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

13. Mai 2013

Datenbanken II B: DBMS-Implementierung
— Hausaufgabe 5 —

a) 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.

b) 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 Datei (Zeiger vom Typ `file_t`) 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 eine Fehlermeldung 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. Man kann außerdem Methoden für jeden einzelnen Typ von Block vorsehen, z.B. eine Methode `data_block()`, die den im Puffer gespeicherten Block vom Typ `BTYPE_REL_DATA` liefert (oder einen Fehler, wenn der Block nicht den gewünschten Typ hat). Mit solchen Spezial-Methoden würde man die Typ-Prüfung und den "Downcast" auf die Subklasse nur an einer Stelle machen.
- 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 in die jeweilige Datei zurückschreibt.

- Statische Methoden `hits()` und `misses()`, die die entsprechenden Anzahlen ausgeben.
- Eine statische Methode `init()` zur Initialisierung von Datenstrukturen der Klasse. Sie muss einmal vor allen anderen Methoden der Klasse aufgerufen werden.

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.
- Für Full Table Scans wäre es günstig, mehrere Blöcke auf einmal einlesen zu lassen. Es gibt Betriebssystem-Aufrufe, mit denen man die eingelesenen Daten auf mehrere Puffer verteilen kann. Es wäre aber auch zu überlegen, ob man eine gewisse Anzahl direkt hintereinander liegender Puffer zu einem großen "Super-Puffer" zusammenlegen kann, oder einen extra Speicherbereich mit größeren Puffern verwendet.
- Es wäre auch möglich, mehrere Puffermanager haben zu können: Dann würden die oben genannten statischen Methoden zu Methoden der Klasse "Puffermanager" werden, und die Klasse für die einzelnen Puffer (für jeweils einen Block) wäre getrennt.
- 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/dbi13/refstring.txt`

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