Università degli Studi di Padova

Dipartimento di Matematica

Laurea Triennale in Informatica

Basi di Dati - Laboratorio 6

Ottimizzazione delle query

Massimiliano de Leoni	deleoni@math.unipd.it
Alessandro Padella	alessandro.padella@phd.unipd.it
Samuel Cognolato	samuel.cognolato@studenti.unipd.it

Indice

1 Ripasso degli indici

Gli indici rappresentano una via di accesso alternativa ai dati memorizzati nel database senza la necessità di dover modificare il layout del database. Introducendo gli indici all'interno della propria base di dati è possibile rendere le operazioni di lettura più efficienti, in cambio di un costo aggiuntivo per le operazioni di aggiornamento. Essi permettono di migliorare l'efficienza di query specifiche. Per questo motivo la loro implementazione viene affiancata da un'accurata analisi del carico computazionale al fine di individuare la struttura dati più appropriata per il tipo di operazioni e il tipo di dati da ottimizzare.

Gli indici richiedono uno spazio di memoria aggiuntivo e rallentano le operazioni di modifica dei dati, in quanto ogni modifica dei dati richiede l'aggiornamento di tutti gli indici. Inoltre, l'utilizzo di indici non migliora le prestazioni di tabelle di piccole dimensioni. Si ottengono infatti miglioramenti nelle operazioni di lettura quando le colonne indicizzate contengono una grande quantità di dati e pochi valori NULL.

2 Sintassi

- Creazione indice su una singola colonna (normalmente viene indicato con il prefisso `idx_`)

```
01 |      CREATE INDEX index_name ON table_name (column_name);
```

- Creazione indice composito

```
01 |      CREATE INDEX index_name ON table_name (column_1,column_2);
```

- Creazione di un indice unico

```
01 |      CREATE UNIQUE INDEX index_name ON table_name (column_name);
```

- Eliminazione di un indice

```
01 |      DROP INDEX index_name;
```

3 Indice di default (B-tree)

In questa sezione proveremo a creare un indice utilizzando l'implementazione di default offerta da PostgreSQL, ovvero basata su B-tree.

Utilizzando lo stesso database che abbiamo già creato nel terzo laboratorio del corso (`hubs`), andremo a misurare il tempo necessario ad eseguire una specifica query prima e dopo la creazione degli indici. È importante notare che, in scenari reali, le dimensioni dei database sono ben maggiori rispetto a quelle utilizzate a fine esemplificativo in questo laboratorio: la differenza di prestazioni nell'utilizzo degli indici in tali scenari sarebbe quindi molto più elevata ed evidente.

Nel nostro caso di studio gli indici verranno creati sulle colonne della tabella *legs*. La query da eseguire sarà la seguente:

```
01 |  SELECT *
02 |  FROM legs
03 |  WHERE destination <> 'MALPENSA AIRPORT'
04 |  ORDER BY destination;
```

Nella Figura 1 osserviamo il tempo d'esecuzione della query prima che vengano creati gli indici.

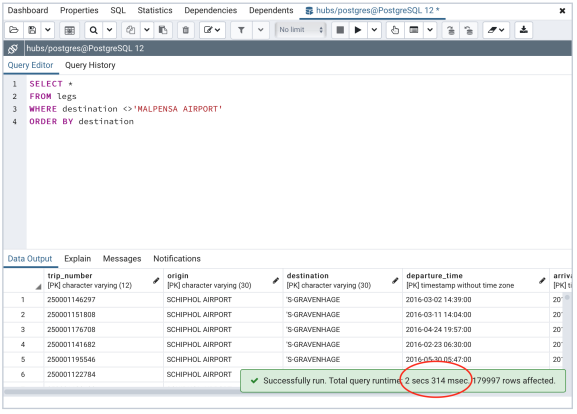

Figura 1: Esecuzione query senza indici

Ora creiamo degli indici sulle colonne che vengono utilizzate nella query, in questo caso solo la colonna *destination*:

```
01 |  CREATE INDEX idx_dest ON legs (destination);
```

Nella Figura 2 possiamo notare le performance temporali di esecuzione della query che utilizza l'indice appena creato. Le performance sono evidentemente migliorate. Dobbiamo però sottolineare che le performance ottenute sono legate all'hardware sul quale si appoggia il computer utilizzato nei nostri test. La stessa query in computer diversi, potrebbe dare risultati molto diversi.

