Lab1 Correction

- Some link errors was in the Lab1 web page
  - 6.830-lab1.tar.gz is an obsolete version
  - Correct version: **CSE444-lab1.tar.gz**

  - **Sorry for the inconveniences**

Overview of the Homeworks

Outline

1. Some rules
2. Setup in Eclipse
3. Grading
4. JUnit
5. SimpleDB Overview

What you should NOT do

- Modifications of the given class names
  - Removal
  - Rename
  - Relocate to other packages
- Modifications of the given method names
  - Removal
  - Rename
  - Changes to accepted parameters
  - Changes to the return types
- Subsequent labs will rely on the classes/methods of foregoing ones

What you should NOT do cont’d

- Using any other third party libraries except the ones under lib
  - JUnit, for unit test
  - JLine, for command line operations
  - Zql, for parsing SQL
  - JZlib, for data compression
  - Mina-core, for parallelism
  - Mina-filter-compression, for parallelism
  - Slf4j-api, for parallelism
- We will not use any other libraries in GRADING

What you are FREE to do

- Adding new classes / interfaces / methods
  - But, if the class/interface names happen to conflict with names we will provide in later labs, please kindly rename them
  - Safer choice: Inner classes

- Adding new packages.
  - Very safe. Do it if you like

What you are ENCOURAGED to do

- Re-implement the given methods
  - Gosh! How can the implementations be so ugly!
  - Welcome to come up with better implementations!
- Find bugs
  - SimpleDB is still in developing, help us improve it!
  - Candy bars

Outline

1. Some rules
2. Setup in Eclipse
3. Grading
4. JUnit
5. SimpleDB Overview

Eclipse Setup

- Demo

Outline

1. Some rules
2. Setup in Eclipse
3. Grading
4. JUnit
5. SimpleDB Overview

Grading

- System test cases
  - Under test/systemtest
  - Mostly those we have released
  - Maybe one or two extra test cases we do not release

- Write up
  - Explain why do you implement in that way

- We’ll read you code
  - Passing all the test cases may not necessary mean you’ll get a high score

Outline

1. Some rules
2. Setup in Eclipse
3. Grading
4. JUnit
5. SimpleDB Overview

JUnit

- A unit testing framework for java
  - Help you organize test cases
- Use java annotations to control how the test cases should run
  - @Test, the method is a test case
  - @Before, this method should run each time before a @Test method runs
  - @After
  - @BeforeClass, this method should run once, before the @Test methods in the class run
  - @AfterClass
- Use assert to check conditions
  - Any condition fails, thet test will fail

JUnit

Example: CatlogTest

JUnit

- Actually, what you only need to bear in mind are:
  - **ant test**
  - **ant systemtest**

- If the bottom of the output likes this:

```
BUILD FAILED
/CSE444-lab1/build.xml:159: The following error occurred
while executing this line:
/CSE444-lab1/build.xml:59: Test
simpledb.systemtest.ScanTest failed

Total time: x second
```

- Something goes wrong in the failed test case

JUnit

- If the bottom of the output likes this:

```
BUILD SUCCESSFUL
Total time: x seconds
```

- Congratulations!
- With very high probability, your implementation should be correct.

