ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

Τμήμα Πληροφορικής

ΕΠΛ 646 – Προχωρημένα Θέματα Βάσεων Δεδομένων

ΑΣΚΗΣΗ 1 – Υλοποίηση Δομής Σελίδας Αρχείου Σωρού της Σχεσιακής Βάσης Δεδομένων Minibase

(Heap File layer Page Structure in Minibase)

Διδάσκων: Δημήτρης Ζεϊναλιπούρ

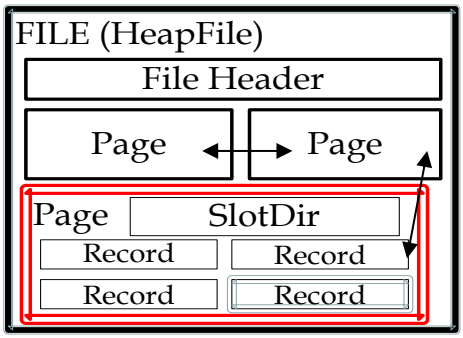
Ημερομηνία Ανάθεσης: Τρίτη, 06/ 02 /2024
Ημερομηνία Παράδοσης: Τρίτη, 20/02/2024 (2 εβδομάδες)

(να παραδοθεί ο εκτυπωμένος κώδικας της κλάσης **HFPage** στο εργαστήριο και να υποβληθεί ηλεκτρονικά μέσω του **Moodle**)

Ι. Στόχος Άσκησης

Ένα **Αρχείο (File)** σε μια βάση δεδομένων είναι μια συλλογή από σελίδες (pages) η κάθε μια εκ των οποίων περιέχει κάποιες εγγραφές (records). Ένα αρχείο υποστηρίζει διάφορες λειτουργίες όπως αυτές της εισαγωγής, διαγράφης και τροποποίησης εγγραφών αλλά και της ανάγνωσης εγγραφών βάση ενός κλειδιού RID (Record ID). Γνωρίζουμε ότι το RID ορίζεται συνήθως ως (PageID, SlotID), όπου PageID προσδιορίζει την σελίδα μέσα στην οποία βρίσκεται η εγγραφή με κλειδί RID και SlotID το εκτόπισμα (offset) μέσα στη σελίδα αυτή. Γνωρίζουμε επίσης ότι η έννοια του Αρχείου σε μια βάση δεδομένων δημιουργείται ουσιαστικά στο επίπεδο πάνω από αυτό του Διαχειριστή Κρυφής Μνήμης - Buffer Manager και κατ' επέκταση και πάνω από τον Διαχειριστή Χώρου Δίσκου - Disk Space Manager, συστατικά τα οποία είδαμε στη διάλεξη.

Το **HeapFile (Αρχείο Σωρού)** είναι ένα ειδικό είδος αρχείου όπου νέες εγγραφές αποθηκεύονται μέσα στο αρχείο με τυχαίο τρόπο (χωρίς κάποια συγκεκριμένη διάταξη). Οι σελίδες ενός τέτοιου αρχείου γίνονται αντιληπτές από τα ανώτερα επίπεδα σαν μια οντότητα (αυτή του αρχείου) μέσω της χρήσης κάποιου κατάλογου (directory) το οποίο αποθηκεύεται μέσα στη κεφαλίδα (header) του αρχείου (η κεφαλίδα δεν θα μας απασχολήσει στην άσκηση αυτή ωστόσο). Επιπρόσθετα, οι σελίδες μπορεί να είναι συνδεδεμένες μεταξύ τους σε μια διπλά συνδεδεμένη λίστα έτσι ώστε να είναι δυνατή η σειριακή προσπέλαση των εγγραφών του αρχείου.



Σχήμα 1: Η δομή ενός Αρχείου Βάσης Δεδομένων το οποίο χρησιμοποιείται σε Αρχεία Σωρού (Heapfiles). Η διπλή γραμμή δείχνει το πλαίσιο αυτής της εργασίας το οποίο είναι η υλοποίηση της **Σελίδας** (HeapPage) ενός HeapFile και όχι του ιδίου του Heapfile.

