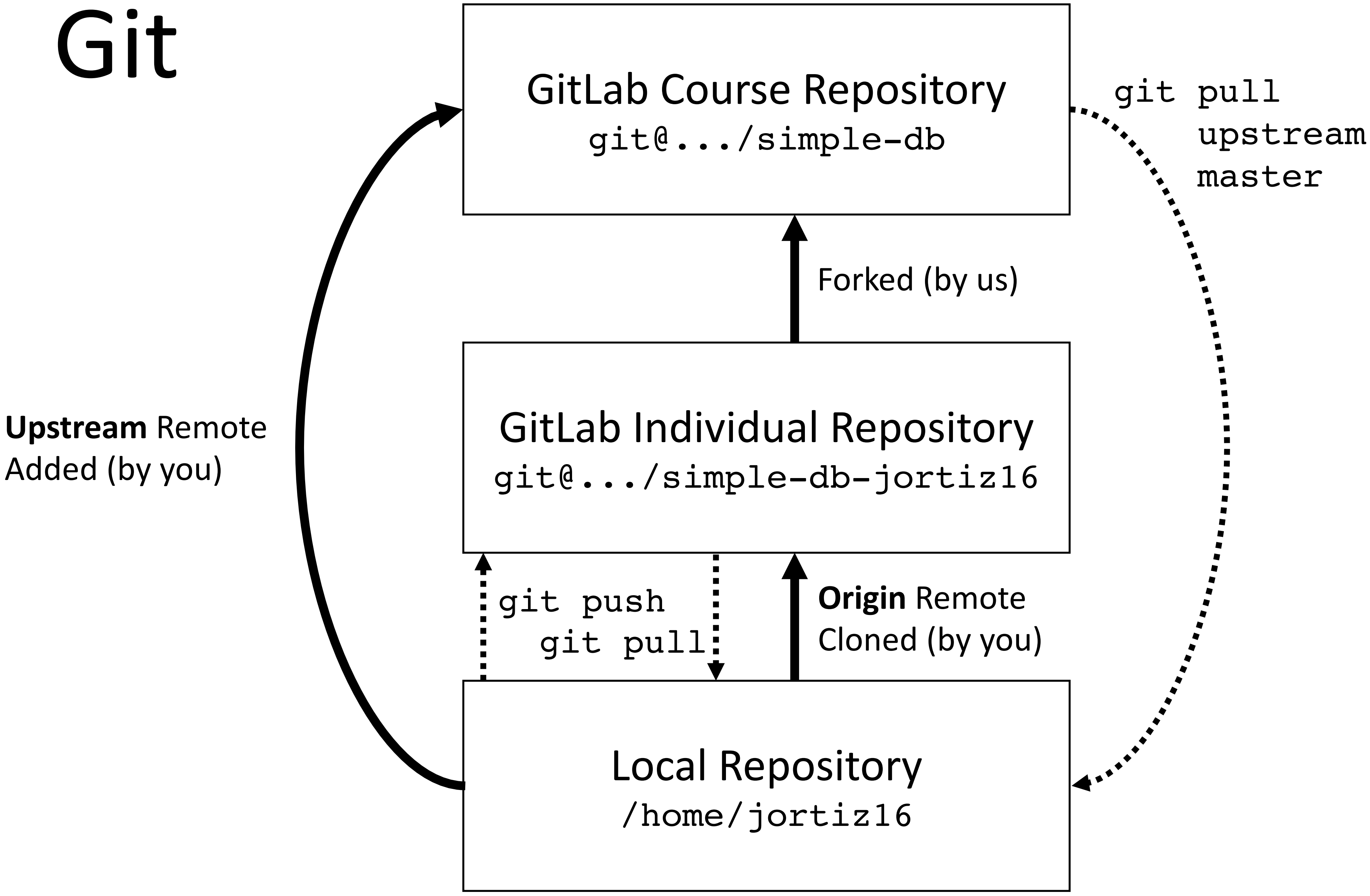
SimpleDB Overview

CSE 444 – Section 1

Today…

- Demo Git/Eclipse Setup
- Go through an overview of SimpleDB

Git


GitLab Course Repository
git@.../simple-db
git pull upstream master
Forked (by us)
Upstream Remote Added (by you)
GitLab Individual Repository
git@.../simple-db-jortiz16
git push
git pull
Origin Remote Cloned (by you)
Local Repository
/home/jortiz16

