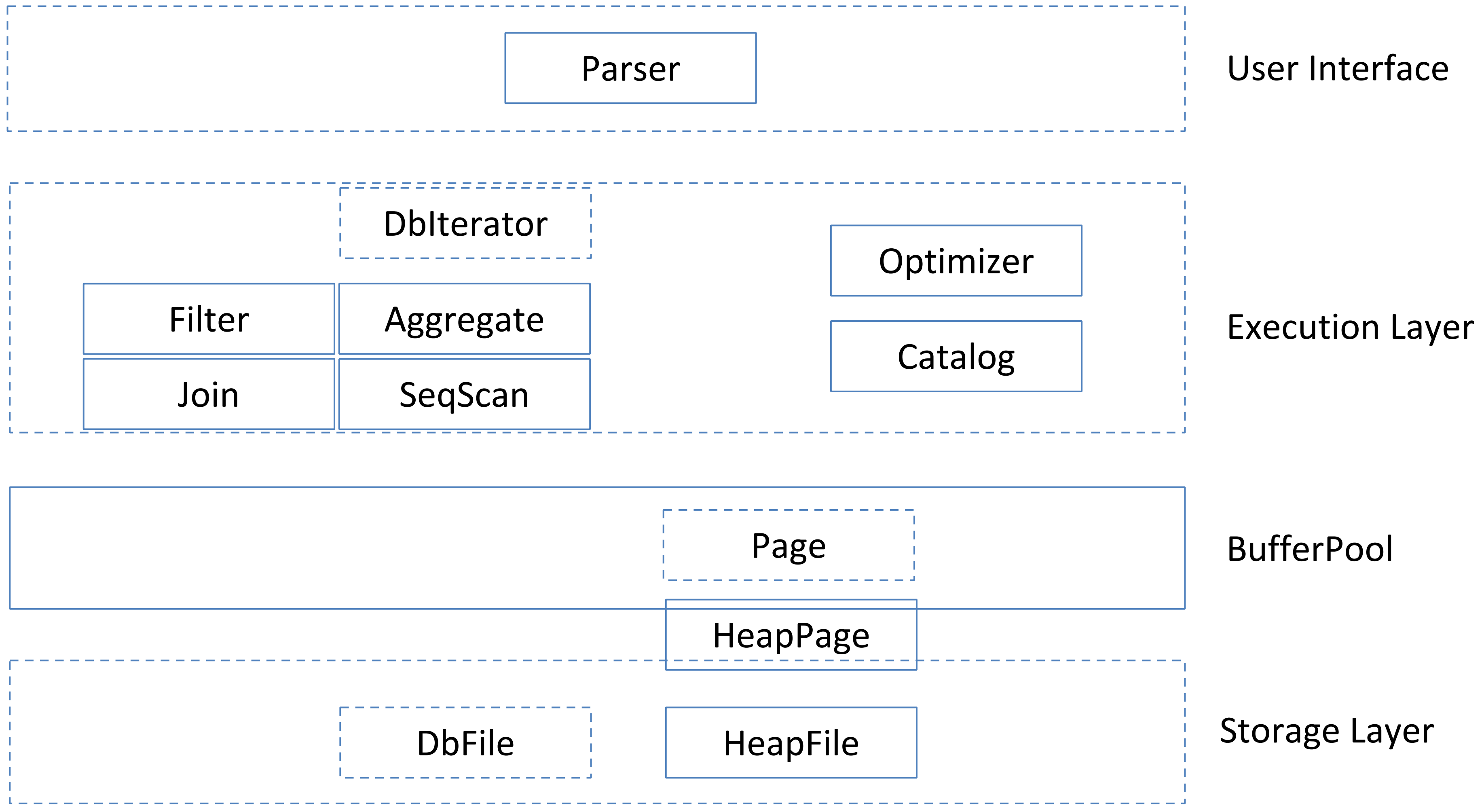
CSE 444 Section 1

Outline

What Is SimpleDB

- A “simple” database system
- It has
  - SQL Front-end
  - Basic Operators (Scan, Filter, Join, Aggregate)
  - Buffer Pool
  - Heap Files
  - Transactions
  - Simple parallelism
  - Simple recovery
  - Simple query optimizer
- It doesn’t have
  - Fancy relational operators (Union, etc)
  - Subquery
  - Indices
  - …

Module Diagram



Labs

- - Lab 1: Heap files + bufferpool
- - Lab 2: operators + updates
- - Lab 3: transactions concurrency
- - Lab 5: transactions recovery
- - Lab 4: query optimizer
- - Lab 6: simple parallelism

Database

- A single database
  - A single tablespace
    - Distinct table names
- Stores references to:
  - A global single instance of Catalog
  - A global single instance of BufferPool