Έχοντας περιγράψει αναλυτικά το πλαίσιο ενός Αρχείου μια Βάση Δεδομένων ήρθε η ώρα να δούμε το ζητούμενο αυτής της εργασίας.

Σε αυτή τη άσκηση θα πρέπει να υλοποιήσετε τη δομή της σελίδας ενός Αρχείου Σωρού (HeapFile) το οποίο θα υποστηρίζει τόσο εγγραφές σταθερού μήκους (constant-length records) όσο και εγγραφές μεταβλητού μήκους (variable-length records). Σημειώστε ότι η άσκηση ΔΕΝ ζητά να υλοποιήσετε ολόκληρο το αρχείο HeapFile αλλά μονάχα το μέρος της σελίδας του αρχείου της σωρού (δες Σχήμα 1)

Μια σελίδα HeapPage θα δημιουργείται μέσα στο πρόγραμμα σας με κάποιο στατικό τρόπο (δηλαδή δεν θα χρειάζεται να δημιουργήσετε/φορτώσετε μια σελίδα μέσω του Διαχειριστή Κρυφής Μνήμης). Οι έλεγχοι της εργασίας περιλαμβάνουν μεταξύ άλλων την προσθήκη διάφορων τιμών στις σελίδες αυτές και τον έλεγχο κατά πόσο αυτές οι προσθήκες έχουν γίνει με τον ορθό τρόπο. Θα σας δοθούν κάποιες διαδικασίες ελέγχου (unit tests) για να δοκιμάσετε τον κώδικα σας.

II. Προεργασία

Διαβάστε από το βιβλίο του Ramakrishnan & Gehrke, 3ED τα κεφάλαια 9.5 (Implementing Heap Files), 9.6 (Page Formats) και 9.7 (Record Formats). Δώστε ιδιαίτερη έμφαση στη περιγραφή που είναι σχετικά με το πώς εγγραφές μεταβλητού μεγέθους μπορούν να αποθηκευτούν σε μια σελίδα αυλακώσεων (slotted page) και ακολουθήστε αυτή την οργάνωση σελίδων. Θα υλοποιήσετε μόνο την κατηγορία **HFPage** και όχι όλο τον κώδικα του στρώματος Αρχείου Σωρού.

Μελετήστε το εκπαιδευτικό εγχειρίδιο για τη Minibase (minibase.pdf) το οποίο βρίσκεται συνημμένο μαζί με την εργασία. Το εγχειρίδιο θα σας βοηθήσει να καταλάβετε πως δουλεύει ο χειρισμός λαθών (error handling) και ο διαχειριστής αποθήκευσης (Storage Manager) στη Minibase.

III. Μεταγλώττιση και Εκτέλεση

Κατεβάστε το αρχείο as1-skeleton.zip στο λογαριασμό του ΠΚ και αποσυμπιέστε το δίνοντας τις ακόλουθες εντολές:

1. mkdir project
2. cd project
3. unzip as1-skeleton.zip
4. cd as1/

Μετά από αυτό το βήμα πρέπει να έχετε τους ακόλουθους καταλόγους::

