettagli dei giochi. Un utente potrà quindi visualizzare liste di giochi e sfide effettuate (compresi punteggi quindi) o da effettuare, ma non navigarne i dettagli (caselle, quiz o giochi contenuti...). Possono inoltre creare squadre o inserirsi in esse, e quindi inserire altri utenti in una squadra da loro creata.

Il **giocatore** lo si può anch'esso definire come un semplice utente (e infatti ne eredita tutti i privilegi), tuttavia ha ulteriori privilegi relativi allo svolgimento dei giochi, in quanto ricoperto per tale finalità, e quindi visualizzarne dettagli come caselle, quiz, task, ecc. Possono inoltre visualizzare i dettagli relativi ai turni in una sfida in corso, così come aggiungere o modificare risposte a quiz e task in sfide in corso moderate, ma non cancellarle.

Il **gameadmin** è, nell'ambito del progetto, colui che gestisce le sfide, indipendentemente dal fatto che le giochi o meno, così come potrebbe essere un admin automatizzato per sfide non moderate, oppure un utente designato come coach o caposquadra, per cui erediterà tutti i privilegi di un giocatore e, in aggiunta, potrà modificare le caselle podio durante una sfida in corso, possono aggiungere nuove sfide, inserire squadre in esse e relativi punteggi. Possono inoltre cancellare le risposte dei giocatori all'interno di una sfida in corso, così come definire nuovi turni e quindi assegnare punteggi e dadi in base a quiz e task.

Infine, il **gamecreator** eredita i privilegi di tutti i ruoli precedenti e si occupa dell'aspetto contenutistico dei giochi, per cui potrà creare, modificare e cancellare giochi, icone e relativi set, così come creare, modificare e cancellare quiz e task e abbinarli a giochi e caselle.

12.2. Ruoli e privilegi assegnati

La seguente tabella specifica, prendendo come riferimento lo schema relazionale risultato della progettazione logica, i ruoli con i relativi privilegi e indicando su quali relazioni tale privilegio è possibile.

	Visualizzazione	Inserimento	Modifica	Cancellazione
Utente	Gioco Set Icona Sfida Punteggio Squadra Composizione_squadra Utente	Squadra Composizione_squadra	Squadra Composizione_squadra	Squadra Composizione_squadra
Giocatore	Gli stessi di Utente Casella_di_gioco Casella_podio Quiz Casella_ha_quiz Risposta_quiz Task Gioco_ha_quiz Gioco_ha_task Risposta_utente Risposta_utente_quiz Turno Turno_risposta_quiz	Gli stessi di Utente Risposta_utente Risposta_utente_quiz	Gli stessi di Utente Risposta_utente Risposta_utente_quiz	Gli stessi di Utente
Gameadmin	Gli stessi di Giocatore e Utente	Gli stessi di Giocatore e Utente Set Icona Sfida Punteggio Turno Turno_risposta_quiz	Gli stessi di Giocatore e Utente Set Icona Sfida Punteggio Turno Turno_risposta_quiz Casella_podio	Gli stessi di Giocatore e Utente Set Icona Sfida Punteggio Turno Turno_risposta_quiz Risposta_utente Risposta_utente_quiz
Gamecreator	Gli stessi di Gameadmin, Giocatore e Utente	Gli stessi di Gameadmin, Giocatore e Utente Gioco Info_dadi Casella_di_gioco Quiz Casella_ha_quiz Risposta_quiz Task Gioco_ha_quiz Gioco_ha_task Casella_podio	Gli stessi di Gameadmin, Giocatore e Utente Gioco Info_dadi Casella_di_gioco Quiz Casella_ha_quiz Risposta_quiz Task Gioco_ha_quiz Gioco_ha_task	Gli stessi di Gameadmin, Giocatore e Utente Gioco Info_dadi Casella_di_gioco Quiz Casella_ha_quiz Risposta_quiz Task Gioco_ha_quiz Gioco_ha_task Casella_podio

Notare che relativamente alla tabella Utente è possibile solo effettuare visualizzazioni, questo perché si ipotizza che **solo gli admin del sito possono gestire gli utenti**, in particolare l'aggiunta, che se fosse fatta dall'utente stesso sarebbe una contraddizione visto che non è ancora registrato nella piattaforma.