JUnit

- Demo

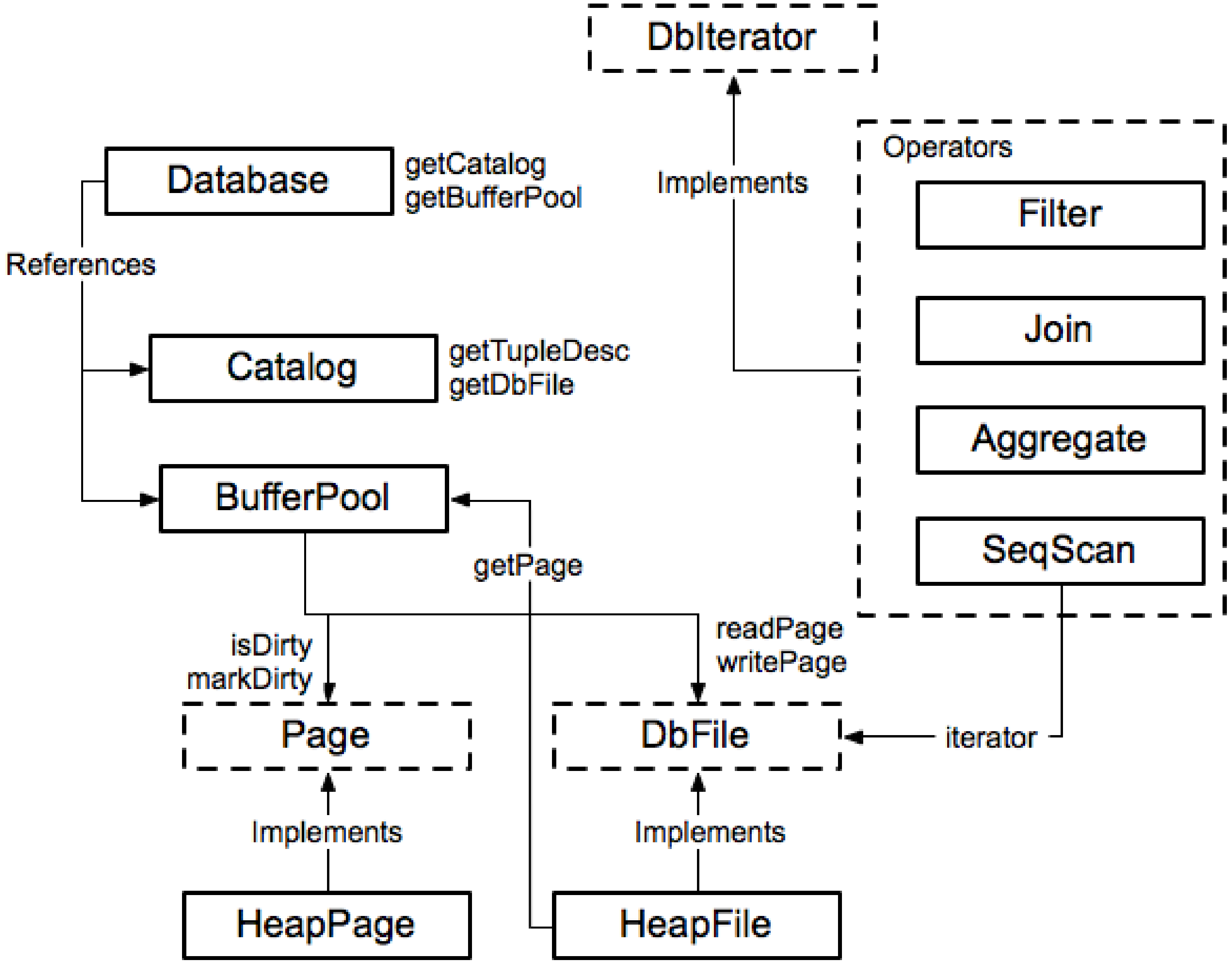
Outline

1. Some rules
2. Setup in Eclipse
3. Grading
4. JUnit
5. SimpleDB Overview

What is SimpleDB?

- A basic database system
- What it has
  - Heapfiles
  - Basic Operators (Scan, Filter, JOIN, Aggregate)
  - Buffer Pool
  - Transactions
  - SQL Front-end
  - Simple Parallelism
- Things it doesn't have
  - Fancy Query optimizer
  - Fancy relational operators (UNION, etc)
  - Subquery
  - Recovery
  - Indices

Module Diagram



Database

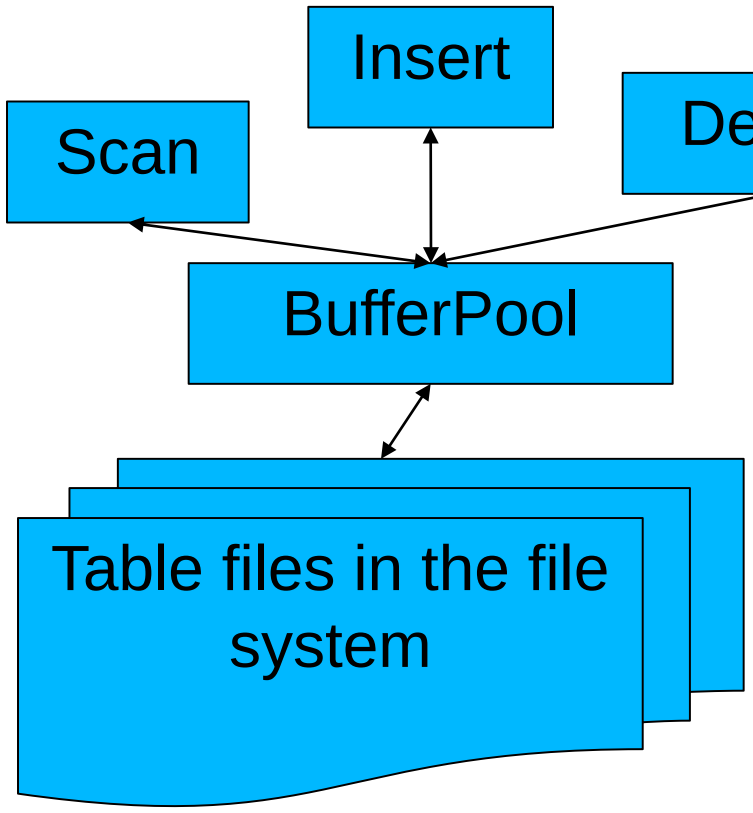
- A single database
  - One schema
  - A bunch of tables
- Stores references to important components:
  - A globally single instance of Catalog
  - A globally single instance of Bufferpool