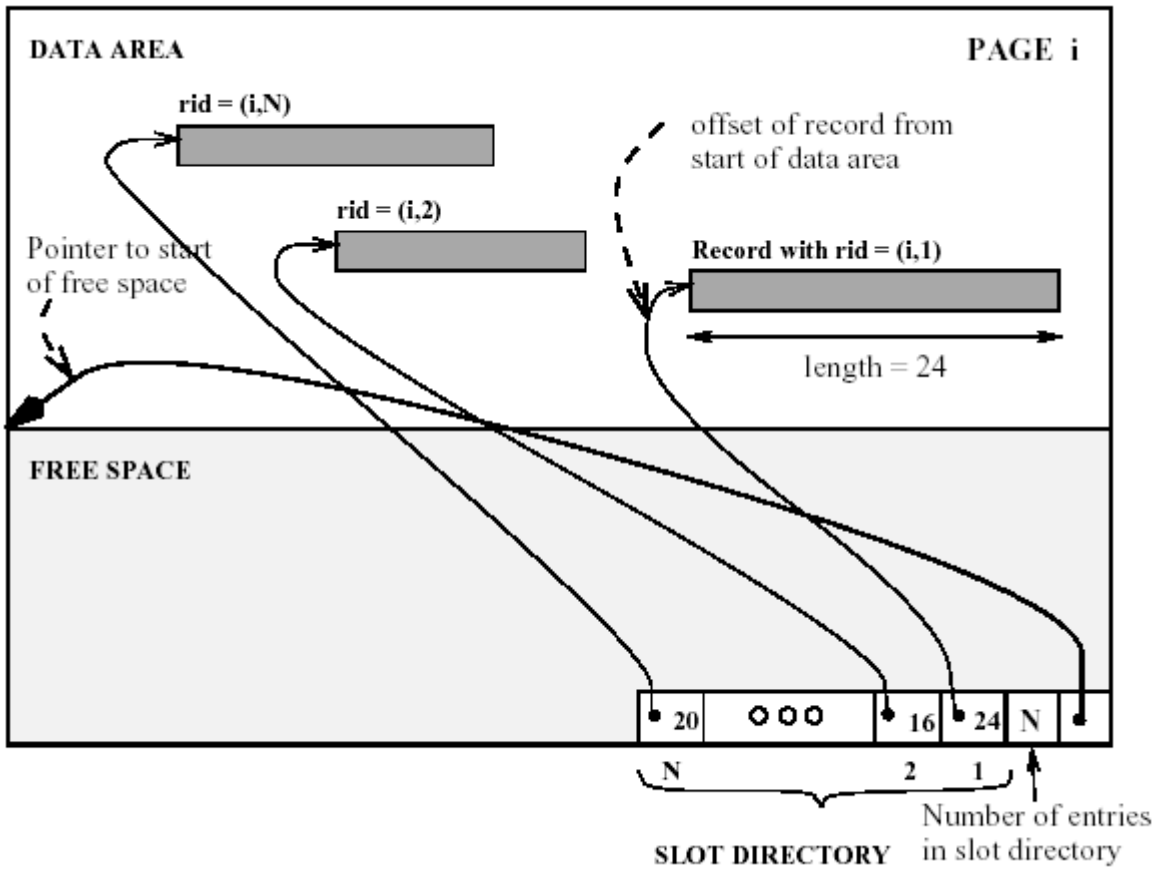
1. include/
2. **src/**

Τώρα μπορείτε να καλέσετε την εντολή **"make depend"** και μετά **"make"** πράγμα το οποίο θα δημιουργήσει το εκτελέσιμο αρχείο ***hfpage***. Εάν δοκιμάσετε να εκτελέσετε το εν λόγω αρχείο θα σας επιστραφεί το μήνυμα ***Segmentation Fault (μετά από***

κάποια αποτελέσματα που θα παρουσιαστούν στον τερματικό). Για να δουλέψει σωστά το πρόγραμμα σας πρέπει να συμπληρώσετε τον κώδικα που λείπει από την κατηγορία **HFPage** (και ότι άλλο χρειαστείτε για την υλοποίηση σας). Οι μέθοδοι ορίζονται (χωρίς το σώμα τους) στο αρχείο ***hfpage.C***. Το αρχείο ***sample_output*** περιέχει παραδείγματα επιτυχημένης εκτέλεσης το προγράμματος για διάφορα δεδομένα εισόδου.

IV. Επισκόπηση Σχεδίασης και Υλοποίησης

Δείτε το αρχείο ***hfpage.h*** του καταλόγου **include/**. Περιέχει τη διεπαφή (interface) για την κατηγορία **HFPage**. Αυτή η κατηγορία υλοποιεί ένα αντικείμενο «σελίδα αρχείου σωρού» ("heap-file page"). Σημειώστε ότι τα προστατευμένα πεδία μέλη (member fields) της σελίδας σας δίνονται. Το μόνο που πρέπει να πρέπει για να κάνετε είναι να υλοποιήσετε τις δημόσιες μεθόδους. Πρέπει να βάλετε τον κώδικά σας στο αρχείο *src/hfpage.C*.



