dreamlegends / DB2

No description, website, or topics provided.

4 commits | 1 branch | 0 releases | 1 contributor | MIT

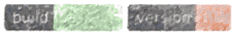
Branch: **master** | **New pull request** | **Find File** | **Clone or download**

dreamlegends update readme fix typo and gramma — Latest commit a1c63c9 7 days ago

src	init add all files	7 days ago
.gitignore	add gitignore	7 days ago
LICENSE	init add all files	7 days ago
LICENSE.md	init add all files	7 days ago
README.md	update readme fix typo and gramma	7 days ago
courses.csv	init add all files	7 days ago
enrollments.csv	init add all files	7 days ago
pom.xml	init add all files	7 days ago
students.csv	init add all files	7 days ago

README.md

DB2



TASKS

In this assignment, you will implement Indexed Nested Loop Join. this file will explain:

- How to setup the code.
- What code you should implement.

Setup and Environment

1. Installation
   i. Setup Java environment, and add java to path. This repo built on JAVA8.
      - Download Official JDK HERE
      - OpendJDK is another choice.
   ii. This project use Apache Maven for package management. Install Maven if you need.

- Installation Guide

2. Code Setup
   - Clone Repo From Github.

     ```
     git clone https://github.com/dreamlegends/DB2.git
     ```

   - Compile and Run Tests.

     ```
     cd DB2
     mvn clean
     mvn compile
     mvn test
     ```

     You will see there are total 152 tests, and you will get 32 errors. You don't need to worry about this, there errors indicate the place you should implement your code.

Getting Familiar with the Code

Navigate to the `src/main/java/` directory. You will find seven directories: `common`, `database`, `databox`, `io`, `table`, `query` and `index`. You do not have to deeply understand all of the code, but since all future programming assignments will reuse this code, it's worth becoming a little familiar with it. **In this assignment, though, you may only modify files in the `index`, `query`, and `test/java/query/` directory.**

common

The `common` directory contains miscellaneous and generally useful bits of code that are not particular to this assignment.

databox

The `databox` directory contains the classes which represent the values stored in a database, as well as their types. Specifically, the `DataBox` class represents values and the `Type` class represents types. Here's an example:

```
```java
DataBox x = new IntDataBox(42); // The integer value '42'.
Type t = Type.intType();        // The type 'int'.
Type xsType = x.type();         // Get x's type: Type.intType()
int y = x.getInt();             // Get x's value: 42
String s = x.getString();       // An exception is thrown.
```
```

io

The `io` directory contains code that allows you to allocate, read, and write pages to and from a file. All modifications to the pages of the file are persisted to the file. The two main classes of this directory are `PageAllocator` which can be used to allocate pages in a file, and `Page` which represents pages in the file. Here's an example of how to persist data into a file using a `PageAllocator`:

```
// Create a page allocator which stores data in the file "foo.data". Setting
// wipe to true clears out any data that may have previously been in the file.
bool wipe = true;
PageAllocator allocator = new PageAllocator("foo.data", wipe);

// Allocate a page in the file. All pages are assigned a unique page number
// which can be used to fetch the page.
int pageNum = allocator.allocPage(); // The page number of the allocated page.
Page page = allocator.fetchPage(pageNum); // The page we just allocated.
System.out.println(pageNum); // 0. Page numbers are assigned 0, 1, 2, ...
```

1. Read through all of the code in the `index` directory. Many comments contain critical information on how you must implement certain functions. For example, `BPlusNode::put` specifies how to redistribute entries after a split. You are responsible for reading these comments. If you do not obey the comments, you will lose points. Here are a few of the most notable points:
    - Our implementation of B+ trees does not support duplicate keys. You will throw an exception whenever a duplicate key is inserted.
    - Our implementation of B+ trees assumes that inner nodes and leaf nodes can be serialized on a single page. You do not have to support nodes that span multiple pages.
    - Our implementation of delete does not rebalance the tree. Thus, the invariant that all non-root leaf nodes in a B+ tree of order `d` contain between `d` and `2d` entries is broken. Note that actual B+ trees **do rebalance** after deletion, but we will **not** be implementing rebalancing trees in this project for the sake of simplicity.
2. Implement the `get`, `getLeftmostLeaf`, `put`, and `remove` methods of `InnerNode` and `LeafNode`. For information on what these methods do, refer to the comments in `BPlusNode`. Don't forget to call `sync` when implementing `put`, and `remove`; it's easy to forget.
3. Implement the `get`, `scanAll`, `scanGreaterEqual`, `put`, and `remove` methods of `BPlusTree`. In order to implement `scanAll` and `scanGreaterEqual`.

After this, you should pass tests in the index folder we have provided to you.

2. Index nested loop Join

The `query` directory contains a full implementation of nested loop join, and a partial implemented Index nested loop join.

