7. Übungsblatt

Ausgabe: 10. Dezember 2013 · Besprechung: 18. Dezember 2013

1 Einleitung

Bei diesem Aufgabenblatt wollen wir die Nutzung und die Effizienz von Bitmap-Indizes am Beispiel des TPC-H-Benchmarks zeigen.

Auf der Kurswebseite finden Sie ein Archiv mit C++-Code, den Sie im Laufe dieses Aufgabenblatts nutzen und ergänzen werden. Außerdem werden wir wieder auf den Datengenerator aus dem TPC-H-Benchmark zurückgreifen, den Sie schon auf dem ersten Aufgabenblatt kennengelernt haben.

2 Anfragen

Zunächst wollen wir einfache Anfragen an die Tabelle `lineitem` des Benchmarks stellen und diese naiv auswerten.

Dazu sind im bereitgestellten Source-Code bereits vier Funktionen `query1()`, `query2()`, `query3()` und `query4()` vorgesehen und auch teilweise schon implementiert.

1. Erzeugen Sie mit dem Datengenerator eine Datei `lineitem.tbl`. Kompilieren Sie den bereitgestellten Quellcode.

   Sie können jetzt das Programm `rawDB` aufrufen. Es wird die Datei `lineitem.tbl` einlesen und als "Tabelle" im Hauptspeicher repräsentieren. Anschließend werden nacheinander die vier Anfragen aufgerufen. Dabei werden Ausführungszeiten gemessen.

2. Ergänzen Sie die fehlenden Implementationen von `query2()`, `query3()` und `query4()`. Insgesamt wollen wir folgende vier Anfragen untersuchen:

   Anfrage 1:

   ```
   SELECT  COUNT(*)
     FROM  lineitem
    WHERE  linenumber = 3 OR linenumber = 5
   ```

   Anfrage 2:

   ```
   SELECT  COUNT(*)
     FROM  lineitem
    WHERE  linenumber = 5 AND shipmode = 'RAIL' AND quantity = 10
   ```

Anfrage 3:

```
SELECT  COUNT(*)
  FROM  lineitem
 WHERE  linenumber IN (2, 4) AND shipmode IN ('SHIP', 'TRUCK')
```

Anfrage 4:

```
SELECT  SUM(extendedprice * (1.0 - discount / 100.0))
  FROM  lineitem
 WHERE  returnflag = 'R' AND shipmode IN ('REG AIR', 'AIR')
   AND  quantity BETWEEN 5 AND 10
```

3 Bitmap Indexing

Im Code ist bereits ein Framework für Bitmap-Indizes vorbereitet. Der Einstiegspunkt hierfür ist die Klasse `BitmapIndex<T>`, parameterisiert bezüglich des zu indizierenden Spaltentyps *T*.

```
class BitmapIndex<T> {
    BitmapIndex (FieldType<T> fieldType, uint32_t numrows);
    uint32_t          buildIndex (lineitem_t *inputtable);
    const BitVector & getBitVector (const T & fieldValue);
};
```

Die Funktion `buildIndex()` liest die entsprechende Spalte, erstellt einen Bitmap-Index und liefert die Anzahl 32-bit-Words zurück, die für den Index benötigt wurden. Die Funktion `getBitVector()` liefert für einen gegebenen Attributwert den entsprechenden Bitvector zurück.

Ein neuer Bitmap-Index kann damit wie folgt erstellt werden (z. B. für eine Spalte des Typs `char[11]`):

```
FieldType<char[11]>    field_shipmode (10, offsetof (lineitem_t, shipmode));
BitmapIndex<char[11]>  bitmap_idx_shipmode (field_shipmode, numrows);
uint32_t               numWords = bitmap_idx_shipmode.buildIndex (inputtable);
```

Dabei sei `10` die Kardinalität der entsprechenden Spalte[1]

Anschließend kann z. B. Anfrage 1 mit Hilfe des Bitmap-Index ausgewertet werden:

```
uint32_t
query1_with_index (...)
{
```

