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

03. Juni 2013

Datenbanken II B: DBMS-Implementierung
— Hausaufgabe 8 —

In dieser Aufgabe soll ein Block mit relationalen Daten, d.h. Tabellenzeilen implementiert werden. Dabei reicht es, Zeilen fester Länge zu implementieren. Als Datentypen für Spalten sollen `int` (32 Bit) und Strings fester Länge (`char[`n`]`, `CHAR(`n`)`) implementiert werden. Informationen wie Offset und Länge der Spalten stammen aus dem Data Dictionary, das nächste Woche implementiert werden wird — unter Verwendung der Datenblöcke, die Gegenstand dieses Übungsblattes sind. Die entsprechenden Werte für das Data Dictionary selbst sind fest im Programm eingebaut (sonst wären die Abhängigkeiten ja zyklisch).

Wie immer ist die hier genannte Schnittstelle nur als Vorschlag zu verstehen. Sie müssen eine Menge von Tabellenzeilen im Block, sowie den Zugriff auf Spalten innerhalb einer Zeile implementieren, aber Sie können die genauen Methoden und Parameter auch anders wählen. Das wäre insbesondere nötig, wenn Sie Zeilen variabler Länge implementieren wollten (was hier nicht gefordert ist).

Implementieren Sie also eine Klasse `rdblk_c` ("Relational Data Block") als Subklasse von `block_c` mit folgenden Methoden:

- `void init(int block_no, int rowlen)`:
  Initialisiert den Block als Datenblock für eine Relation mit `rowlen` Bytes pro Zeile.
- `str_t onload_check() const`:
  Prüft die Datenstrukturen im Block nach Laden von der Platte. Im positiven Fall wird ein Nullzeiger zurückgeliefert, im negativen Fall eine Fehlermeldung.
- `int num_rows() const`:
  Liefert die aktuelle Anzahl Zeilen im Block (inklusive Zeilen, die als gelöscht markiert sind).
- `bool has_space() const`:
  Hat dieser Block doch Platz für eine weitere Zeile?
- `int freelist_next() const`:
  Liefert die Nummer des nächsten Blocks in der Liste von Blöcken mit freiem Speicherplatz für diese Relation.
- `void set_freelist_next(int block_no)`:
  Setzt den Verkettungszeiger für die Liste von Blöcken mit freiem Speicherplatz für diese Relation.
- `rptr_t first_row()` const:
  Liefert einen Zeiger auf die erste Zeile im Block. Sie können einen extra Typ `rptr_t` für Zeiger auf Zeilen anlegen, oder z.B. auch `char *` oder `int *` verwenden (je

nachdem, wie Sie den Datenbereich im Block strukturiert haben). Falls der Block keine Zeilen enthält, liefern Sie einen Null-Zeiger. In meiner Implementierung werden gelöschte Zeilen automatisch übersprungen. Wenn Sie das anders machen wollen, sollten Sie klar dokumentieren, dass der Aufrufer dafür verantwortlich ist. Der Vorteil des automatischen Überspringens ist, dass man die Existenz gelöschter Zeilen nach außen verbergen kann.

- `rptr_t next_row(rptr_t row)` const:
  Gegeben ein Zeiger auf eine Zeile im Block, wird ein Zeiger auf die nächste Zeile geliefert (wieder mit Überspringen gelöschter Zeilen). Falls der Block keine weiteren Zeilen enthält, liefern Sie einen Null-Zeiger.

- `rptr_t nth_row(int n)` const:
  Dies liefert die `n`-te Zeile im Block (bzw. einen Null-Zeiger, wenn Sie gelöscht ist). Diese Methode dient der Implementierung von ROWIDs. Der Wert `n` bleibt also stabil für die Dauer der Existenz einer Zeile. Er wird nicht angepasst, wenn vor der Zeile eine Zeile gelöscht wird.

- `int get_int(rptr_t row, int offset)`
  Dies liefert den Wert einer ganzzahligen Spalte innerhalb einer Zeile. Die Spalte ist durch einen Offset-Wert angegeben.

- `str_t get_str(rptr_t row, int offset)`:
  Dies liefert den Wert einer String-wertigen Spalte innerhalb einer Zeile. Die Spalte ist durch einen Offset-Wert angegeben.

- `void set_int(rptr_t row, int offset, int val)`
  Dies setzt den Wert einer ganzzahligen Spalte.

- `void set_str(rptr_t row, int offset, str_t val)`:
  Dies setzt den Wert einer String-wertigen Spalte.

- `void delete_row(rptr_t row)`:
  Dies löscht eine gegebene Zeile.

- `rptr_t start_insert()`:
  In meinem bisherigen Programm geschieht das Einfügen von Zeilen in drei Schritten: (1) Mit dieser Methode wird Speicherplatz für die Zeile reserviert, sie ist aber noch "unsichtbar", d.h. wird ähnlich wie eine gelöschte Zeile übersprungen ("temporäre Zeile"). (2) Mit den `set`-Methoden werden die Attributwerte eingefügt. (3) Mit der Methode `finish_insert` wird die Einfügung abgeschlossen, und die Zeile sichtbar.

  Eine Alternative, die vermutlich einfacher ist, wäre dass die Zeile gleich eingefügt wird und sichtbar ist. Die Attributwerte der neuen Zeile sind all `0` bzw. der leere String. Eine weitere Alternative wäre, dass der Aufrufer dafür verantwortlich ist, die Zeile im Hauptspeicher zusammen zu bauen, und sie dann als Ganzes in den Block gespeichert wird. Dann braucht man aber Speicherplatz außerhalb des Block-Puffers in unbekannter Größe.

- `finish_insert(rptr_t row)`:
  Siehe Erklärung bei der vorigen Methode.