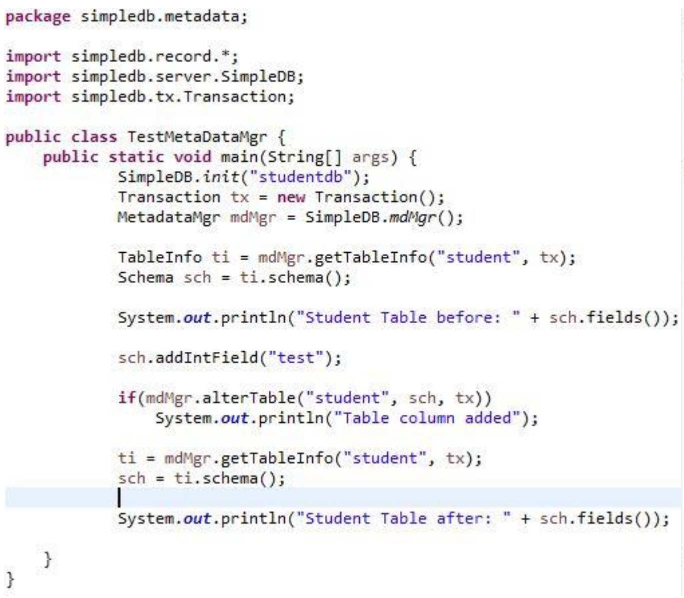
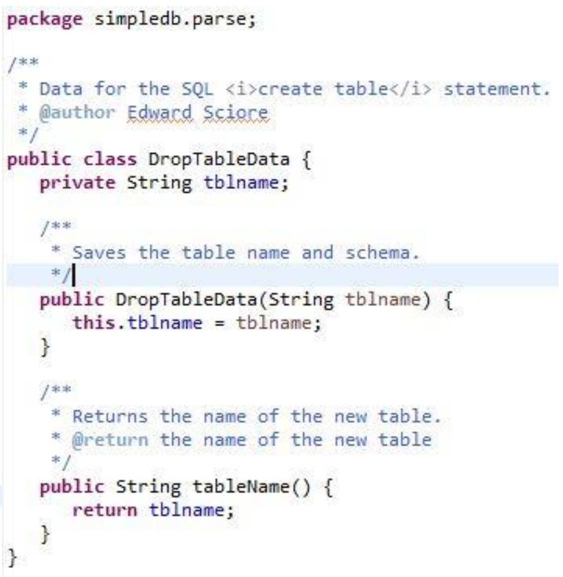
CSCE-310 Database Systems
Homework 8
Raymond Zhu
923008555

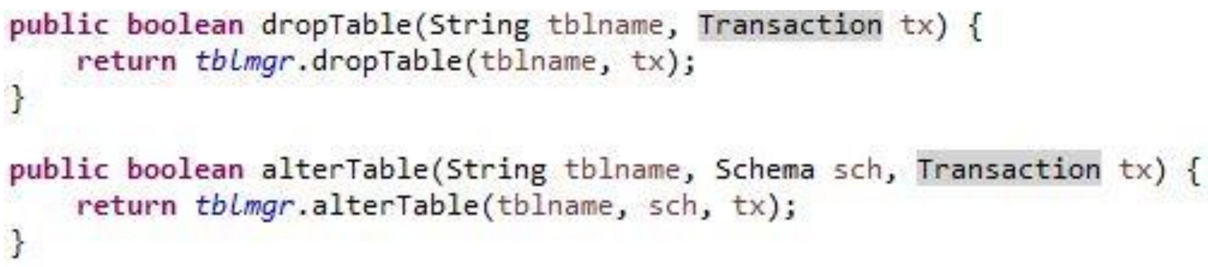
Problem 1:



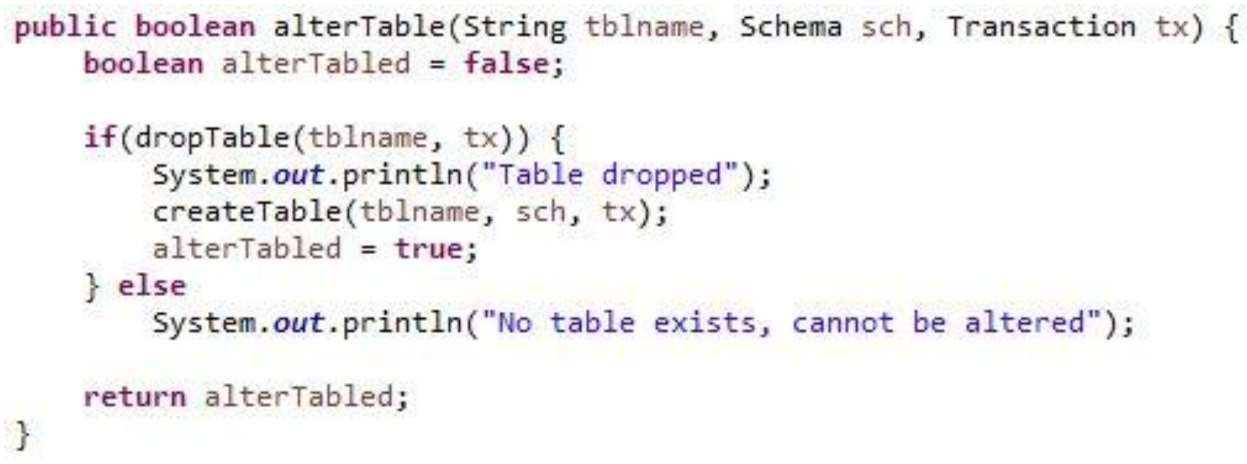
```java
package simpledb.metadata;

import simpledb.record.*;
import simpledb.server.SimpleDB;
import simpledb.tx.Transaction;

public class TestMetaDataMgr {
    public static void main(String[] args) {
            SimpleDB.init("studentdb");
            Transaction tx = new Transaction();
            MetadataMgr mdMgr = SimpleDB.mdMgr();

            TableInfo ti = mdMgr.getTableInfo("student", tx);
            Schema sch = ti.schema();

            System.out.println("Student Table before: " + sch.fields());

