Prof. Dr. Stefan Brass
Institut für Informatik
MLU Halle-Wittenberg

22. Januar 2016

Datenbanken II B: DBMS-Implementierung
— Hausaufgabe 10 —

Es sollen nun die entwickelten Klassen für Datenbank-Blöcke, den Puffer-Manager, und den Datei-Zugriff genutzt werden, um einfache Relationen zu implementieren. Die Relationen sollen n Spalten vom Typ `int` haben. Ausserdem soll jede Relation in einer eigenen Datei gespeichert werden.

a) Definieren Sie eine Klasse `rel_c` für Relationen. Die Klasse soll folgende Schnittstelle haben:

- Ein Konstruktor `rel_c()` ohne Parameter. Das Objekt ist zunächst in geschlossenem Zustand, so dass viele Operationen nicht anwendbar sind.
- `open(id, filename)` zum Öffnen der Datei `filename` und Laden von Basisdaten der Relation (Relationenname und Anzahl Spalten). Die `id` ist eine eindeutige Nummer, die für das Datei-Interface zu verwenden ist. (Solche festen IDs wären wohl nur interessant, wenn ROWIDs mit Verweisen zwischen Relationen/Dateien verwaltet werden müssten, die es in dieser Version noch nicht gibt). Falls beim Öffnen ein Fehler auftritt (z.B. Datei existiert nicht, oder Inhalt unpassend), soll `false` geliefert werden. Wenn die Operation erfolgreich war, `true`.
- `create(id, filename, rel_name, num_cols, num_blocks)` zum Anlegen der Tabelle. Dabei soll eine Datei mit dem Dateinamen `filename` angelegt werden. (Die `id` ist entsprechend wieder für das Datei-Interface aus Blatt 8.). Diese Datei soll aus `num_blocks` Blöcken bestehen. Der Relationsname und die Anzahl Spalten müssen in der Datei gespeichert werden. Sie dürfen eventuell zu lange Relationsnamen auf 15 Zeichen kürzen. Falls das Anlegen der Datei erfolgreich war, soll `true` geliefert werden, sonst `false`. Nach dem `create(...)`-Aufruf ist die Relation offen, so dass z.B. `insert(...)` verwendet werden kann.
- `name()` liefert den Namen der Relation (bzw. den Nullzeiger, falls die Datei nicht offen ist).
- `num_cols()` liefert die Anzahl Spalten der Relation (bzw. `-1`, falls die Datei nicht offen ist).
- `close()` schließt die Datei und stellt sicher, dass spätestens jetzt alle veränderten Blöcke geschrieben werden. Falls es dabei zu einem Fehler kommt, wird `false` geliefert, sonst `true`.
- `insert(row)` fügt die Zeile `row` ein (ein Array von `int`-Werten passender Länge, bzw. ein Zeiger vom Typ `int *`). Die Einfügeposition können Sie selbst wählen. Wie bei Relationen üblich, ist nicht verlangt, dass die Einfügung immer am Ende erfolgt. Auch diese Operation liefert einen booleschen Wert, der den Erfolg anzeigt.

b) Definieren Sie eine Klasse `cur_t` für Cursor/Scans/Iteratoren über Relationen, mit denen man also alle aktuell existierenden Tupel einer Relation durchlaufen kann. Die Klasse soll folgende Schnittstelle haben:

- Einen Konstruktor, dem Sie einen Zeiger auf die Relation übergeben, über der der Cursor laufen soll. D.h. der Konstruktor hat einen Parameter vom Typ `rel_ *`.
- Eine Methode `open()`, die den Cursor vor die erste Zeile positioniert.
- Eine Methode `fetch()`, die den Cursor auf die nächste Zeile bewegt. Diese Methode muss auch aufgerufen werden, um die erste Zeile zu laden. Die Methode soll `true` liefern, wenn es noch eine Zeile gab, und `false`, wenn das Ende der Tabelle erreicht ist.

  Der Block, in dem die Zeile steht, muss im Puffer gepinnt sein. Wenn `fetch()` auf den nächsten Block wechselt, muss der vorige Block freigegeben werden.
- Eine Methode `col(i)`, die den Wert der `i`-ten Spalte der aktuellen Zeile liefert (ein `int`). Die Spalten sollen wie in einem Array von `0` an gezählt werden. Es ist ein Fehler, auf eine Spalte zuzugreifen, wenn nicht vorher ein erfolgreiches `fetch()` durchgeführt wurde.
- Eine Methode `delete()`, die die aktuelle Zeile löscht. Im Fehlerfall soll `false` geliefert werden, bei Erfolg `true`.
- Eine Methode `close()`, die den Durchlauf beendet und den aktuell gepinnten Block freigibt.

Sie müssen diese Klasse vermutlich als "`friend`" der Relationsklasse deklarieren, damit sie auch auf private Daten der Relationsklasse zugreifen kann. Das geschieht mit "`friend class cur_c;`" in der Deklaration der Klasse `rel_c`.

c) Schreiben Sie ein Testprogramm, dass eine 100.000 Tupel mit jeweils 4 Spalten einfügt, und sie anschließend wieder liest. Prüfen Sie, dass die richtigen Werte gelesen werden. Wenn möglich, messen Sie die Laufzeit für beide Operationen auf dem Rechner, der für diese Vorlesung reserviert ist.

d) Wie verhält sich Ihre Implementierung, wenn eine Zeile eingefügt wird, während ein Durchlauf mit einem Cursor läuft? Oder wenn zwei Durchläufe laufen, und einer davon eine Zeile löscht?

e) Wie verhält sich Ihre Implementierung, wenn das Programm beendet wird, ohne dass `close()` für die Relation aufgerufen wurde? Ist es möglich, dass die Datenstrukturen in der Datei in einem inkonsistenten Zustand zurückbleiben?

Wie in der Vorlesung besprochen, steht es Ihnen frei, statt dieser Aufgabe einen einfachen B-Baum zu programmieren, der z.B. eine Abbildung von `int`-Werten in `int`-Werte implementiert. Sie dürfen dafür auch eine andere Sprache als C++ verwenden (sofern Sie früher schon eine andere Aufgabe in C++ gelöst haben).