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

17. Juni 2013

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

In Aufgabe 9A wurde die Entwicklung eines einfachen Systemkatalogs mit Releationen-Objekten begonnen. Jetzt soll die Entwicklung mit Spalten-Objekten fortgesetzt werden. Entwickeln Sie dazu eine Oberklasse `col_c` und Unterklassen `col_int_c` und `col_str_c`. Es soll leicht möglich sein, später weitere Unterklassen zu definieren, z.B. für ROWIDs. Man kann entweder die Oberklasse abstrakt machen, also direkte Instanziierungen verhindern, oder die Unterklassen nur als Abkürzung und Verbesserung der Typsicherheit verstehen, aber die eigentliche Funktionalität in der Oberklasse implementieren. In meiner Implementierung habe ich mich für den zweiten Weg entschieden, und insbesondere einen Konstruktor vorgesehen mit einem zusätzlichen Parameter für den Typ (man könnte diesen Konstruktor `protected` machen, das habe ich nicht getan).

Die Spalten-Objekte haben drei Anwendungen:

- Vor dem Anlegen der Relation werden sie dem Relations-Objekt zugeordnet (das Relationsobjekt muss also eine verkettete Liste von Spalten-Objekten enthalten). Wenn man für die Relation dann `create` aufruft, werden die Spalten mit im Data Dictionary gespeichert.
- Auch vor dem Öffnen einer existierenden Relation ordnet man dem Relationsobjekt Objekte für die Spalten zu, für die man sich später interessiert (das müssen nicht immer alle Spalten sein). Beim Öffnen wird dann geprüft, dass es die Spalten wirklich gibt, und es werden Zugriffsdaten für die Spalten geladen (z.B. Offsets in Tupeln fester Länge).
- Später werden wir u.a. den Full Table Scan implementieren. Dann muß man natürlich auf jeden Spaltenwert des aktuellen Tupels zugreifen können. Dazu gibt man ein Spaltenobjekt an (dieses enthält ja die Zugriffsdaten für die Spalte in einem Tupel der Relation).

Zunächst müssen Sie für die Unterklassen den jeweiligen Konstruktor implementieren. Meine erste Idee sah so aus:

- `col_int_c(rel_t rel, str_t name)`
- `col_str_c(rel_t rel, str_t name, int maxlen)`

Weil ich zyklische Abhängigkeiten zwischen den Klassen vermeiden wollte, habe ich mich dann aber dafür entschlossen, dass Spalten-Objekte keinen Zeiger auf die Relation enthalten. Nur umgekehrt verweist das `rel_c`-Objekt auf die erste Spalte, und jede Spalte auf die nächste Spalte der gleichen Relation. Ich habe daher die Klasse `rel_c` um eine Methode `add_col(col_t col)` erweitert. Die Konstruktoren für die Spalten sind:

- `col_int_c(str_t name)`

- `col_str_c(str_t name, int maxlen)`

Wenn die Spalte einer Relation zugeordnet wird, muß die Relation natürlich die Verkettung entsprechend verwalten. Dazu braucht die Klasse `col_c` folgende Methoden:

- `class col_c *next() const`: Abfragen des Verkettungs-Zeigers ("get")
- `void set_next(class col_c *next_col)`: Setzen des Verkettungs-Zeigers

Außerdem enthält ein Spalten-Objekt die Position der Spalte innerhalb der Relation (1 für die erste/am weitesten links stehende Spalte, 2 für die nächste, u.s.w.):

- `int pos() const`
- `void set_pos(int col_pos)`

Dann muß der Offset der Spalte verwaltet werden, also mit welchem Abstand (in Bytes) vom Beginn des Tupels sie gespeichert wird:

- `int offset() const`: Liefert den Offset ("get"-Funktion)
- `void set_offset(int col_offset)`: Speichert den Offset.

In meiner Implementierung verwendet ich statt `int` einen eigenen Typ `coff_t`, der als `unsigned short int` definiert ist. Unsigned-Typen machen aber relativ viele Schwierigkeiten bei Rechnungen und Vergleichen. Da man den Offset im Data Dictionary speichert, würde man einen Vorteil durch die 16-Bit Größe auch nur haben, wenn man auch Spalten vom Typ "`short int`" hat.

Weitere Attribute, die im Konstruktoir gesetzt werden, und nicht geändert werden können, sind:

- `str_t name() const`: Spalten-Name.
- `dtype_t type() const`: Datentyp der Spalte, `DTYPE_INT` oder `DTYPE_STR`.
- `int maxlen() const`: Maximale Länge der Strings. Liefert einen Fehler bei Zugriff für eine `int`-Spalte.

Sie müssen aber auch die Klasse `rel_c` und hier besonders die Methoden `create` und `open` entsprechend erweitern.

Wenn Sie nicht Subklassen für die verschiedenen Datentypen wollen, sondern nur eine Klasse mit einem Typ-Attribut, dürfen Sie auch diese Variante wählen. Falls der String-Datentyp eine Länge bzw. Maximallänge hat, und der `int`-Datentyp keine Längenangabe braucht, wird dieser Punkt etwas schwieriger. Allgemein ist die Schnittstelle wieder nur als Vorschlag zu verstehen. Z.B. wäre es möglich, Konstruktoren ohne Parameter anzubieten, und die Daten dann später zu setzen. Sie dürfen die maximale Länge von Spaltennamen begrenzen, z.B. auf 18 (wie in SQL-86).

Es wäre später auch wünschenswert, die Spalten einer Tabelle abfragen zu können. Das ist mit der obigen Schnittstelle nicht möglich. Falls Sie das Data Dictionary über normale Relationen implementieren, hat man diese Funktionalität natürlich automatisch ohne zusätzlichen Aufwand. Ansonsten wäre zu überlegen, ob man später einen zusätzlichen Scan speziell für Spalten einführt, oder eine Methode, mit der man die `i`-te Spalte

einer Tabelle (Name und Datentyp) abfragen kann. Die Relationenklasse müsste dann die Möglichkeit bieten, die Anzahl Spalten der Relation abzufragen.

— Hausaufgabe 10B —

Gegeben sei eine Datenbank mit den Tabellen

- EMP(EMPNO, ENAME, SAL, DEPTNO→DEPT, MGR°→EMP)
- DEPT(DEPTNO, DNAME, LOC)

Dies ist eine etwas vereinfachte Version der klassischen Oracle-Beispieldatenbank. Nehmen Sie an, die Angestellten-Tabelle `EMP` enthält 10000 Zeilen, und die Abteilungs-Tabelle `DEPT` 25 Zeilen, und die Angestellten pro Abteilung seien gleichverteilt. Ein Vorgesetzter (`MGR`) habe durchschnittlich 10 Untergebene. Nehmen Sie weiter an, dass 10 Datensätze pro Block gespeichert werden (bei beiden Tabellen). Es sei die folgende Anfrage gegeben:

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

Beschreiben Sie, wie man die Anfrage auswerten kann, wenn man folgende Indexe hat:

a) Keinen.

b) `EMP(EMPNO)` und `DEPT(DEPTNO)` (beide natürlich `UNIQUE`).

c) Wie b), zusätzlich `EMP(MGR)`.

d) Wenn Sie sich einen Index aussuchen dürften, auch einen mit mehreren Attributen, welchen würden Sie wählen? Dabei soll es nur darum gehen, dass die obige Anfrage möglichst schnell abgearbeitet wird.