CSCE 310: Database Systems
Homework 7
Raymond Zhu
923008555

Problem 1:

a)

Capacity of a disk = # surface * # track * # sector * sector size
Capacity of a disk = 10 * 100,000 * 1000 * 1024
Capacity of a disk = 1,000,000,000 * 1024
Capacity of a disk = 10^9 * 1024 bytes

c)

Maximum seek time = 1 + 0.0002*n where n = # track
Maximum seek time = 1 + 20 = 21 milliseconds

d)

Maximum rotational latency = time (seconds) / rotations per minute
Maximum rotational latency = 60 seconds / 10,000 rpm
Maximum rotational latency = 0.006 seconds

e)

Given a block size of 65,546 bytes (64 sectors)
Transfer time = # sectors for size of block * (rotational latency / # sectors)
Transfer time = 64 * (0.006 / 1000)
Transfer time = 64 * 6.0 x 10^{-6} seconds

f)

Average seek time = Maximum seek time / 3
Average seek time = 21 / 3
Average seek time = 7 milliseconds

g)

Average rotational latency = half of maximum rotational latency
Average rotational latency = 0.006 / 2
Average rotational latency = 0.003 seconds

Problem 2:

If we are using a Megatron 747 disk with an average seek time of 6.46, rotational latency of 4.17, and transfer time of 0.13, we can calculate completion times given the sum of rotational latency, transfer time, and the incurred seek time.

a)

Initially reading down to 8000 from 32000, and then by that time 4000 has arrived so we seek 4000. 40000 hasn't arrived yet so we seek to 48000. Lastly, we then seek to 40000

Cylinder of Request	Time Completed (ms)
8000	11.3
4000	17.6
48000	33.9
40000	41.2

b)

Cylinder of Request	Time Completed (ms)
8000	11.3
48000	26.6
4000	39.9
40000	54.2

Problem 3:

a) 00111011 – 5 ones

Even parity = 001110111 – adding 1 to make the # of 1-bits even
Odd parity = 001110110 – adding 0 to keep the # of 1-bits odd

b) 0000000 – 0 ones

Even parity = 00000000 – adding 0 to keep the # of 1-bits even
Odd parity = 00000001 – adding 1 to make the # of 1-bits odd

c) 10101101 – 5 ones

Even parity = 101011011 – adding 1 to make the # of 1-bits even
Odd parity = 101011010 – adding 0 to keep the # of 1-bits odd

Problem 4:

a) Given 01010110 11000000 00111011, and 11111011
Take the #1-bits (mod 2) where even = 0 and odd = 1
1st bit = [0101] = 0
2nd bit = [1101] = 1
3rd bit = [0011] = 0
4th bit = [1011] = 1
5th bit = [0011] = 0
6th bit = [1000] = 1
7th bit = [1011] = 1
8th bit = [0011] = 0
The recovered block is: 01010110

b) Given 11110000 11111000, 00111111, and 00000001
1st bit = [1100] = 0
2nd bit = [1100] = 0
3rd bit = [1110] = 1
4th bit = [1110] = 1
5th bit = [0110] = 0
6th bit = [0010] = 1
7th bit = [0010] = 1
8th bit = [0011] = 0
The recovered block is: 00110110

Problem 5:

a) Fields can start at any byte
bytes = 15 (character string of length 15) + 2 (size of integer) + 3 (size of date) + 5 (size of time)
bytes = 15 + 2 + 3 + 5
bytes = 25 bytes

b) Fields must start at a byte that is a multiple of 4. Need to pad spacing to values that aren't multiples of four
bytes = [15 + [1] (16 mod 4)] + [2 + [2] (4 mod 4)] + [3 + [1] (4 mod 4)] + [5 (last field)]
bytes = 16 + 4 + 4 + 5
bytes = 29 bytes

c) Fields must start at a byte that is a multiple of 8. Need to pad spacing for values that aren't multiples of eight
bytes = [15 + [1] (16 mod 8)] + [2 + [6] (8 mod 8)] + [3 + [5] (8 mod 8)] + [5 (last field)]
bytes = 16 + 8 + 8 + 5
bytes = 37 bytes

Problem 6:

StudentMajorNoServer

```
public class StudentMajorNoServer {
    public static void main(String[] args) {
        try {
            // analogous to the driver
            SimpleDB.init("studentdb");

            FileMgr fm = SimpleDB.fileMgr();

            // analogous to the connection
            Transaction tx = new Transaction();

            // analogous to the statement
            String qry = "select SName, DName "
                + "from DEPT, STUDENT "
                + "where MajorId = DId";
            Plan p = SimpleDB.planner().createQueryPlan(qry, tx);

            // analogous to the result set
            Scan s = p.open();

            System.out.println("Name\tMajor");
            while (s.next()) {
                String sname = s.getString("sname"); //SimpleDB stores field names
                String dname = s.getString("dname"); //in lower case
                System.out.println(sname + "\t" + dname);
            }
            s.close();
            tx.commit();

            System.out.println("Average block read: " + fm.readAverage() + " ns");
            System.out.println("Average block write: " + fm.writeAverage() + " ns");
            fm.resetAverages();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
Name    Major
joe     compsci
max     compsci
lee     compsci
amy     math
sue     math
kim     math
pat     math
bob     drama
art     drama
transaction 2 committed
Average block read: 56412 ns
Average block write: 16663746 ns
```