[1] In der Spalte *shipmode* stehen nur feste Werte. Im Code wurde ein wenig getrickst, um die entsprechenden Zeichenketten in eine eindeutige Zahl zwischen 0 und 9 umzurechnen.

```
    const BitVector & vec_ln3 = bitmap_idx_linenumber.getBitVector (3);
    const BitVector & vec_ln5 = bitmap_idx_linenumber.getBitVector (5);
    BitVector res = vec_ln3 | vec_ln5;
    return res.count ();
}
```

3. Ergänzen Sie fehlenden Teile zur Implementation des Bitmap-Index in den beiden Dateien `PlainBitVector.h` und `PlainBitVector.cpp`. Die Stellen an denen Sie Code ergänzen müssen sind entsprechend mit `/* TODO: ... */` markiert.

   Zur Implementation könnten Ihnen insbesondere die folgenden Klassen aus der C++-Standardbibliothek hilfreich sein:

   `std::vector<`*T*`>` Diese Klasse stellt eine wesentlich mächtigere und flexiblere Alternative zu Arrays zur Verfügung. Die Klasse wird parameterisiert bezüglich des Element-Typs. Sie können `std::vector<`*T*`>` anschließend wie ein gewöhnliches Array verwenden, allerdings wird das Array bei Bedarf automatisch vergrößert.[2]

   `std::bitset<`*N*`>` Diese Klasse ermöglicht es Ihnen, einigermaßen elegant mit Bit-Feldern zu operieren. Anstelle z. B. die einzelnen Bits aus einem `uint32_t` "herauszufummeln" können Sie bei einem `std:bitset<32>` bequem die einzelnen Bits ansprechen.

4. Jetzt können Sie auch die Anfragen 2-4 unter Verwendung eines Bitmap-Index formulieren. Beobachten Sie danach, wie sich die Effizienz mit/ohne Verwendung des Bitmap-Index verhält.

4 WAH-Komprimierte Bitmap-Indizes

Der zur Verfügung gestellte Code ist vorbereitet für zwei alternative Implementierungen des Bitmap-Index. Wenn Sie Ihren Code kompilieren mit

```
make WAH=1
```

wird nicht die Klasse `PlainBitVector` eingebunden sondern die Klasse `WAHBitVector` aus den Dateien `WAHBitVector.h` und `WAHBitVector.cpp`. In letzteren können Sie — analog zur obigen Bitmap-Index-Implementation — Bitvektoren mit Hilfe der des WAH-Kompressionsverfahrens implementieren. Auch in diesen beiden C++-Dateien wurden die entsprechenden Stellen im Code wieder annotiert.

5. Ergänzen Sie die fehlenden Implementationen in `WAHBitVector.h` und `WAHBitVector.cpp`. Vergessen Sie nicht, Ihren Code jeweils mit `make WAH=1` zu kompilieren, damit Ihr Code auch tatsächlich ausgeführt wird.

[2] `std::vector<bool>` ist in der Klassenbibliothek typischerweise als Spezialisierung implementiert und damit sehr effizient.

6. Vergleichen Sie die gemessenen Zahlen (Speicherverbrauch, Performanz) mit/ohne Verwendung von Bitmap-Indizes, sowie mit/ohne Kompression.

Hinweise

In der Ausgabe des Programms werden jeweils die Anfrageergebnisse mit ausgegeben. Diese sind für die Bearbeitung nicht unmittelbar relevant, jedoch helfen Sie Ihnen, die Korrektheit Ihrer Implementation zu überprüfen.

Zur Implementation der Bitmap-Indizes gehören insbesondere Operatoren, um Bitvektoren logisch zu verknüpfen (im Code ist das durch Überladen der entsprechenden Operatoren gelöst). Logische Operationen lassen sich auf WAH-komprimierten Daten besonders effizient ausführen. Entsprechende Algorithmen und Hinweise dazu finden Sie in der Originalpublikation [1].

Literatur

[1] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems (TODS)*, 31(1):1--38, March 2006.