            sch.addIntField("test");

            if(mdMgr.alterTable("student", sch, tx))
                System.out.println("Table column added");

            ti = mdMgr.getTableInfo("student", tx);
            sch = ti.schema();

            System.out.println("Student Table after: " + sch.fields());

    }
}
```



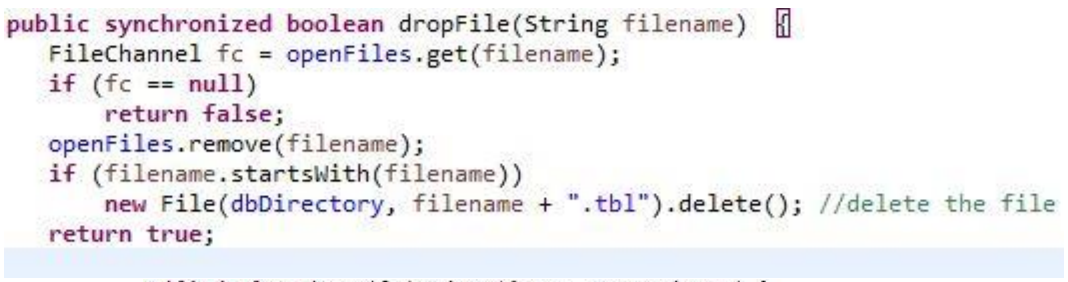
```java
package simpledb.parse;

/**
 * Data for the SQL <i>create table</i> statement.
 * @author Edward Sciore
 */
public class DropTableData {
   private String tblname;

   /**
    * Saves the table name and schema.
    */
   public DropTableData(String tblname) {
      this.tblname = tblname;
   }

   /**
    * Returns the name of the new table.