Σημείωση για τον κατάλογο αυλακώσεων (slot directory): Στην πιο πάνω περιγραφή (όπως και στη περιγραφή που δίνεται στο βιβλίο σας), ο κατάλογος αυλακώσεων βρίσκεται στο τέλος της σελίδας, και αυξάνεται προς την αρχή. Επειδή αυτό μπορεί να προκαλέσει άσκοπα σύγχυση, δεδομένου ότι πρέπει να χρησιμοποιηθούν αρνητικά offset στον κατάλογο αυλακώσεων, ο τρέχων ορισμός της κατηγορίας HFPage έχει τον κατάλογο αυλακώσεων στην αρχή της σελίδας, μετά από μερικά σταθερά πεδία μέλη (member fields), και ανάπτυξη προς το τέλος. Αυτό

σημαίνει, εντούτοις, ότι θα πρέπει να γράψετε τον κώδικα σας έτσι ώστε οι εγγραφές να τοποθετούνται αρχίζοντας από το τέλος της σελίδας. Να είστε πολύ προσεκτικοί με την αριθμητική των δεικτών σας.

Επίσης σημειώστε ότι προκειμένου να προστεθεί μια εγγραφή σε μια σελίδα, πρέπει να υπάρχει χώρος για την εγγραφή στην περιοχή δεδομένων, και επίσης χώρος για μια νέα αυλάκωση (slot) στην περιοχή δεδομένων (εκτός και αν συμβαίνει να είναι μια προ-διατιθέμενη αυλάκωση που είναι κενή).

Ακολουθήστε το πρωτόκολλο λαθών της Minibase. Ένα αρχείο παραδείγματος που επεξηγεί τη χρήση του πρωτοκόλλου λαθών είναι διαθέσιμο στο **src/ErrProc.sample**. Καλύπτει ένα μεγάλο μέρος αυτών που πρέπει να ξέρετε για το πρωτόκολλο. Μπορείτε να εξετάσετε το **new_error.h** για περισσότερες λεπτομέρειες (βρίσκεται στο κατάλογο **include/**).

V. Διασύνδεση Δομής Σελίδας Στρώματος Αρχείου Σωρού (Heap File Layer Page Structure)

Οι μέθοδοι που πρέπει να υλοποιήσετε περιγράφονται ακολούθως:

void HFPage::init(PageId pageNo)

```
// This member function is used to initialize a new heap file page with page number
// pageNo. It should set the following data members to reasonable defaults:
// nextPage, PrevPage, slotCnt, curPage, usedPtr, freeSpace.
// You will find the definitions of these data members in hfpage.h.
// The nextPage and prevPage data members are used for keeping track of pages
// in a HeapFile. A good default unknown value for a PageId is INVALID_PAGE,
// as defined in page.h. Note that usedPtr is an offset into the data array, not a pointer.
```

PageId HFPage::getNextPage()

```
// This member function should return the page id stored in the nextPage data member.
```

PageId HFPage::getPrevPage()

```
// This member function should return the page id stored in the prevPage data member.
```

void HFPage::setNextPage(PageId pageNo)

```
// This member function sets the nextPage data member.
```

void HFPage::setPrevPage(PageId pageNo)

```
// This member function sets the prevPage data member.
```

Status HFPage::insertRecord(char* recPtr, int reclen, RID& rid)

```
// This member function should add a new record to the page. It returns OK if
// everything went OK, and DONE if sufficient space does not exist on the page
// for the new record. If it returns OK, it should set rid to be the RID of the new
```

// record (otherwise it can leave rid untouched.) Please note in the parameter
// list **recPtr** is a char pointer and **RID&** denotes passed by reference.
// The Status enumerated type is defined in new_error.h if you're curious about it.
// You may want to look that file over and handle errors in a more informative
// manner than suggested here.

