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

19. Januar 2012

Datenbanken II B: DBMS-Implementierung
— Hausaufgabe 10A —

Implementieren Sie eine Klasse für Tupel/Tabellenzeilen, die Sie zusammen mit den Relationenscans aus Aufgabe 10B (s.u.) verwenden können. Die Klasse `row_c` soll u.a. folgende Methoden haben:

- `row_c(rel_t rel)`
  Konstruktor, erzeugt nicht initialisiertes Tupel-Objekt für eine Relation. Bevor man mit dem Tupel arbeiten kann, muß das Objekt an ein Tupel aus der Datenbank gebunden werden. Das geht z.B. mit der Methode `fetch` des Relationenscans (s.u.). Alternativ kann man auch ein neues Tupel mit der Methode `create()` dieser Klasse anlegen.
- `int get_int(col_int_t col)`
  Liefert den Wert der Spalte `col` vom Typ `int`. Vorausgesetzt ist, dass das Objekt an ein Tupel aus der Datenbank gebunden wurde (oder ein neu erzeugtes Tupel).
- `str_t get_str(col_str_t col)`
  Liefert den Wert der Spalte `col` vom Typ `str_t` (String).
- `bool create()`
  Erzeugt ein neues Tupel. Beim Zugriff auf Spalten bekommt man zunächst die Zahl 0 bzw. den leeren String. Sie können selbst entscheiden, ob die Erzeugung des neuen Tupels sofort auf die Relation durchschlagen soll, oder das Tupel zuerst im Hauptspeicher konstruiert wird, und dann mit einer zusätzlichen Methode `save()` abgespeichert werden muß. Wenn Sie Tupel fester Länge verwenden, wäre die erste Methode wohl einfacher, da Sie nicht extra dynamischen Speicher für das Tupel brauchen. Man könnte auch darüber nachdenken, die Tupellänge zu begrenzen, z.B. auf 512 oder 1024 Byte. In jedem `row_c` Objekt so viel Speicher vorzusehen, könnte aber doch ineffizient sein, da Tabellenzeilen auch sehr kurz sein können, und man u.U. für jedes Tupel der Relation ein extra `row_c`-Objekt anlegen will (wenn man z.B. die ganze Tabelle im Hauptspeicher als C++-Objekte haben möchte).
- `void set_int(col_int_t col, int val)`
  Diese Methode ändert den Wert einer Spalte vom Typ `int`. Wieder können Sie selbst entscheiden, ob die Änderung sofort wirksam wird, oder zunächst auf einer Kopie des Tupels arbeitet.
- `void set_str(col_str_t col, str_t val)`
  Diese Methode ändert den Wert einer Spalte vom Typ `str_t` (String). Natürlich darf der String nicht länger sein als die für die Spalte deklarierte Maximallänge. Sie können selbst entscheiden, ob Sie bei Überschreitung der Länge (1) eine Exception

  auslösen, (2) den String stillschweigend kürzen, oder (3) den Rückgabetyp in `bool` ändern und damit anzeigen, ob dieser Fehler aufgetreten ist.
- `void close()`
  Löst die Bindung von `row_c`-Objekt und Tupel aus der Datenbank. Wenn man das Tupel-Objekt z.B. als Zeiger in den Pufferspeicher implementiert hat, würde hier `unpin()` aufgerufen.
- `~row_c()`
  Sie könnten außerdem den Destruktor der Klasse so implementieren, dass er das Tupel freigibt (also implizit `close()` aufruft), falls das Objekt noch an ein Tupel gebunden ist.

Außer diesen Methoden brauchen Sie noch eine Methode (z.B. `bind` oder `set_row`), mit der `fetch` (s.u.) das Tupel an eine Relation binden kann. Der Benutzer der kleinen Datenbank wird diese Methode natürlich nicht direkt aufrufen.

Schließlich würde man natürlich noch eine Methode zum Löschen von Tabellenzeilen brauchen, die wir aber zur Vereinfachung (zunächst) weglassen. Eine weitere nützliche Funktionalität wäre, dass man das Tupel zu einer gegebenen TID/ROWID laden kann. Dann müßte man auch die TID/ROWID des aktuellen Tupels abfragen können.

— Hausaufgabe 10B —

Implementieren Sie eine Klasse `rscan_c` für Relationenscans. Die Klasse sollte folgende Methoden haben:

- `rscan_t(rel_t rel)`
  (Konstruktor)
- `bool open()`
  Öffnet einen Full-Table-Scan über der Relation. Die aktuelle Position ist dann vor dem ersten Tupel.
- `bool fetch(row_t row_buf)`
  Füllt das Objekt `row_buf` der Klasse `row_c` mit dem aktuellen Tupel der Relation (bzw. bindet das Objekt an das aktuelle Tupel — es muß nicht unbedingt aus dem Block im Pufferspeicher heraus kopiert werden). Liefert `true`, falls es noch ein weiteres Tupel gab, sonst `false`.
- `void close()`
  Schließt den Scan wieder.

Sie können selbst entscheiden, was passieren soll, wenn ein Tupel in die Relation eingefügt wird, während ein Scan über der Relation läuft. Das Programm darf aber auf keinen Fall abstürzen. Es gibt also (mindestens) drei Möglichkeiten:

1. Das neue Tupel wird bei einem laufenden Scan nicht geliefert.

2. Neue Tupel werden bei einem laufenden Scan immer ganz am Ende geliefert.
3. Es ist nicht vorhersehbar, ob ein neues Tupel noch von dem Scan erfasst wird, oder nicht.

In diesem Zusammenhang können Sie auch darüber nachdenken, ob Ihre Lösung auch funktionieren würde, wenn es später Löschungen gibt.

— Hausaufgabe 10C —

Betrachten Sie noch einmal die Anfrage aus Aufgabe 8B:

```
SELECT EMPNO, SAL, DNAME
FROM   EMP E, DEPT D
WHERE  E.DEPTNO = D.DEPTNO
AND    E.MGR = 7839
ORDER  BY SAL
```

Welchen Auswertungsplan verwendet Oracle? Legen Sie dann einen sehr gut passenden Index an. Ändert sich der Auswertungsplan dadurch?

— Hausaufgabe 10D —

Geben Sie mit einer SQL-Anfrage an das Data Dictionary alle Indexe aus, die es über den Tabellen `EMP` und `DEPT` gibt. Genauer geben Sie bitte folgende Spalten aus:

1. Den Namen der Tabelle
2. Den Namen des Indexes
3. Die Information, ob der Index ein `UNIQUE`-Index ist, oder nicht.
4. Die Position der Spalte
5. Die indizierte Spalte

Für einen Index über einer Kombination von n Spalten würden Sie also n Ausgabezeilen bekommen (zum Teil mit redundanter Information). Sortieren Sie das Anfrage-Ergebnis bitte übersichtlich. In SQL*Plus können Sie mit `BREAK ON` A erreichen, das der Wert in der Spalte A nur angezeigt wird, wenn er sich von dem in der vorigen Zeile unterscheidet. Mit `BREAK ON` A `SKIP 1` wird zusätzlich eine Leerzeile eingefügt.

Für einen der Indexe ermitteln Sie bitte zusätzlich die Höhe des B-Baums.