What you should NOT do:

- **Modify given classes**
    - Removing, renaming, relocating to other packages
- **Modify given methods**
    - Changing parameters or return types
- **Use third-party libraries**
    - Except the ones under lib/directory
    - You can do everything using regular Java libraries

What you CAN do:

- **Add new classes/interfaces/methods/packages**
  - Watch out for name conflicts with future labs!
  - Safer choice: use new packages (best) or inner classes (meh)
- **Re-implement provided methods**
  - Just don’t destroy correctness or specification!
- **Find bugs!**

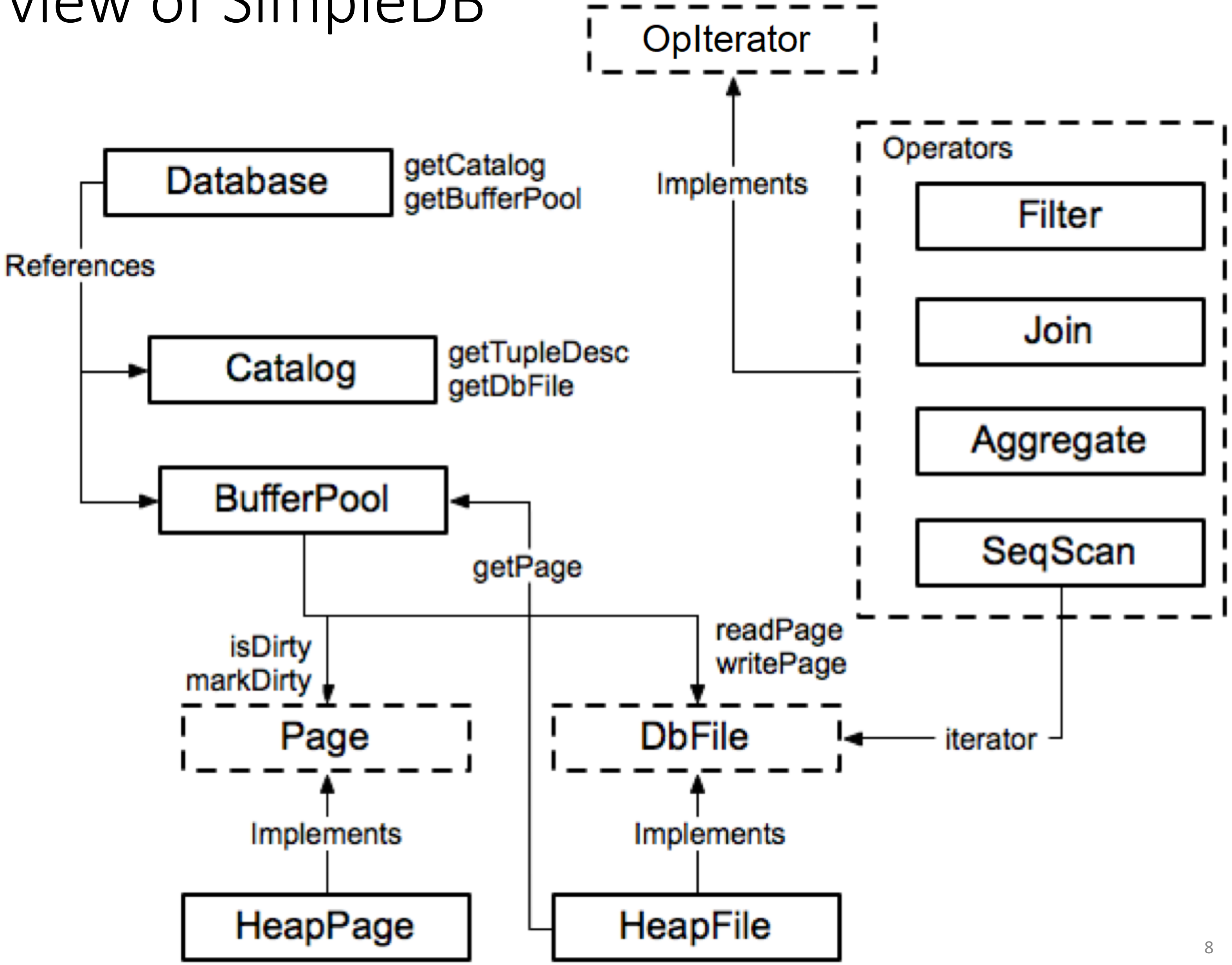
What you CAN do (continued):