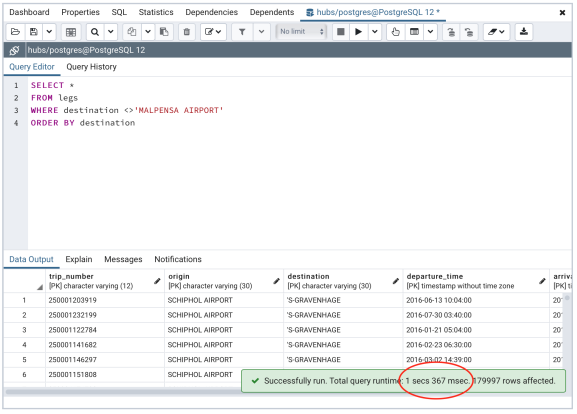

Figura 2: Esecuzione query con indici

Tuttavia, gli indici occupano memoria. Selezionando l'indice precedentemente creato nel menù di sinistra ((1) in Figura 3) e selezionando Statistics dal menù in alto ((2) in Figura 3), è possibile visualizzare le statistiche relative all'indice, tra cui la dimensione dello stesso ((3) in Figura 3).

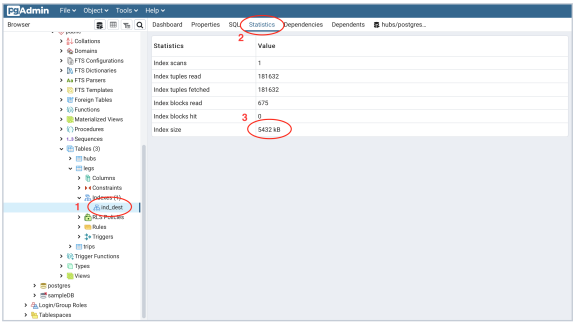

Figura 3: Visualizzazione dell'occupazione di memoria dell'indice su `destination`.

Come menzionato in precedenza, gli indici permettono di ottimizzare il tempo di esecuzione delle query. Tuttavia, ad ogni aggiornamento della tabella su cui l'indice è definito, esso deve essere ricalcolato e pertanto tutte le operazioni di inserimento e update richiedono un tempo maggiore di esecuzione. Ad esempio, utilizzando sempre il medesimo database, eseguiamo un update senza definire alcun indice.

```
01 |  UPDATE legs
02 |  SET origin = 'ADUN'
03 |  WHERE destination = 'MALPENSA AIRPORT'
```

Questo aggiornamento ha effetto su circa 200000 record di record ed impiega circa 200 millisecondi, come mostrato in Figura 4.

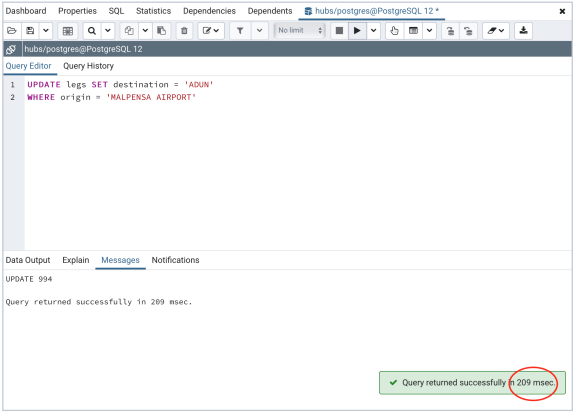

Figura 4: Tempo di update senza l'utilizzo di indici

Ora definiamo un nuovo indice sulla tabella `legs` ed in particolare sulla colonna `destination`. Eseguiamo nuovamente lo stesso update, modificando il valore di aggiornamento (questa volta inserendo "ABBADIA" come valore per il campo `origin`). In questo caso l'aggiornamento richiede