In this assignment, do the following:

1. Read through the code of `QueryOperator`, `SNLJOperator`. They contain critical information on how to Implement INLJ.
2. Read through the code of `SequentialScanOperator` and implement `BtreeScanOperator`。
3. Then Implement the `iterator` function in INLJ, you will need to scan the btree here, so finish the `BtreeScanOperator` before INLJ.

3. Tests Two table join and three table join.

Test your Index Nested Loop Join. We provide three CSV files in the root directory, courses.csv, students.csv, and enrollments.csv.

```
courses.csv

cid, cname, dept
1,CS 186,Computer Science


students.csv

sid, sname, major, gpa
1,Augustina Mazzoni,Chemistry,1.005420210172708


enrollments.csv

sid, cid
2,11
```

In this assignment, do the following:

1. Load data from each CSV file, create table and build build btree. Implement `loadEnrollment`, `loadCourse`, and `loadStudent` in `test\java\query\TestINLJ.java`

```
// Write data into the page. All data written to the page is persisted in the
// file automatically.
Buffer buf = page.getBuffer(transaction);
buf.putInt(42);
buf.putInt(9001);
```

And here's an example of how to read data that's been persisted to a file:

```
// Create a page allocator which stores data in the file "foo.data". Setting
// wipe to false means that this page allocator can read any data that was
// previously stored in "foo.data".
bool wipe = false;
PageAllocator allocator = new PageAllocator("foo.data", wipe);

// Fetch the page we previously allocated.
Page page = allocator.fetchPage(0);

// Read the data we previously wrote.
Buffer buf = page.getBuffer(transaction);
int x = buf.getInt(); // 42
int y = buf.getInt(); // 9001
```

table

In future assignments, the `table` directory will contain an implementation of relational tables that store values of type `DataBox`. For now, it only contains a `RecordId` class which uniquely identifies a record on a page by its page number and entry number.

```
// The jth record on the ith page.
RecordId rid = new RecordId(i, (short) j);
```

query

The `query` directory contains what are called query operators. These are operators that are applied to one or more tables, or other operators. They carry out their operation on their input operator(s) and return iterators over records that are the result of applying that specific operator. We call them "operators" here to distinguish them from the Java iterators you will be implementing.

`JoinOperator` is the base class that join operators you will implement extend. It contains any methods you might need to deal with tables through the current running transaction. This means you should not deal directly with `Table` objects in the `Query` directory, but only through methods given through the current transaction.

index

The `index` directory contains a partial implementation of a B+ tree (`BPlusTree`), an implementation that you will complete in this assignment. Every B+ tree maps keys of type `DataBox` to values of type `RecordId`. A B+ tree is composed of inner nodes (`InnerNode`) and leaf nodes (`LeafNode`). Every B+ tree is persisted to a file, and every inner node and leaf node is stored on its own page.

Implementation

1. B+ Tree

The `index` directory contains a partial implementation of a B+ tree (`BPlusTree`), an implementation that you will complete in this assignment. Every B+ tree maps keys of type `DataBox` to values of type `RecordId`. A B+ tree is composed of inner nodes (`InnerNode`) and leaf nodes (`LeafNode`). Every B+ tree is persisted to a file, and every inner node and leaf node is stored on its own page.

In this assignment, do the following:

2. Join Student and enrollment on students.sid = enrollment.sid.

   Implement `testINLJ_SJoinE` function. The function is partially implemented. The comments in the function contains the steps, it works as a guideline. Finish the function and print both schema and result to console. The result should contains 1000 tuples.

3. Join Student, enrollment on students.sid = enrollment.sid, then join Course on course.cid = enrollment.cid.

   Implement `testINLJ_SJoinEJoinC` function. The function is partially implemented. The comments in the function contain the steps, it works as a guideline. Finish the function and print both schema and result to console. The result should contains 1000 tuples. You can reuse your code in question 2.

database.TestCSVF.h2 database/f

testINLJStudentEnrollment
testCSVFile2Btree
testINLJStudentEnrollmentCourses

que.j.TestINLJ loadStudent

TestINLJ_SJoinE
testINLJ_SJoinEJoinC

index. LeafNode. put main → index / LeafNode. java test → TestLeafNode. java

index. TestLeafNode

- testScanGreaterEqual
- testNoOverflowPuts
- TPuts FromDisk
- testOverflowPuts
- testSimpleRemoves
- testToAndFromBytes
- testScanAll
- testToString
- testAbsentRemoves
- testDuplicatePuts
- testNoOverflowOutOfOrderPuts
- testOutOfOrderRemoves

index. TestInnerNode. get main → index / InnerNode. java

- testGet
- testGetLeftMostLeaf
- testNoOverflowPuts
- testOverflowPuts
- testRemove

index TestBPlusNode
TestBPlusTree main → index / BPlusTree. java

- testSimpleGets
- testBPlusTreeFromDisk
- testEmptyScans
- testEmptyTree
- testRandomOps
- testWhiteBoxTest
- testPartiallyEmptyScans
- testRepeatedInsertsAndRemoves
- testDuplicatePut