- **System test cases**
  - Under test/systemtest
  - We’ll grade using additional tests
- **Write up**
  - Explain why do you implement in that way
- **We’ll read your code**
  - Reading horrible code is horrible, so spend some time polishing
  - Passing all the test cases may not necessary mean you’ll get a high score

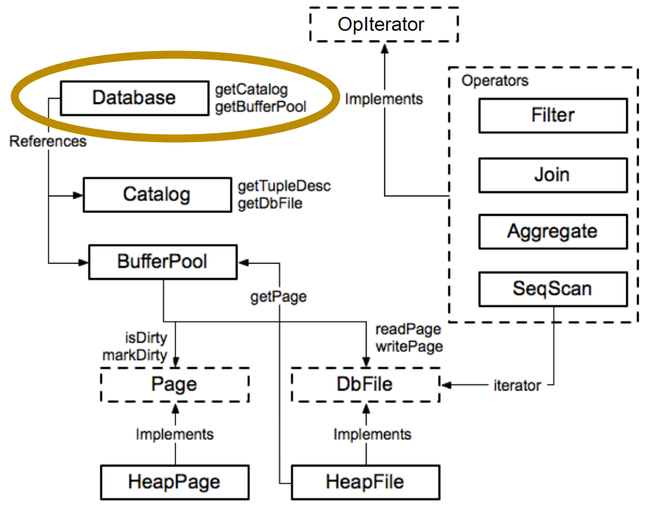
Setting up SimpleDB

Any questions or concerns?

Overview of SimpleDB

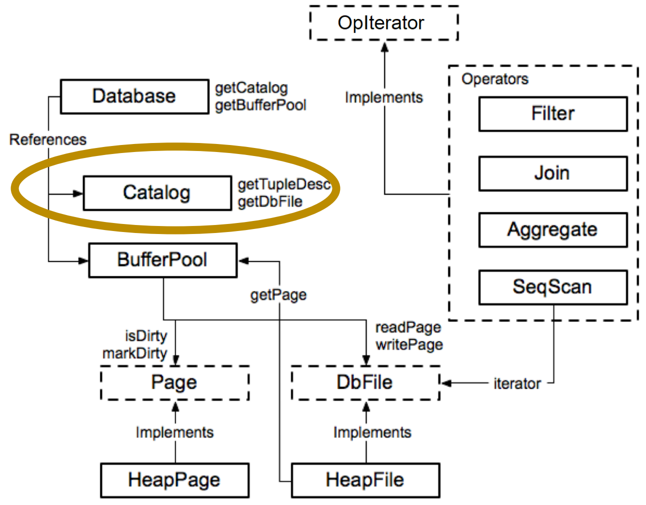


Database



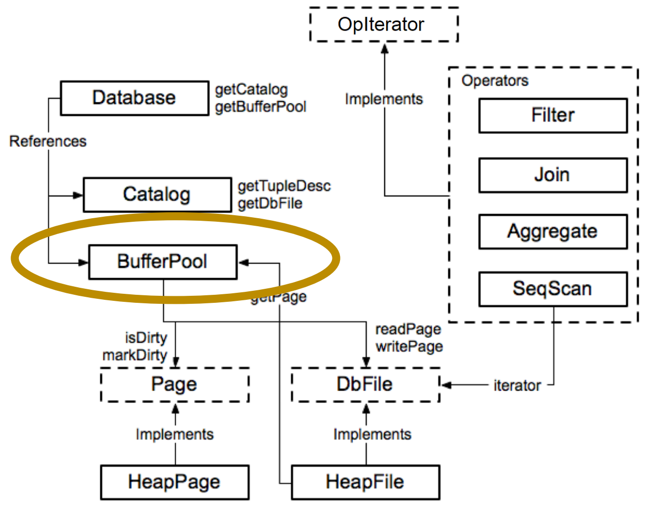
- A single database
  - One schema
  - List of tables
- References to major components
  - Global instance of Catalog
  - Global instance of BufferPool

