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

10. November 2011

Datenbanken II B: DBMS-Implementierung — Hausaufgabe 4A —

Definieren Sie eine Klasse `block_c` für Blöcke in der Datenbank. Mindestens muß es eine Methode `type()` geben, die Werte eines Aufzählungstyps `btype_t` liefert. Bisher haben wir nur den "file header block" (`BT_FILE_HEADER`), aber es wird mindestens noch einen "relation data block" (`BT_REL_DATA`) geben, auch "segment header block", "index block" und weitere wären denkbar.

In der Übung sollen verschiedene Möglichkeiten diskutiert werden, wie man mit den verschiedenen Blocktypen umgehen kann. Wenn ein Ziel ist, dass Objekte der Klasse `block_c` immer `VER_BLOCKSIZE` Bytes groß sind, würde es (mindestens) zwei Möglichkeiten geben:

- Die Oberklasse definiert neben dem Typfeld und möglicherweise anderen Daten, die in jedem Block enthalten sein sollen (z.B. Bitmuster vorne und hinten, das bei jedem Schreiben invertiert wird, als Indikator für partiell geschriebene Blöcke), ein großes Array von Zeichen. Es gibt Unterklassen für die einzelnen Blocktypen, die aber keine Attribute hinzufügen, sondern nur Methoden.
- Es gibt für jeden Blocktyp eine eigene Klasse, und `block_c` enthält eine `union` dieser Klassen (wobei eine Komponente ein Zeichenarray ist, das die Mindestgröße `VER_BLOCKSIZE` sicherstellt. Die Klassen für die Blocktypen dürfen dann nicht größer sein. Der "file header block" hätte z.B. im Moment eine sehr kleine Klasse mit nur 1–2 Komponenten (im wesentlichen die Anzahl Blöcke der Datei).

Wenn man aber zulässt, dass `block_c`-Objekte auch kleiner als `VER_BLOCKSIZE` Bytes sein können, gibt es natürlich weitere Möglichkeiten.

— Hausaufgabe 4B —

Beschaffen Sie sich den Artikel "Principles of Database Buffer Management" von Wolfgang Effelsberg und Theo Haerder, erschienen in den ACM Transactions of Database Systems, Vol. 9, No. 4, Dezember 1984, Seiten 560–595. Von Rechnern der Universität aus können Sie die ACM Digital Library kostenlos zugreifen, diesen Artikel bekommen Sie über die URI/DOI:

`http://dx.doi.org/10.1145/1994.2022`

Verschaffen Sie sich einen kurzen Überblick über den Inhalt. Nennen Sie eine Puffer-Verdrängungsstrategie außer LRU, die in dem Artikel behandelt wird, und beschreiben Sie sie mit wenigen Sätzen.

— Hausaufgabe 4C —

Erweitern Sie `ver.h` um eine Konstante `VER_MAXBUFS` für die maximale Anzahl Blöcke, die im Hauptspeicher gepuffert werden (Cache).

Definieren Sie dann eine Klasse `buf_c` mit folgenden Methoden:

- Eine statische Methode (Klassenmethode) `pin`, die zu gegebener File-ID und Blocknummer einen Zeiger auf ein `buf_c`-Objekt liefert, das den betreffenden Block enthält. Falls es keinen freien Puffer gibt, und wegen der Grenze `VER_MAXBUFS` kein weiterer Hauptspeicher mehr angefordert werden kann, soll nach einer beliebigen Verdrängungsstrategie versucht werden, einen Puffer frei zu machen. Falls das nicht möglich ist, soll der Nullzeiger zurückgeliefert werden (bzw. eine Exception ausgelöst werden) und ein Eintrag in die Alert-Datei geschrieben werden (bzw. eine Fehlermeldung auf den Bildschirm ausgegeben werden). Bei Mehrbenutzerbetrieb (den wir noch nicht haben) wäre auch denkbar, einige Zeit zu warten und es dann erneut zu probieren.
- Eine Methode `contents()`, die einen Zeiger auf das `block_c`-Objekt im Puffer liefert.
- Eine Methode `unpin()`, die den Block im Puffer freigibt.
- Eine Methode `set_modified()`, die den Inhalt des Puffers als verändert markiert. Der Puffer kann dann nur wiederverwendet werden, nachdem der Block in die Datei zurückgeschrieben wurde.
- Eine statische Methode `checkpoint()`, die alle veränderten Puffer zurückschreibt.
- Statische Methoden `hits()` und `misses()`, die die entsprechenden Anzahlen ausgeben.

Es gibt mehrere Erweiterungen, die in den Übungen diskutiert werden sollen:

- Man könnte eine Variante der `pin`-Methode schreiben mit einem dritten Argument vom Typ Zeiger auf `buf_c`, das als "hint" verstanden werden kann: Wenn der Aufrufer einen Block schon einmal gepinnt hatte, und ihn dann freigegeben hat (für den Fall, das der Speicher knapp wird), sich aber die Adresse des Puffer gemerkt hat, könnte die Suche nach dem Block eingespart werden.
- Man braucht auch eine Möglichkeit, sich temporären Speicherplatz zu verschaffen, also Blöcke einer temporären Datei, die bei ausreichend Hauptspeicher nie geschrieben werden müssen. Eine einfache, aber etwas eingeschränkte Lösung wäre eine statische Methode `tmp_buf`, die einen leeren und gepinnten Puffer zurückliefert. Wenn man keine andere Identifikation für den Block hat, muß er aber so lange gepinnt bleiben, bis man den Block nicht mehr braucht (dann würde es nie eine temporäre Datei

geben). Damit könnte man vorübergehend leben, aber bei großen temporären Daten und wenig Hauptspeicher kommt diese Methode an Grenzen.

- Auch wenn man keinen Mehrbenutzer-Betrieb macht, möchte man vermutlich Logging und sichere Schreibmöglichkeiten. Die alte Version eines Blockes müßte dann gesichert werden, bevor man den Block überschreibt.

Testen Sie Ihre Pufferverwaltung mit den Zugriffen aus

`http://www.informatik.uni-halle.de/~brass/dbi11/refstring.txt`

und einer Puffergröße von 2000 Blöcken. Wie groß ist die Hit Ratio?