FileMgr

```
public class FileMgr {
   private File dbDirectory;
   private boolean isNew;
   private Map<String,FileChannel> openFiles = new HashMap<String,FileChannel>();
   private ArrayList<Long> readList = new ArrayList<Long>();
   private ArrayList<Long> writeList = new ArrayList<Long>();
```

```
public long writeAverage() {
    long avg = 0;
    //System.out.println(writeList.toString());
    for(long x : writeList) {
        //System.out.println(x);
        avg += x;
    }
    avg /= writeList.size();
    return avg;
}

public long readAverage() {
    long avg = 0;
    //System.out.println(readList.toString());
    for(long x : readList) {
        //System.out.println(x);
        avg += x;
    }
    avg /= readList.size();
    return avg;
}

public void resetAverages() {
    readList = new ArrayList<Long>();
    writeList = new ArrayList<Long>();
}
```

```
synchronized void read(Block blk, ByteBuffer bb) {
   try {
      bb.clear();
      FileChannel fc = getFile(blk.fileName());
      long start = java.lang.System.nanoTime();
      fc.read(bb, blk.number() * BLOCK_SIZE);
      long end = java.lang.System.nanoTime();
      long diff = end - start;
      readList.add(diff);
   }
   catch (IOException e) {
      throw new RuntimeException("cannot read block " + blk);
   }
}

/**
 * Writes the contents of a bytebuffer into a disk block.
 * @param blk a reference to a disk block
 * @param bb  the bytebuffer
 */
synchronized void write(Block blk, ByteBuffer bb) {
   try {
      bb.rewind();
      FileChannel fc = getFile(blk.fileName());
      long start = java.lang.System.nanoTime();
      fc.write(bb, blk.number() * BLOCK_SIZE);
      long end = java.lang.System.nanoTime();
      long diff = end - start;
      writeList.add(diff);
   }
   catch (IOException e) {
      throw new RuntimeException("cannot write block" + blk);
   }
}
```

Problem 7:

```
Start of TestFileMgr
new transaction: 1
recovering existing database
transaction 1 committed
home directory = C:\Users\rayzr
Block 89 contains hello
Block 90 contains boolean value true
Block 91 contains boolean value false
Block 92 contains boolean value null
end of TestFileMgr
```

```
public class Page {

    public static final BOOLEAN bFalse = new BOOLEAN(0, "false");
    public static final BOOLEAN bTrue = new BOOLEAN(1, "true");
    public static final BOOLEAN bNull = new BOOLEAN(2, "null");
```

```
public synchronized void setBoolean(int offset, String val) {
    if(val == "true") {
        this.setInt(offset, bTrue.intVal);
        this.setString(offset+4, bTrue.stringVal);
    }

    else if(val == "false") {
        this.setInt(offset, bFalse.intVal);
        this.setString(offset+4, bFalse.stringVal);
    }
    else if(val == "null") {
        this.setInt(offset, bNull.intVal);
        this.setString(offset+4, bNull.stringVal);
    }

}

public synchronized String getBoolean(int offset) {
    int x = this.getInt(offset);

    if(this.isBOOLEAN(offset+4, x)) {
        return this.getString(offset+4);
    }
    else
        return "";
}
```

```
public synchronized boolean isBOOLEAN(int offset, int val) {
    String bString = this.getString(offset);

    if(val==1 && bString.equals("true"))
        return true;
    else if(val==0 && bString.equals("false"))
        return true;
    else if(val==2 && bString.equals("null"))
        return true;
    else
        return false;
}

public static class BOOLEAN{

    int intVal;
    String stringVal;

    public BOOLEAN(int x, String y) {
        intVal = x;
        stringVal = y;
    }
}
```

```
Page p4 = new Page();
p4.setBoolean(40, "true");
blk = p4.append("junkfile");

Page p5 = new Page();
p5.read(blk);
String ss = p5.getBoolean(40);
System.out.println("Block " + blk.number() + " contains boolean value " + ss);

Page p6 = new Page();
p6.setBoolean(50, "false");
blk = p6.append("junkfile");

Page p7 = new Page();
p7.read(blk);
ss = p7.getBoolean(50);
System.out.println("Block " + blk.number() + " contains boolean value " + ss);

Page p8 = new Page();
p8.setBoolean(60, "null");
blk = p8.append("junkfile");

Page p9 = new Page();
p9.read(blk);
ss = p9.getBoolean(60);
System.out.println("Block " + blk.number() + " contains boolean value " + ss);
```

Problem 8:

```
private boolean firstBlock() {
    boolean status = currentblknum == 0;
    return status;
}

public void afterLast() {
    moveTo(tx.size(filename)-1);
}

public boolean previous() {
    while (true) {
        if (rp.previous())
            return true;
        if (firstBlock())
            return false;
        moveTo(currentblknum-1);
    }
}
```

```
public boolean previous() {
    if(currentslot < 0) {
        currentslot = BLOCK_SIZE / slotsize;
    }

    currentslot--;

    while(currentpos() >= 0) {
        int pos = currentpos();
        if (tx.getInt(blk, pos) == INUSE)
            return true;
        currentslot--;
    }

    return false;
}
```

```
14 1982 Nature: Antarctica
15 1988 Neil Diamond: Greatest Hits L
16 1996 Screamers
17 2005 7 Seconds
18 1994 Immortal Beloved
19 2000 By Dawn's Early Light
BACKWARD listing
19 2000 By Dawn's Early Light
18 1994 Immortal Beloved
17 2005 7 Seconds
16 1996 Screamers
15 1988 Neil Diamond: Greatest Hits L
14 1982 Nature: Antarctica
13 2003 Lord of the Rings: The Return
11 1999 Full Frame: Documentary Shorts
10 2001 Fighter
9 1991 Class of Nuke 'Em High 2
8 2004 What the #$*! Do We Know!?
7 1992 8 Man
6 1997 Sick
5 2004 The Rise and Fall of ECW
4 1994 Paula Abdul's Get Up & Dance
3 1997 Character
2 2004 Isle of Man TT 2004 Review
1 2003 Dinosaur Planet
transaction 2 committed
```