Catalog

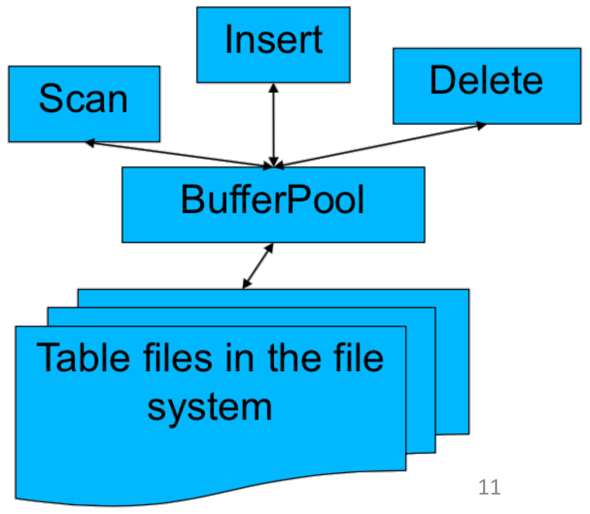


- Stores metadata about tables in the database
  - void addTable(DbFile d, TupleDesc d)
  - DbFile getTable(int tableid)
  - TupleDesc getTupleDesc(int tableid)
  - …
- NOT persisted to disk
  - Catalog info is reloaded every time SimpleDB starts up

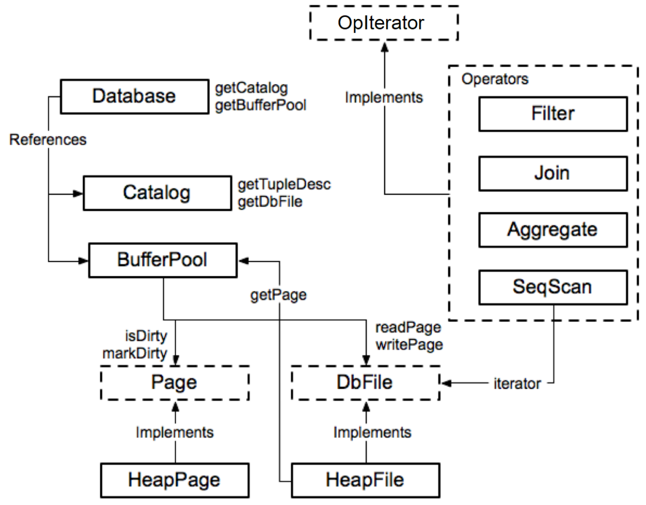
BufferPool



- The ONLY bridge between data-processing operators and actual data files
  - Strict interface for physical independence!
- Data files are never accessed directly
- Later labs:
  - Locking for transactions
  - Flushing pages for recovery

