# Τεχνολογία Βάσεων Δεδομένων
## Εργασία 2
### Υλοποίηση δένδρου B+-tree

**Προθεσμία Υποβολής: 6 Ιουνίου 2018 (23:55)**

## 1. Εισαγωγή
Στην εργασία αυτή καλείστε να υλοποιήσετε έναν κατάλογο B+-tree χρησιμοποιώντας τον πηγαίο κώδικα που είχαμε χρησιμοποιήσει και στην πρώτη εργασία. Σε περίπτωση που δεν ολοκληρώσατε την πρώτη εργασία, δίνεται η λύση της πρώτης εργασίας ώστε να μπορέσετε να προχωρήσετε στη δεύτερη εργασία. Ακόμη και αν έχετε ολοκληρώσει την πρώτη εργασία, **παρακαλώ χρησιμοποιήστε τη βάση του κώδικα που δίνεται εκ νέου διότι υπάρχουν μικρές αλλαγές στον buffer manager**.

## 2. Αναλυτική Περιγραφή
Ο κατάλογος B+-tree που θα κατασκευάσετε αποθηκεύει δεδομένα της μορφής <**key, record_id**>. Το αρχείο στο οποίο θα αποθηκευθεί ο κατάλογος είναι διαφορετικό από το αρχείο δεδομένων. Εκτός από τα αρχεία που είχατε στην πρώτη εργασία δίνονται επιπλέον οι κλάσεις: **`PageFile`**, **`BlobFile`**, και **`FileScan`**.

Οι κλάσεις `PageFile` και `BlobFile` προκύπτουν από την κλάση `File`. Οι κλάσεις αυτές υλοποιούν την πρόσβαση σε ένα αρχείο με διαφορετικούς τρόπους. Η κλάση `PageFile` υλοποιεί στην ουσία τις λειτουργίες της `File` και χρησιμοποιείται για την αποθήκευση των δεδομένων της βάσης. Η κλάση `BlobFile` υλοποιεί ένα interface για την προσπέλαση αρχείων που είναι διαφορετικό από αυτό που υλοποιεί η `PageFile`. Στην `PageFile` υπάρχει η έννοια της προηγούμενης και επόμενης σελίδας δίσκου (υπάρχει η έννοια της συνδεδεμένης λίστας). Αντίθετα, στην `BlobFile` τα περιεχόμενα της κάθε κάθε σελίδας (block) δε χρειάζεται να ανταποκρίνεται σε κάποιες απαιτήσεις. Το κάθε block αποτελείται από 8KB και μπορούμε να αποθηκεύσουμε οτιδήποτε. Ο κατάλογος B+-tree θα αποθηκευθεί σε ένα αρχείο που δημιουργείται από την `BlobFile`. **Κάθε block του αρχείου αντιστοιχεί σε έναν κόμβο του B+-tree**. Η βασική δουλειά γίνεται στο αρχείο **`btree.cpp`** που υλοποιεί τη βασική λειτουργικότητα του καταλόγου B+-tree.

### `Class FileScan`
Η κλάση `FileScan` χρησιμοποιείται για την προσπέλαση εγγραφών σε ένα αρχείο. Η κλάση αυτή χρησιμοποιείται για τα δεδομένα των πινάκων της ΒΔ και όχι για τον κατάλογο. Ο πηγαίος κώδικας που δίνεται στο αρχείο main.cpp περιέχει κώδικα που δείχνει τον τρόπο χρήσης αυτής της κλάσης. Οι public μέθοδοι είναι:

#### `FileScan (const std::string &relationName, BufMgr *bufMgr)`
Ο constructor λαμβάνει το relationName (ποιον πίνακα της ΒΔ θα διαβάσουμε) και δείκτη στον BufMgr.

#### `~FileScan()`
Κλείνει την προσπέλαση και κάνει unpin όλες τις pinned σελίδες.

#### `void scanNext(RecordId& outRid)`
Επιστρέφει στην παράμετρο outRid το record id του επόμενου record από τον πίνακα που διαβάζουμε. Δίνει το `EndOfFileException` όταν φτάσουμε στο τέλος του πίνακα και δεν υπάρχει άλλο record να διαβάσουμε.

