Data Bases II
Progettazione fisica

Michele Beretta

michele.beretta@unibg.it

Tema d'esame dicembre 2017

Si ha il seguente schema

```
Misura(IdLuogo, Tempo, Temperatura)
Luogo(Id, Nome, Regione, Longitudine, Latitudine)
```

Con le seguenti caratteristiche

Tabella	Tuple	Blocchi	Indice primario	Indici secondari
Misura	10G	10M	Hash(IdLuogo, Tempo)	B+(Temperatura, IdLuogo, Tempo) B+(IdLuogo, Temperatura, Tempo)
Luogo	10K	1K	Sequenziale	B+(Regione, Nome)

Gli alberi su `Misura` hanno 10M di foglie a 3 passi di distanza dalla radice, mentre quello su `Luogo` ha 100 foglie a 1 passo dalla radice.

Considerare la seguente query

```
SELECT IdLuogo, MAX(Temperatura), MIN(Temperatura)
FROM Misura
GROUP BY IdLuogo
```

Individuare la migliore strategia di esecuzione, fornendo una stima del numero di accessi richiesto.

Un approccio iniziale può essere quello in cui si leggono tutti i 10M di blocchi di `Misura` e poi lavorare in memoria. Il costo in questo caso è

$$C_1 = 10M$$

Un approccio ancora più naïve consiste nell'effettuare la ricerca del massimo e del minimo partendo dai luoghi: per ogni luogo si leggono le misure ad esso associate

$$C_2 = \underbrace{1K}_{\text{Blocchi di Luogo}} + \underbrace{10K}_{\text{Tuple di Luogo}} \cdot \underbrace{10M}_{\text{Blocchi di Misura}}$$

Ovviamente questo costo è peggiore del primo. Dobbiamo usare gli indici a disposizione.

Nel nostro caso, possiamo usare il secondo indice B+ tree su `Misura`, ordinato per `IdLuogo`, `Temperatura`, `Tempo`, per trovare il valore minimo e massimo di `Temperatura` per ogni `IdLuogo`.

In questo caso, possiamo accedere ad un dato `IdLuogo` con costo 4 (3 per navigare l'albero, più 1 per l'accesso alla tupla su disco).

Inoltre, essendo l'albero ordinato per `IdLuogo` e poi per `Temperatura`, possiamo trovare direttamente il minimo ed il massimo per ogni luogo semplicemente facendo 2 accessi a quest'albero.

Non solo, nelle foglie dell'albero il valore massimo di un luogo `A` è adiacente al valore minimo del luogo successivo `B`, e questo vale anche tra foglie diverse dato che l'albero è B+ (e quindi le foglie sono collegate in un linked list). Possiamo quindi condividere l'accesso al valore massimo e minimo di `Temperatura` per un luogo ed il suo successivo, rispettivamente. Abbiamo quindi solo 4 accessi per luogo al posto di 8.

Di conseguenza, il costo ottimale per la query risulta essere

$$C = \underbrace{4}_{\text{Costo uso B+ tree}} \cdot \underbrace{10K}_{\text{Tuple di Luogo}} = 40K$$

Tema d'esame dicembre 2016

Si ha il seguente schema

```
Studente(Matr, Cognome, Nome, DataNascita, Indirizzo)
Esame(Matr, CodCorso, Data, Voto)
Corso(CodCorso, Titolo, Docente)
```

Con le seguenti caratteristiche

Tabella	Tuple	Blocchi	Indice primario	Indici secondari
Studente	10K	1K	Sequenziale	B+(Matr), fanout 100, foglie pari a 1/10 dei blocchi
Esame	100K	10K	Sequenziale	B+(Matr, CodCorso), fanout 100, foglie pari a 1/10 dei blocchi
Corso	1K	100	Sequenziale	B+(CodCorso), fanout 100, foglie pari a 1/10 dei blocchi

Si consideri la seguente query

```
SELECT *
FROM Studente S
  JOIN Esame E ON S.Matr = E.Matr
  JOIN Corso C ON E.CodCorso = C.CodCorso
WHERE Cognome = "Rossi"
  AND Titolo = "DB2"
```

Sapendo che ci sono 100 studenti con cognome `"Rossi"` e 1000 esami di DB2 in totale, fornire la migliore strategia di esecuzione nei seguenti tre scenari:

1. La tabella `Esame` ha un solo indice su `(Matr, CodCorso)`
2. La tabella `Esame` ha due indici per i diversi ordinamenti della chiave primaria
3. Oltre ai due indici su `Esame`, si ha un indice su `Studente` per gli attributi `(Cognome, Nome, DataNascita)`

Gli indici si intendono sempre B+ tree.

1. La tabella `Esame` ha un solo indice su `(Matr, CodCorso)`

Non avendo nessun indice di hash ed essendo le tabelle *entry-sequenced*, bisogna usare gli indici secondari. È conveniente trovare il miglio *nested loop*.

In questo caso, possiamo fare uno scan di `Studente`, filtrare in memoria per ottenere solamente gli studenti con cognome Rossi, e poi usare gli indici secondari per fare prima l'accesso ad `Esame` e poi a `Corso`.

Dobbiamo fare una supposizione di *quanti* esami per studente ci sono in media. Una supposizione ragionevole è quella di distribuzione uniforme, quindi 10 esami per studente.

Di conseguenza, il costo in questo caso è

$$C_1 = \underbrace{1K}_{\text{Scan di Studente}} + \underbrace{100}_{\text{Studenti 'Rossi'}} \cdot \left(\underbrace{3}_{\text{Accesso B+tree a Esame}} + \underbrace{10}_{\text{Esami per studente}} \cdot \underbrace{2}_{\text{Accesso B+ tree a Corso}} \right) = 3.3K$$

Ricordare che l'accesso con il B+ tree è, in questo caso, pari all'altezza dell'albero più 1 per l'accesso a disco.

2. La tabella `Esame` ha due indici per i diversi ordinamenti della chiave primaria

La strategia al punto 1 è ancora valida. Come alternativa, vediamo se possiamo sfruttare il secondo indice su `Esame` per ottenere miglioramenti, quindi leggere e trovare prima gli esami di DB2 e poi filtrare gli studenti.

Abbiamo quindi

$$C_2 = \underbrace{100}_{\text{Scan di Corso}} + \underbrace{3}_{\text{Accesso B+tree a Esame}} + \underbrace{1000}_{\text{Esami di DB2}} \cdot \underbrace{2}_{\text{Accesso B+tree a Studente}} \approx 2.1K$$

3. Oltre ai due indici su `Esame`, si ha un indice su `Studente` per gli attributi `(Cognome, Nome, DataNascita)`

La strategia ai punti 1 e 2 sono ancora valide.

Vediamo ora cosa succede se sfruttiamo questo nuovo indice su `Studente`. Possiamo ottenere gli studenti Rossi direttamente tramite questo indice con un singolo accesso all'albero B+, e successivamente andare a prendere i relativi esami e corsi come era stato fatto per C_1.

Abbiamo quindi

$$C_3 = \underbrace{2}_{\text{Accesso B+tree a Studente}} + \underbrace{100}_{\text{Studenti 'Rossi'}} \cdot \left(\underbrace{3}_{\text{Accesso B+tree a Esame}} + \underbrace{10}_{\text{Esami per studente}} \cdot \underbrace{2}_{\text{Accesso B+ tree a Corso}} \right) \approx 2.3K$$