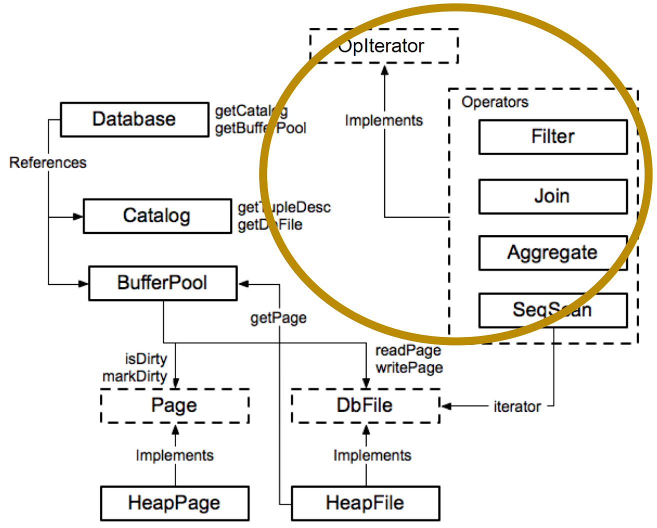


Data Types



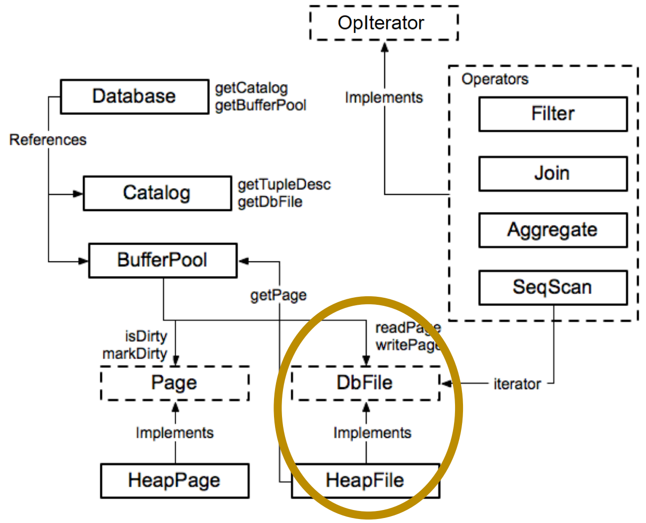
- Integer
  - Type.INT_TYPE
  - 4 byte width
- Fixed-length Strings
  - Type.STRING_TYPE
  - 128 bytes long (Type.STRING_LEN)
  - Do not change this constant!

OpIterator

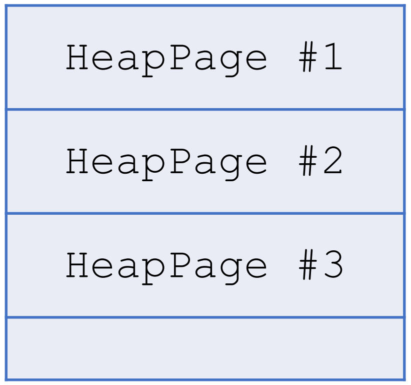


- Ancestor class for all operators
  - Join, Project, SeqScan, etc…
- Each operator has methods:
  - open(), close(), getTupleDesc(), hasNext(), next(), rewind()
- Iterator model: chain iterators together

HeapFile

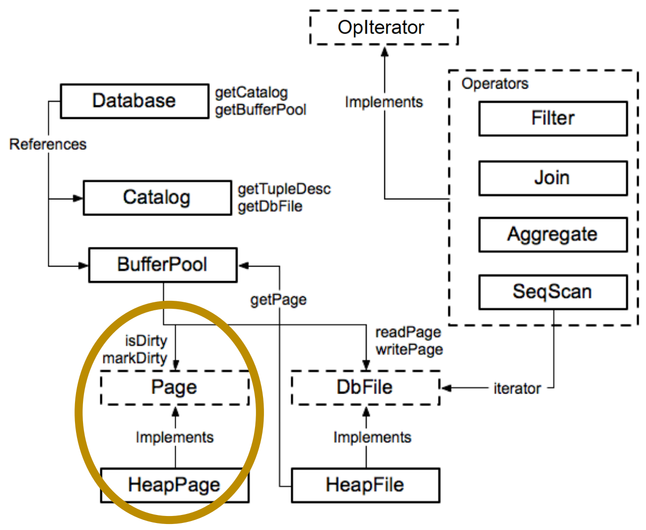


- Main class that organizes the physical storage of tables
- Collection of HeapPages on disk
  - One HeapFile for each table
  - Fixed-size pages means efficient lookup of pages

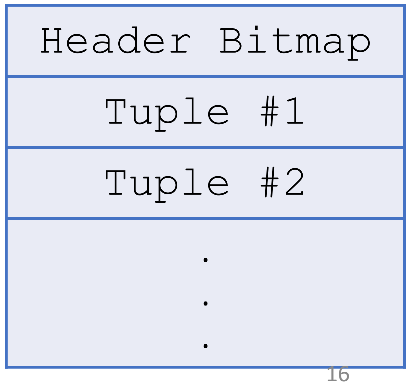


```
// construct a 3-column table schema
Type types[] = new Type[]{ Type.INT_TYPE, Type.INT_TYPE, Type.INT_TYPE };
String names[] = new String[]{ "field0", "field1", "field2" };
TupleDesc descriptor = new TupleDesc(types, names);

// 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();
```

HeapPage



- A chunk of data that can reside in the BufferPool
- Format: Header + Tuples
  - # of 1 bits in Bitmap = # of active tuples on page
- Fixed size: BufferPool.PAGE_SIZE



Questions?