**`std::string getRecord()`**
Επιστρέφει το record που αναφέρεται στο record id που προσδιορίστηκε με την προηγούμενη εκτέλεση της `scanNext()`.

**`void markDirty()`**
Μαρκάρει dirty τη σελίδα που διαβάσαμε. Για την εργασία αυτή δε θα χρειαστεί να χρησιμοποιηθεί η συνάρτηση αυτή, απλά υπάρχει για λόγους πληρότητας.

Στόχος σας είναι η κατασκευή μία απλοποιημένης εκδοχής του B+-tree. Υποθέτουμε αρχικά ότι **όλα τα records έχουν το ίδιο μέγεθος** (fixed length), επομένως η τιμή ενός attribute σε κάποιο record θα έχει πάντα το ίδιο offset για όλα τα records. Επίσης υποθέτουμε ότι ο κατάλογος **εφαρμόζεται μόνο σε μία στήλη** του πίνακα (δε θα υποστηρίξουμε multi-attribute καταλόγους). Επίσης, ο τύπος δεδομένων βάσει του οποίου κατασκευάζεται ο κατάλογος μπορεί να είναι integer, double ή string. Σε περίπτωση που ο κατάλογος οργανώνει strings, να υποθέσετε ότι οι συμβολοσειρές αποτελούνται τουλάχιστον από 10 χαρακτήρες και ότι **οι 10 πρώτοι χαρακτήρες είναι unique για κάθε συμβολοσειρά**. Άρα, οι 10 πρώτοι χαρακτήρες μπορούν να χρησιμοποιηθούν ως κλειδί. Επίσης, υποθέστε ότι ποτέ δε θα εισάγουμε το ίδιο κλειδί στον κατάλογο. Ο κατάλογο θα υλοποιηθεί απευθείας πάνω από το I/O επίπεδο (κλάση `BlobFile` και κλάση `Page`).

Ο κατάλογος πρέπει να αποθηκεύει πληροφορίες σε ένα αρχείο δίσκου. Για τη δημιουργία του καταλόγου πρέπει να κληθεί ο constructor της κλάσης `BlobFile` με παράμετρο το όνομα του αρχείου που περιέχει τον κατάλογο. Το αρχείο που δημιουργείται είναι στην ουσία raw file, αποτελείται από blocks χωρίς η `BlobFile` να γνωρίζει τη δομή. Πάνω από την απλή αυτή μορφή των blocks θα πρέπει να υλοποιοηθεί η λογική των blocks του B+-tree. Όπως θα παρατηρήσετε, η κλάση `PageFile` προσφέρει μία συγκεκριμένη δομή πάνω από ένα block. Όπως η κλάση File χρησιμοποιεί το πρώτο block για την αποθήκευση metadata και άλλων βοηθητικών πληροφοριών, έτσι κι εσείς πρέπει να αφιερώσετε **ένα block ως header** για την αποθήκευση των metadata.

**`Class BTreeIndex`**
Ας δούμε την περιγραφή της κλάσης **`BTreeIndex`**. Θα πρέπει να γράψετε κώδικα για να υλοποιηθούν οι συναρτήσεις αυτές. Μπορείτε να προσθέσετε και άλλες συναρτήσεις αν θέλετε στο public τμήμα, αλλά δεν επιτρέπεται να αλλάξετε το interface στις συναρτήσεις που περιγράφονται στη συνέχεια.

**`BTreeIndex`**
Ο constructor ελέγχει αρχικά εάν το αρχείο καταλόγου υπάρχει. Αν το αρχείο υπάρχει τότε ανοίγει. Διαφορετικά δημιουργείται ένα νέο καταλόγου. Η γενική μορφή του ονόματος του αρχείου είναι: **`relName.attrOffset`**. Ο κώδικας για την κατασκευή του ονόματος ενός καταλόγου δίνεται στη συνέχεια:

```
std::ostringstream idxStr ;
idxStr << relationName << '.' << attrByteOffset ;
std::string indexName = idxStr.str( ) ; // indexName είναι το όνομα του αρχείου όπου αποθηκεύεται ο κατάλογος
```

Ο constructor **`BtreeIndex`** δέχεται τις ακόλουθες παραμέτρους:

**`const string &relationName`**: το όνομα του πίνακα πάνω στον οποίο θα κατασκευαστεί ο κατάλογος. Ο constructor θα πρέπει να σκανάρει τον πίνακα (χρησιμοποιώντας την FileScan) και να εισάγει τα records του πίνακα στον κατάλογο ένα-προς-ένα (δε χρειάζεται να ασχοληθείτε με τεχνικές μαζικής κατασκευής τύπου bulk loading).