    * @return the name of the new table
    */
   public String tableName() {
      return tblname;
   }
}
```



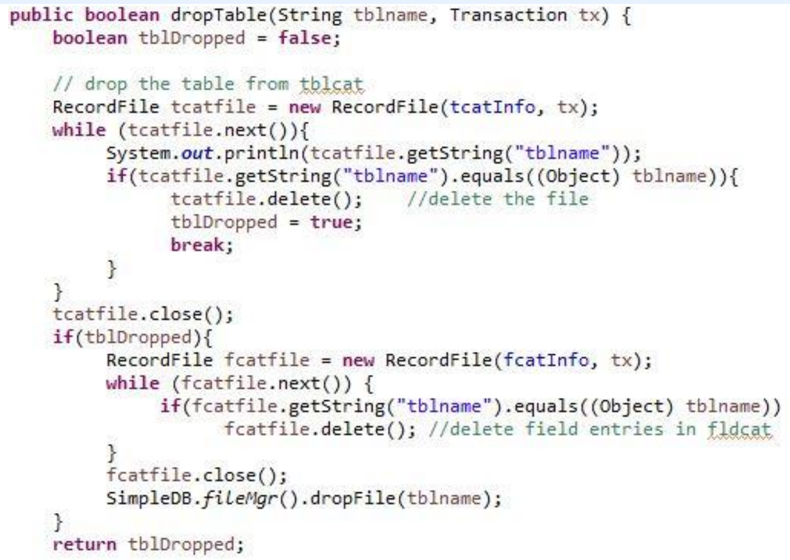
```java
public boolean dropTable(String tblname, Transaction tx) {
    return tblmgr.dropTable(tblname, tx);
}

public boolean alterTable(String tblname, Schema sch, Transaction tx) {
    return tblmgr.alterTable(tblname, sch, tx);
}
```



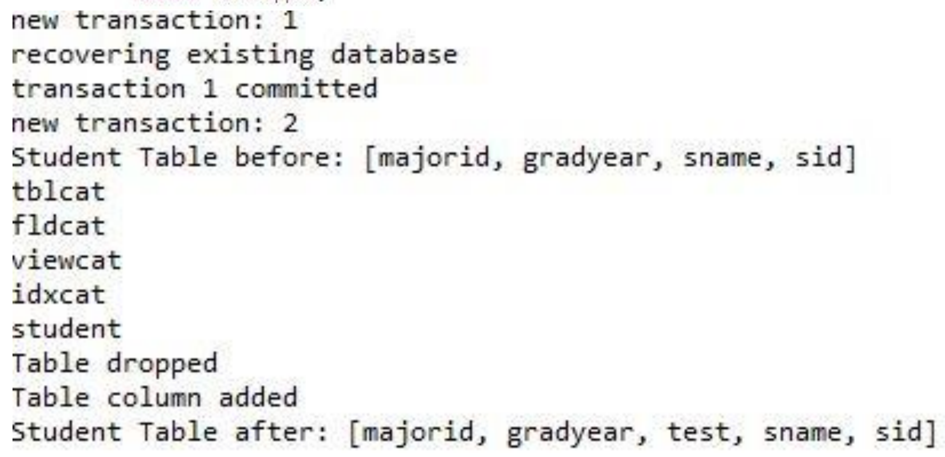
```java
public boolean alterTable(String tblname, Schema sch, Transaction tx) {
    boolean alterTabled = false;

    if(dropTable(tblname, tx)) {
        System.out.println("Table dropped");
        createTable(tblname, sch, tx);
        alterTabled = true;
    } else
        System.out.println("No table exists, cannot be altered");

    return alterTabled;
}
```



```java
public synchronized boolean dropFile(String filename)  {
   FileChannel fc = openFiles.get(filename);
   if (fc == null)
       return false;
   openFiles.remove(filename);
   if (filename.startsWith(filename))
       new File(dbDirectory, filename + ".tbl").delete(); //delete the file