Ένα RID struct ορίζετε ως εξής:

```
Struct RID {

        PageID pageNo;
        int slotNo;

        int operator == (const RID rid) const
        { return (pageNo == rid.pageNo) && (slotNo == rid.slotNo); };

        int operator != (const RID rid) const
        { return (pageNo != rid.pageNo) || (slotNo != rid.slotNo); };

        friend ostream& operator << (ostream& out, const struct RID rid);

};
```

Στη C++, ένα struct είναι συνολικά αθροιστικοί τύποι δεδομένων χτισμένοι χρησιμοποιώντας στοιχεία άλλων τύπων. Το pageNo προσδιορίζει έναν φυσικό αριθμό σελίδας (κάτι που τα στρώματα του διαχειριστή προσωρινής μνήμης και DB καταλαβαίνουν) στο αρχείο. Το slotNo διευκρινίζει μια είσοδο στο πίνακα αυλακώσεων στη σελίδα.

Status HFPage::deleteRecord(const RID& rid)

// This member function deletes the record with RID rid from the page, compacting
// the hole created. Compacting the hole, in turn, requires that all the offsets
// (in the slot array) of all records after the hole be adjusted by the size of the hole,
// because you are moving these records to "fill" the hole. You should leave a "hole"
// in the slot array for the slot which pointed to the deleted record, if necessary, to make
// sure that the rids of the remaining records do not change. The slot array can be
// compacted only if the record corresponding to the last slot is being deleted.
// It returns OK if everything goes OK, or FAIL otherwise. (what could go wrong here?)

Status HFPage::firstRecord(RID& firstRid)

// This routine should set firstRid to be the rid of the "first" record on the page.
// The order in which you return records from a page is entirely up to you.
// If you find a first record, return OK, else return DONE.

Status HFPage::nextRecord(RID curRid, RID& nextRid)

// Given a *valid* current RID, curRid, this member function stores the next RID
// on the page in the nextRid variable. Again, the order of your return records is up to you,
// but do make sure you return each record exactly once if someone continually calls
// nextRecord! Don't worry about changes to the page between successive calls
// (e.g. records inserted to or deleted from the page). If you find a next RID,
// return OK, else return DONE. In case of an error, return FAIL.

Status HFPage::getRecord(RID rid, char * recPtr, int& recLen)

// Given a rid, this routine copies the associated record into the memory address *recPtr.
// You may assume that the memory pointed by *recPtr has been allocated by the caller.
// RecLen is set to the number of bytes that the record occupies. If all goes well,
// return OK, else return FAIL.

Status HFPage::returnRecord(RID rid, char*& recPtr, int& recLen)

// This routine is very similar to HFPage::getRecord, except in this case you do not
// copy the record into a caller-provided pointer, but instead you set the caller's recPtr
// to point directly to the record on the page. Again, return either OK or FAIL.

// DONE is a special code for non-errors that are nonetheless not "OK":
// it generally means "finished" or "not found." FAIL is for errors that happen
// outside the bounds of a subsystem.

int HFPage::available_space(void)

// This routine should return the amount of space available for a new record that
// is left on the page. For instance, if all slots are full and there are 100 bytes of
// free space on the page, this method should return (100 - sizeof(slot_t)) bytes.
// This accounts for the fact that sizeof(slot_t) bytes must be reserved for a
// new slot and cannot be used by a new record.

bool HFPage::empty(void)

// Returns true if the page has no records in it, and false otherwise.

Παραδοτέα

Πρέπει να παραδώσετε **όλα** τα πηγαία αρχεία σας μέσω του Moodle. Καθυστερημένες υποβολές δεν θα γίνονται αποδεκτές.

Παρακαλώ βεβαιωθείτε ότι το πρόγραμμα σας παράγει το αναμενόμενο αποτέλεσμα συγκρίνοντας τα αποτελέσματα σας με αυτά που βρίσκονται στο αρχείο *sample_output*.

Καλή Επιτυχία!