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

01. Juli 2013

Datenbanken II B: DBMS-Implementierung
— Hausaufgabe 11A —

Implementieren Sie eine Klasse für Tupel und Relationenscans: Die Objekte haben die Funktionalität eines Relationenscans (Iterator über Tupel), und erlauben den Zugriff auf das aktuelle Tupel im Scan. Da alle Schnittstellen in dieser Übung nur als Vorschlag zu verstehen sind, können sie die beiden Teile auch trennen. Wenn man z.B. alle Tupel einer Relation als `C++`-Objekte im Hauptspeicher verfügbar machen will, wäre es eventuell günstiger, reine Tupel-Objekte zu haben.

Die Klasse `rscan_c` soll u.a. folgende Methoden haben:

- `rscan_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`. Alternativ kann man auch ein neues Tupel mit der Methode `start_insert()` dieser Klasse anlegen.

- `void open()`
  Diese Methode beginnt einen Scan über die Relation. Setzt die Position für `fetch()` vor das erste Tupel der Relation.

- `bool fetch()`
  Läd das nächste Tupel der Relation in den Puffer (bzw. beim ersten Aufruf nach `start_scan()` das erste Tupel). Falls es kein nächstes Tupel mehr gibt, wird `false` geliefert, sonst `true`. Bei erfolgreichen Aufruf kann man anschließend auf die Datenwerte des Tupels mit `get_int()` und `get_str()` zugreifen.

- `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 start_insert()`
  Erzeugt ein neues Tupel. Beim Zugriff auf Spalten bekommt man zunächst die Zahl 0 bzw. den leeren String. (Siehe weitere Hinweise unten.)

- `void finish_insert()`
  Nachdem man die Datenwerte des neuen Tupels mit den Methoden `set_int()` und `set_str()` gesetzt hat, kann man das Tupel mit dieser Methode in der Relation abspeichern.

- `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()`
  Schließt den Scan und 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.
- `~rscan_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.

Hinweise zum Einfügen neuer Tupel:

- Die obige Schnittstelle ist nur als Vorschlag zu verstehen, alternative Ideen sind willkommen (ich bin nicht besonders glücklich mit dieser Schnittstelle). Der Vorteil der vorgeschlagenen Schnittstelle ist, dass man keinen zusätzlichen Speicher zur Pufferung des neuen Tupels braucht: Bei `start_insert()` kann das Tupel schon in der Datenbank angelegt werden, so dass man Speicherplatz im Puffer des entsprechenden Datenblocks verwenden kann. Dies geht so allerdings nur bei Relationen mit Tupeln fester Länge. Bei Tupeln variabler Länge kann man sie erst in die Datenbank einfügen, wenn man ihre Länge kennt.
- Man könnte auch darüber nachdenken, die Tupellänge zu begrenzen, z.B. auf 512 oder 1024 Byte. Dann könnte man jedem `row_c` Objekt so viel Speicher vorzusehen, und darin das Tupel zunächst zusammenbauen, bevor es in die Datenbank gespeichert wird.
- Sie können selbst entscheiden, ob die Erzeugung des neuen Tupels sofort auf die Relation durchschlagen soll (schon bei `start_insert()`). In meiner Implementiertung wird es sofort eingefügt, aber mit einem speziellen Status als "temporäres" Tupel markiert, das bei einem Relationenscan übersprungen wird. Allerdings belegen die temporären Tupel Speicherplatz, und wenn das "`finish_insert()`" nie aufgerufen wird, bleibt der Platz belegt (man müßte eigentlich aufräumen). Die Lösung mit der Konstruktion es Tupels im Hauptspeicher würde dieses Problem vermeiden. Statt `start_insert()` und `finish_insert()` könnte man die Methoden `create()` und `save()` nennen).

Unabhängig vom Problem mit der Speicherung von Tupeln während der Einfügung stellt sich die Frage, was passiert, wenn ein Tupel über einen Scan in die Relation eingefügt wird, während ein zweiter Scan offen ist. Sie können selbst ein sinnvolles Verhalten wählen, 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.

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. Auch dies lassen wir aus Zeitgründen weg.

— Hausaufgabe 11B —

Betrachten Sie noch einmal die Anfrage aus Aufgabe 10B:

```
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 11C —

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.