   return true;
```



```java
public boolean dropTable(String tblname, Transaction tx) {
    boolean tblDropped = false;

    // drop the table from tblcat
    RecordFile tcatfile = new RecordFile(tcatInfo, tx);
    while (tcatfile.next()){
         System.out.println(tcatfile.getString("tblname"));
         if(tcatfile.getString("tblname").equals((Object) tblname)){
               tcatfile.delete();    //delete the file
               tblDropped = true;
               break;
         }
    }
    tcatfile.close();
    if(tblDropped){
         RecordFile fcatfile = new RecordFile(fcatInfo, tx);
         while (fcatfile.next()) {
              if(fcatfile.getString("tblname").equals((Object) tblname))
                    fcatfile.delete(); //delete field entries in fldcat
         }
         fcatfile.close();
         SimpleDB.fileMgr().dropFile(tblname);
    }
    return tblDropped;
```

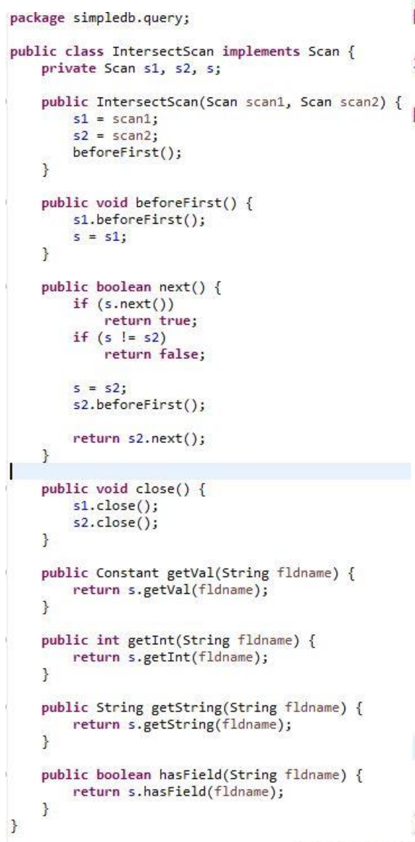


```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
Student Table before: [majorid, gradyear, sname, sid]
tblcat