Catalog

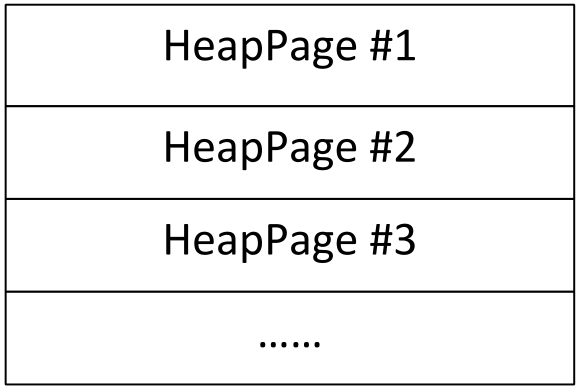
- Manages meta information of the tables in the current database
  - void addTable(DbFile d, TupleDesc d)
  - DbFile getTable(int tableid)
  - TupleDesc getTupleDesc(int tableid)
  - getPrimaryKey(tableid)
  - …
- Not persistent, needs to be reconstructed every time SimpleDB starts

BufferPool

- The ONLY bridge between operators and the data files on disk
- NEVER directly access data files

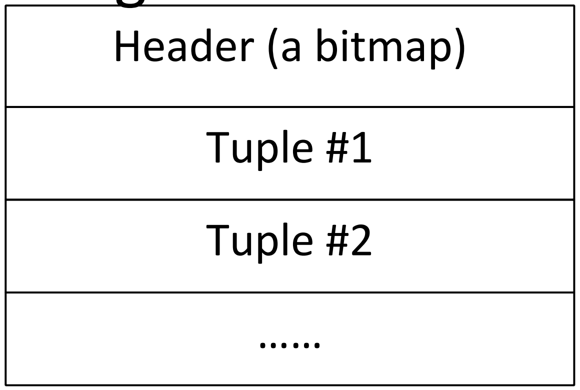
HeapFile

- A file format
- Organizes the physical storage of tables
  - One heap file for each table
- An array of HeapPages
- Heap pages have the same fixed size: BufferPool.PAGE_SIZE
  - To efficiently locate any page



HeapPage

- Header is a bitmap
  - Indicates empty slots
  - Number of bits in Header = number of Tuples
- Following is an array of fixed-length Tuples
- Full page size = BufferPool.PAGE_SIZE
  - Fixed, Do not change!



DbIterator

- An interface that all the operators need to implement
  - open()
  - close()
  - hasNext()
  - next()
  - getTupleDesc()

HeapFileEncoder

- HeapFile has its own format
- Converts CSV files to HeapFiles
- Produces a Heap File csv-file.dat, that can be passed to the HeapFile constructor

- Usage:
  – java -jar dist/simpledb.jar convert csv-file.txt numFields fieldTypes fieldSeparator

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

```java
// 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();
```

Javadoc

- Javadoc is your friend
- Always follow the guidance of the Javadoc

Outline

- 1. SimpleDB Overview
- 2. Use Eclipse
- 3. JUnit
- 4. Grading
- 5. Tips

Use Eclipse

- build.xml
- ant eclipse
  - .classpath
  - .project
- Open Eclipse
- File -> Import -> Existing Projects into Workspace -> select the directory -> done

Outline

- 1. SimpleDB Overview
- 2. Setup in Eclipse
- 3. JUnit
- 4. Grading
- 5. Tips

JUnit

- If you are lazy
  - ant test
  - ant systemtest
- If the bottom of the output likes:

  BUILD FAILED

  The following error occurred while executing this line:

  Test simpledb.systemtest.ScanTest failed
- Something goes wrong in the failed test case
- If the bottom of the output likes :

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

JUnit

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

Outline

- 1. SimpleDB Overview
- 2. Setup in Eclipse
- 3. JUnit
- 4. Grading
- 5. Tips

Grading

- Test cases
  - test/systemtest
  - Some extra test cases that we do not release
  - Each test case:
    - Run multiple times if concurrency is involved
    - All-or-nothing / average?

- Write up
  - Explain why you implement in that way

- We will read you code
  - Passing all test cases doesn’t equal to a high score

Outline

- 1. SimpleDB Overview
- 2. Setup in Eclipse
- 3. JUnit
- 4. Grading
- 5. Tips

Don’t

- Modifications of the given class names
  - Removal, rename, relocate to other packages
- Modifications of the given method names
  - Removal, rename, change parameters, change return types
- Using any other third-party libraries except provided ones
  - 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

Feel Free to

- 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

And you are encouraged to

- Find bugs
  - SimpleDB is still under developing, help us improve it!
  - Candy bars
    - A lot of them were sent out last year
- Re-implement the given methods
  - Gosh! How can the implementations be so ugly!
  - Welcome to come up with better implementations
  - Be aware of your time management
    - (you will not get bonus point)

Questions