**`String& outIndexName`**: Το όνομα του αρχείου όπου θα αποθηκευτεί ο κατάλογος (δείτε τον κώδικα παραπάνω).

**`BufMgr *bufMgrIn`**: Pointer σε αντικείμενο `BufMgr` (buffer manager).

**`const int attrByteOffset`**: Το byte offset της στήλης πάνω στην οποία θα κατασκευαστεί ο κατάλογος. Για παράδειγμα, αν τα records έχουν την ακόλουθη μορφή:

```
struct RECORD {
            int i ;
            double d ;
            char s[64] ;
};
```

και θέλουμε να δημιουργήσουμε κατάλογο στο attribute double d, τότε η τιμή του `attrByteOffset` πρέπει να είναι `0 + offsetof(RECORD,i)`, όπου το `offsetof` **παρέχεται από την standard C++ library**. (π.χ. δείτε εδώ http://www.cplusplus.com/reference/cstddef/offsetof/)

**`const Datatype attrType`**: Ο τύπος δεδομένων πάνω στον οποίο θα κατασκευαστεί ο κατάλογος. Το Datatype enumeration περιέχει τις τιμές INTEGER, DOUBLE, STRING και ορίζεται στο **`btree.h`**

**`~BtreeIndex`**
Destructor. Καθαρίζει ότι χρειάζεται και κάνει unpin όποιες σελίδες του καταλόγου είναι pinned και εκτελεί τη λειτουργια `bufMgr→flushFile()`. **Προσοχή, δεν πρέπει να διαγραφεί το αρχείο του καταλόγου**. Ωστόσο, θα πρέπει να γίνει διαγραφή του αντικειμένου κλάσης `File`, με αποτέλεσμα να εκτελεστεί ο destructor της `File` και το αρχείο να γίνει closed.

**`insertEntry`**
Η μέθοδος αυτή εισάγει ένα νέο entry της μορφής <key, recordID> στον κατάλογο. Δέχεται τα ακόλουθα ορίσματα:

**`const void* key`**: Pointer στην τιμή που θέλουμε να εισάγουμε (integer/double/string).
**`const RecordId& rid`**: Το recordID στο οποίο αναφερόμαστε από τον πίνακα.

**`startScan`**
Η μέθοδος χρησιμοποιείταθ για να ξεκινήσει ένα filtered scan στον κατάλογο. Για παράδειγμα, εάν η μέθοδος κληθεί με παραμέτρους `startScan("a",GT,"d",LTE)`, τότε η startScan θα πρέπει να επιστρέψει όλα τα στοιχεία που είναι μεγαλύτερα από (GT) “a” και μικρότερα από ή ίσα με (LTE) “d”.

Παράμετροι της συνάρτησης:

**`const void* lowValue:`** Το αριστερό άκρο του διαστήματος (low value).

**`const Operator lowOp`**: Η λειτουργία που πρέπει να εκτελεστεί στο `lowValue`. Θα πρέπει να υποστηρίξεται μόνο GT (greater than) και GTE (greater than or equal). Οτιδήποτε άλλο θα πρέπει να δίνει `BadOpcodesException`. Επίσης, το enumeration για τους τελεστές βρίσκεται στο `btree.h`

**`const void *highValue`**: Το δεξιό άκρο του διαστήματος (high value).

**`const Operator highOp`**: Η λειτουργία που πρέπει να εκτελεστεί στο high`Value`. Θα πρέπει να υποστηρίξεται μόνο LT (less than) και LTE (less than or equal). Οτιδήποτε άλλο θα πρέπει να δίνει `BadOpcodesException`.

Σε περίπτωση που `lowValue > highValue`, πρέπει να έχουμε `BadScanrangeException`.