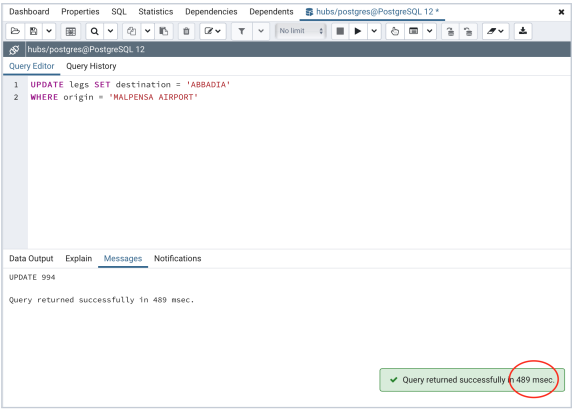

Figura 5: Tempo di update con l'utilizzo di un indice sulla colonna `destination`

circa 500 millisecondi per essere completato, ovvero oltre il doppio del tempo rispetto a prima (Figura 5).

4 Indice basato su hashing

In PostgreSQL è possibile specificare il tipo di implementazione con cui si vuole creare l'indice. In particolare, ora proveremo ad utilizzare il metodo basato su hash anziché su B-tree. Innanzitutto, eliminiamo l'indice creato:

```
01 | DROP INDEX idx_dest;
```

Proviamo ad eseguire la seguente query:

```
01 | SELECT *
02 | FROM legs
03 | WHERE destination = 'SCHIPHOL AIRPORT'
04 | ORDER BY destination;
```

tenendo traccia del tempo richiesto ad eseguire tale interrogazione in assenza di indici.
Creiamo ora lo stesso indice della sezione precedente ma usando hash. La sintassi è la seguente:

```
01 | CREATE INDEX idx_dest ON legs USING hash (destination);
```

Confrontiamo il tempo di esecuzione della query che utilizza l'operatore `<>` e quella che usa `=`.

Eseguendo la query:

```
01 | SELECT *
02 | FROM legs
03 | WHERE destination <> 'MALPENSA AIRPORT'
04 | ORDER BY destination;
```

non dovremmo vedere un particolare miglioramento rispetto all'esecuzione senza indici. Questo perché l'implementazione basata su hash è particolarmente efficace nel caso di interrogazioni che utilizzano l'uguaglianza.

NOTA: provate a cambiare il nome dell'aeroporto poiché PostgreSQL potrebbe tenere in cache risultati di query effettuate in precedenza.

Proviamo ora:

```
01 | SELECT *
02 | FROM legs
03 | WHERE destination = 'SCHIPHOL AIRPORT'
04 | ORDER BY destination;
```

Qui, l'indice basato su hash performa meglio, e si nota un evidente abbassamento dei tempi di esecuzione.

NOTA: questa differenza di tempo non è garantito essere così evidenti in un database di queste dimensioni.

5 Esercizio

Per esercitarsi sugli indici useremo un nuovo database che riguarda la gestione dei dipendenti di una azienda. Questo database è stato generato automaticamente e potrebbe contenere alcune inconsistenze. Tuttavia, è adatto per esercitarsi sugli indici.

Scaricate il file `employees.zip` da moodle e decomprimetelo. Da **pgAdmin** create un nuovo DB con un nome qualsiasi (e.g., "BigDB"). Per creare e popolare il database, eseguite i comandi del file employees.sql.

NOTA: Si consiglia di caricare i file da interfaccia grafica. Fare attenzione che l'apertura e l'esecuzione del file possono impiegare diversi minuti.

Una volta creato e popolato il DB, eseguire la seguente interrogazione:

```
01 | SELECT DISTINCT first_name, last_name
02 | FROM employees e JOIN salaries s ON (e.emp_no=s.emp_no)
03 | WHERE salary > 80000
```

Si chiede di creare un indice che ottimizzi tale interrogazione e di spiegare il perchè della scelta del tipo di indice.