the methods implemented:

public Tuple getTuple(int entryID)

{

int x = tupleLocation( entryID );

//int y = tupleLocation(entryCount());;

//int y = sizeOfHeader();

//ByteArrayInputStream byteArray = new ByteArrayInputStream(data, x, BufferPool.PAGE\_SIZE/y );

ByteArrayInputStream byteArray = new ByteArrayInputStream(data, x, data.length - x );

DataInputStream in = new DataInputStream(byteArray);

Tuple newTuple = new Tuple(createTuple(in).getTupleDesc());

newTuple.setRecordId(new RecordId(pid, entryID));

return newTuple;

}

public Iterator<Tuple> iterator()

{

//List<Tuple> tupleList = new ArrayList<Tuple>();

return new Iterator<Tuple>()

{

//stopping point for when it doesnt exist

private int lastPtr = -1;

private int ptr = 0;

@Override

public boolean hasNext()

{

// for(int i=0; i<tuples.length; i++)

// if(getTuple(entryID).getTuple(entryID))

// {

// tupleList.add(tuples[i]);

// }

for (;this.ptr < HeapPage.this.numSlot; this.ptr++)

{

if (HeapPage.this.tuples[this.ptr] != null)

{

return true;

}

}

return false;

}

@Override

public Tuple next()

{

this.lastPtr = this.ptr;

for (;this.ptr < HeapPage.this.numSlot; this.ptr++)

{

if (HeapPage.this.tuples[this.ptr] != null)

{

return HeapPage.this.tuples[this.ptr++];

//return tupleList.iterator();

}

}

return null;

}

@Override

public void remove()

{

if (this.lastPtr >= 0)

{

HeapPage.this.tuples[this.lastPtr] = null;

}

}

};

}

public Page readPage(PageId pid)

{

try

{

RandomAccessFile randomFile = new RandomAccessFile(this.file, "r");

int x = pid.pageno() \* BufferPool.PAGE\_SIZE;

randomFile.seek(x);

byte[] page = new byte[BufferPool.PAGE\_SIZE];

randomFile.read(page, 0, BufferPool.PAGE\_SIZE);

randomFile.close();

HeapPageId id = (HeapPageId) pid;

return new HeapPage(id, page);

} catch (IOException e)

{

e.printStackTrace();

}

throw new IllegalArgumentException();

}

public DbFileIterator iterator(TransactionId tid) throws DbException, TransactionAbortedException {

List<Tuple> ftupleList = new ArrayList<Tuple>();

int page\_count = this.numPages();

for (int i = 0; i < page\_count; i++) {

PageId pid = new HeapPageId(getId(), i);

// using bufferpool class get page method

HeapPage page = (HeapPage) Database.getBufferPool().getPage(tid, pid, Permissions.READ\_ONLY);

Iterator<Tuple> it = page.iterator();

while (it.hasNext())

ftupleList.add(it.next());

}

return new HeapIterator(tid, ftupleList);

}

public Page getPage(TransactionId tid, PageId pid, Permissions perm)

throws TransactionAbortedException, DbException

{

if (pages.containsKey(pid))

{

return pages.get(pid);

}

else

{

if (pages.size() >= numPages)

{

evictPage();

}

DbFile dbfile = Database.getCatalog().getDbFile(pid.getTableId());

Page newPage = dbfile.readPage(pid);

pages.put(pid, newPage);

return newPage;

}

// return null;

}

public DbFile getDbFile(int tableid) throws NoSuchElementException

{

// some code goes here

Table match = this.IdHash.get(tableid);

if (match == null)

{

throw new NoSuchElementException();

}

else

{

return match.getDbFile();

}

}

public TupleDesc getTupleDesc(int tableid) throws NoSuchElementException

{

// some code goes here

Table match = this.IdHash.get(tableid);

if (match == null)

{

throw new NoSuchElementException();

}

else

{

DbFile file = match.getDbFile();

return file.getTupleDesc();

}

}

public int getTableId(String name) throws NoSuchElementException

{

// some code goes here

Table match = this.NameHash.get(name);

//if(this.name2tableID.get(i).getName().equals(name))

if (match == null)

{

throw new NoSuchElementException();

}

else

{

DbFile file = match.getDbFile();

//return this.name2tableID.get(i).getId();

return file.getId();

}

}

**public** String toString()

{

String str\_Des = **new** String();

//for(int k=0 ; k<this.num\_Fields ; k++){

**for**(**int** k=0 ; k< types.length ; k++)

{

//System.out.println(this.types[k]);

str\_Des = str\_Des + **this**.types[k];

//System.out.println(this.types[k]);

str\_Des = str\_Des + "(";

str\_Des = str\_Des + **this**.names[k];

str\_Des = str\_Des + ")";

System.***out***.println(**this**.types[k]);

System.***out***.println(str\_Des);

**if**(k< types.length-1){

str\_Des = str\_Des + "|";

}

}

System.***out***.println(str\_Des);

**return** str\_Des;

//throw new UnsupportedOperationException("Implement this");

}

}

public Type getType(int i) throws NoSuchElementException {

if( i >= names.length ){

throw new UnsupportedOperationException("return index out of range!");

}else{

return this.types[i];

}

}

public String toString()

{

String fields\_Des = new String();

//for(int k=0 ; k<this.num\_Fields ; k++){

for(int k=0 ; k< fields.length ; k++)

{

fields\_Des = fields\_Des + this.fields[k];

System.out.println(this.fields[k]);

System.out.println(fields\_Des);

if(k< fields.length-1)

{

fields\_Des = fields\_Des + "|";

}

}

System.out.println(fields\_Des);

return fields\_Des;

//throw new UnsupportedOperationException("Implement this");

}

any missing or incomplete elements of the code

• the iteration method of part 5 is not fully working and scan test is also not fully working.

any changes made to the original API

• There was a typo in part for for getting get tuple to work. For getting the location for the tuple.

the total amount of time spent for this assignment (also the days when you worked on this

assignment and the amount of time spent for each day)

• Every day for at least several hours each day, probably spent a total of at least 40 hours total coding time for this assignment.

suggestions or comments if any

Should have been explained better in class. Having 310 as the only java pre rec is a complete understatement! Java should have or should be still also taught in the class to teach a lot of the concepts of java that were not taught in 310. Pretty much everything for this assignment I had to learn myself and was not taught in 310.