**`scanNext`**
Η μέθοδος αυτή επιστρέφει το record id του επόμενου record που ικανοποιεί τα κριτήρια που έχουν προσδιοριστεί στη `startScan`. Έαν έχουμε φτάσει στο τέλος, τότε θα πρέπει να δώσουμε `IndexScanCompletedException`. Για παράδειγμα, εάν σε κάποιο scan υπάρχουν 3 records που ικανοποιούν τα κριτήρια, αν κάνουμε 4 φορές scanNext, την τέταρτη φορά θα πρέπει να έχουμε `IndexScanCompletedException`.

Ένα φύλλο του δένδρου που έχει διαβαστεί και έχει φορτωθεί στον buffer δε θα πρέπει να γίνει unpinned εκτός και αν έχουν διαβαστεί όλα τα records που αναφέρονται από αυτό ή έχουμε φτάσει στο τέλος του scan. Χρησιμοποιήστε το δεξιό γείτονα για να μεταβείτε στο επόμενο φύλλο του δένδρου.

Η μέθοδος παίρνει την παράμετρο:
**`RecordId& outRid`**: Είναι το record id που ικανοποιεί τα κριτήρια που έχουν προσδιοριστεί στη `startScan`.

**`endScan`**
Η μέθοδος αυτή τερματίζει το τρέχον scan και κάνει unpinned όλες τις σελίδες που έχουν γίνει pinned για την εκτέλεση του scan. Δίνει `ScanNotInitializedException` εάν κληθεί πριν από κάποια `startScan`.

**3. Υποδείξεις**
1. Για την υλοποίηση των μεθόδων που απαιτούνται, θα πρέπει να καλέσετε και συναρτήσεις του buffer για read/write στις σελίδες. Φροντίστε ώστε να μην κρατάτε τις σελίδες pinned εκτός και εάν υπάρχει σοβαρός λόγος να το κάνετε.

2. Για την υλοποίηση ενός scan θα πρέπει να θυμάστε την κατάσταση μέχρι να τερματιστεί. Για το λόγο αυτό μπορείτε να χρησιμοποιήσετε member variables στην κλάση `BTreeIndex`. Προσέξτε ώστε οι μεταβλητές που θα χρησιμοποιήσετε να γίνονται reset όταν εκτελείται η endScan ή ο destructor.

3. Ο αλγόριθμος για την εισαγωγή στοιχείου στο B+-tree **να είναι ο απλός**, που κάνει split χωρίς να προσπαθεί να μοιράσει στοιχεία αριστερά ή δεξιά για να αποφύγει το split.

4. Στα φύλλα του B+-tree να έχετε απλή συνδεδεμένη λίστα προς τα δεξιά και όχι διπλά συνδεδεμένη λίστα.

**4. Παραδοτέα και Βαθμολόγηση**

Η εργασία λαμβάνει το **25%** του τελικού βαθμού του μαθήματος. Στη βαθμολόγηση θα ληφθούν υπόψη τόσο η ορθότητα της λύσης όσο και η καθαρότητα και ο σχολιασμός του κώδικα. Η εργασία δε θα βαθμολογηθεί σε περίπτωση που διαπιστωθεί αντιγραφή από άλλη εργασία ή από έτοιμο κώδικα από οποιαδήποτε πηγή. Σημειώνεται ότι σε περίπτωση αντιγραφής υπάρχει ο κίνδυνος μηδενισμού στο μάθημα. Επίσης, δε θα βαθμολογηθούν εργασίες που δεν περνούν τη φάση της μεταγλώττισης τουλάχιστον.

Θα πρέπει να παραδώσετε τον πηγαίο κώδικα όλου του project και ένα report στο οποίο να αναφέρεται πως δουλέψατε. Για να θεωρηθεί η εργασία ολοκληρωμένη θα πρέπει να παραδώσετε το σύνολο του πηγαίου κώδικα μαζί με το **`Makefile`** έτσι ώστε να μπορεί να γίνει η μεταγλώττιση και η σύνδεση με χρήση του εργαλείου **`make`**, χρησιμοποιώντας compiler **`g++`**. Επίσης, θα πρέπει να υπάρχει και μία τεχνική αναφορά (**σε `pdf`**) το πολύ **μέχρι 4 σελίδες** όπου θα περιγράφετε τη λύση σας.

Οι εργασίες θα πρέπει να παραδοθούν στο elearning μέχρι τις 6 Ιουνίου 2018.

**Καλή επιτυχία!**