Catalog

- Catalog stores the meta information of the tables in the current database
  - void addTable(DbFile d, TupleDesc d)
  - DbFile getTable(int tableid)
  - TupleDesc getTupleDesc(int tableid)
  - getPrimaryKey(tableid)
  - …
- Not persisted to disk
  - All the information managed by Catalog should be reloaded every time simpleDB starts

BufferPool

- The ONLY bridge between the data processing operators and the data files



- NEVER directly access data files

Data types

- Integer
  - Type.INT_TYPE
  - 4 bytes long
- Fixed-length String
  - Type.STRING_TYPE
  - 128 bytes long = Type.STRING_LEN
  - Do not change this constant

DbIterator

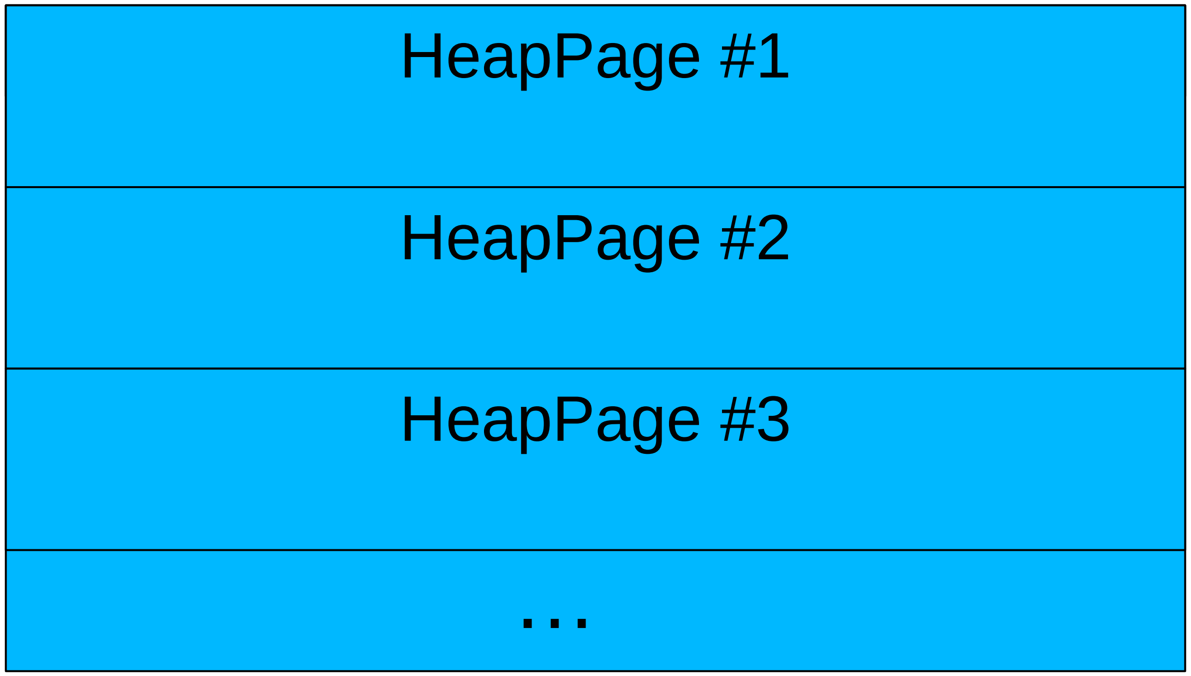
- The ancestor class for all the operators
  - open()
  - close()
  - getTupleDesc()
  - hasNext()
  - next()
  - rewind()

- Iterator model: chain iterators together

```
// construct a 3-column table schema
Type types[] = new Type[]{ Type.INT_TYPE, Type.INT_TYPE, Type.INT_TYPE };
String names[] = new String[]{ "field0", "field1", "field2" };
TupleDesc descriptor = new TupleDesc(types, names);

// create the table, associate it with some_data_file.dat
// and tell the catalog about the schema of this table.
HeapFile table1 = new HeapFile(new File("some_data_file.dat"), descriptor);
Database.getCatalog().addTable(table1);

// construct the query: we use a simple SeqScan, which spoonfeeds
// tuples via its iterator.
TransactionId tid = new TransactionId();
SeqScan f = new SeqScan(tid, table1.id());

// and run it
f.open();
while (f.hasNext()) {
        Tuple tup = f.next();
        System.out.println(tup);
}
f.close();
Database.getBufferPool().transactionComplete();
```

HeapFile

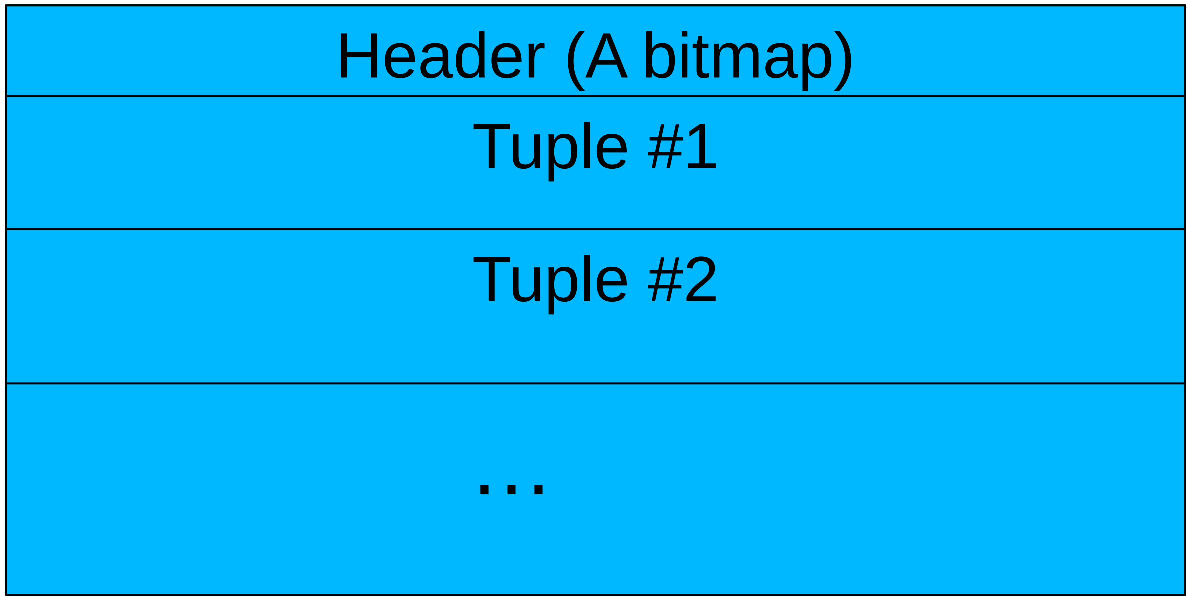
- The main class that organize the physical storages of the tables
  - One heap file for each table
- An array of HeapPages on disk



- Heap pages are of the same fixed size: BufferPool.PAGE_SIZE
  - Efficiently locate any page

HeapPage

- Format
  - Header is a bitmap
  - Page contents are an array of fixed-length Tuples



HeapPage, cont’d

- Full page size = BufferPool.PAGE_SIZE
  - Fixed, Do not change BufferPool.PAGE_SIZE !
- Number of bits in Header = number of Tuples
- PAGE_SIZE-1-size of a tuple < Header size + size of tuples <= PAGE_SIZE

HeapFileEncoder

- Because you haven’t implemented insertTuple, you have no way to create data files
- HeapFileEncoder converts CSV files to HeapFiles
- Usage:
  - java -jar dist/simpledb.jar convert csv-file.txt numFields fieldTypes fieldSeparator
- Produces a file csv-file.dat, that can be passed to HeapFile constructor.

Java Docs

- Java Docs are your friends
- Always follow the guidance of the java docs

Questions