fldcat
viewcat
idxcat
student
Table dropped
Table column added
Student Table after: [majorid, gradyear, test, sname, sid]
```

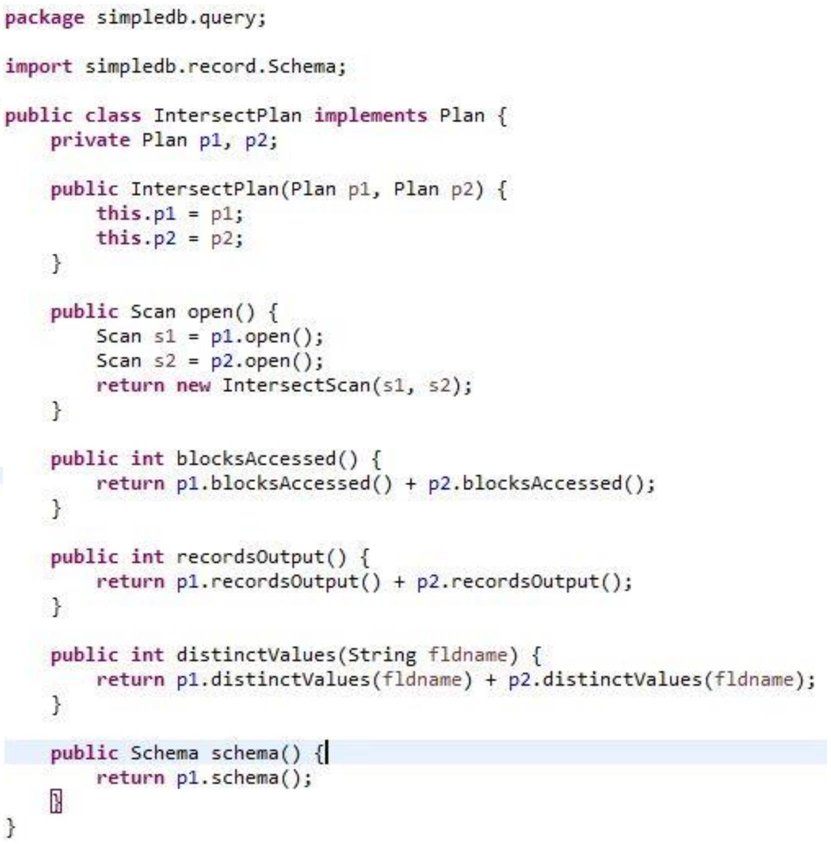
List of Classes used:

BasicUpdatePlanner.java
DropTableData.java
FileMgr.java
IndexUpdatePlanner.javaParser.java
Lexer.java
MetadataMgr.java
Planner.java
TableMgr.java
TestMetaDataMgr.java
UpdatePlanner.java

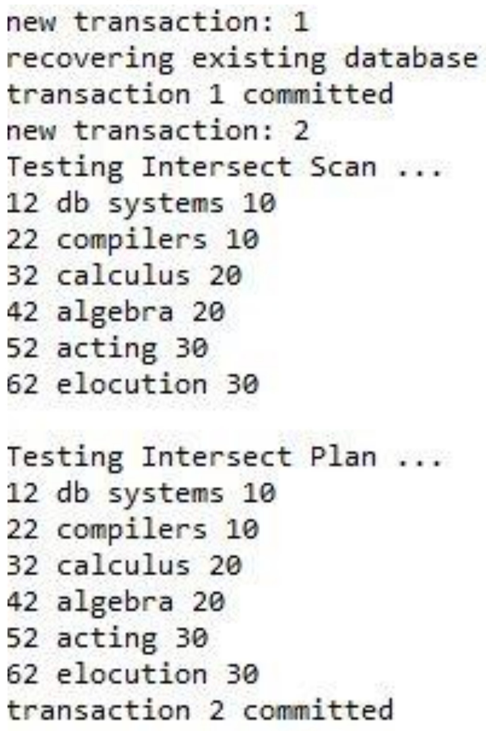
Problem 2:



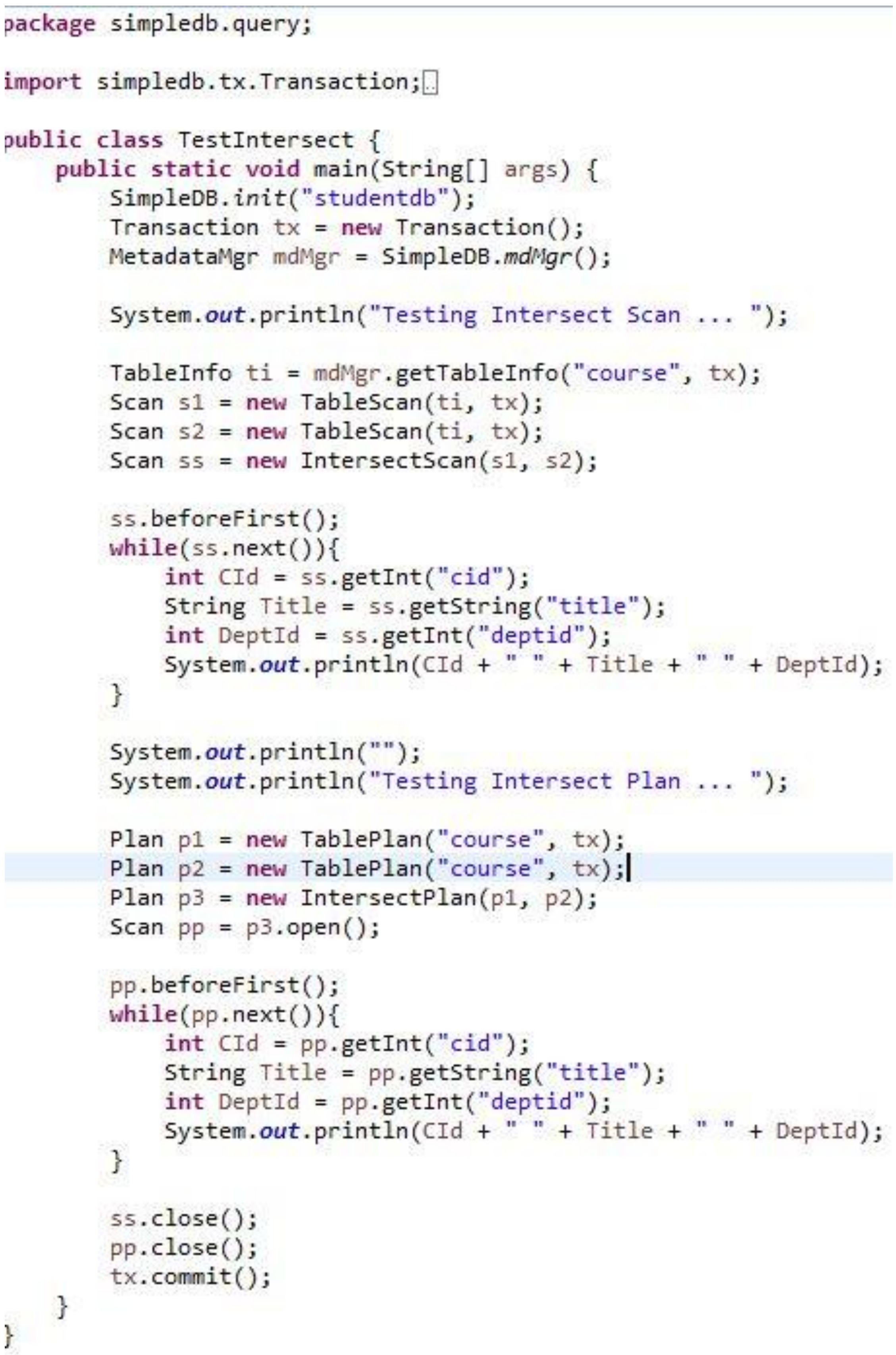
```java
package simpledb.query;

public class IntersectScan implements Scan {
    private Scan s1, s2, s;

    public IntersectScan(Scan scan1, Scan scan2) {
        s1 = scan1;
        s2 = scan2;
        beforeFirst();
    }

    public void beforeFirst() {
        s1.beforeFirst();
        s = s1;
    }

    public boolean next() {
        if (s.next())
            return true;
        if (s != s2)
            return false;

        s = s2;
        s2.beforeFirst();

        return s2.next();
    }

    public void close() {
        s1.close();
        s2.close();
    }

    public Constant getVal(String fldname) {
        return s.getVal(fldname);
    }

    public int getInt(String fldname) {
        return s.getInt(fldname);
    }

    public String getString(String fldname) {
        return s.getString(fldname);
    }

    public boolean hasField(String fldname) {
        return s.hasField(fldname);
    }
}
```



```java
package simpledb.query;

import simpledb.record.Schema;

public class IntersectPlan implements Plan {
    private Plan p1, p2;

    public IntersectPlan(Plan p1, Plan p2) {
        this.p1 = p1;
        this.p2 = p2;
    }

    public Scan open() {
        Scan s1 = p1.open();
        Scan s2 = p2.open();
        return new IntersectScan(s1, s2);
    }

    public int blocksAccessed() {
        return p1.blocksAccessed() + p2.blocksAccessed();
    }

    public int recordsOutput() {
        return p1.recordsOutput() + p2.recordsOutput();
    }

    public int distinctValues(String fldname) {
        return p1.distinctValues(fldname) + p2.distinctValues(fldname);
    }

    public Schema schema() {
        return p1.schema();
    }
}
```

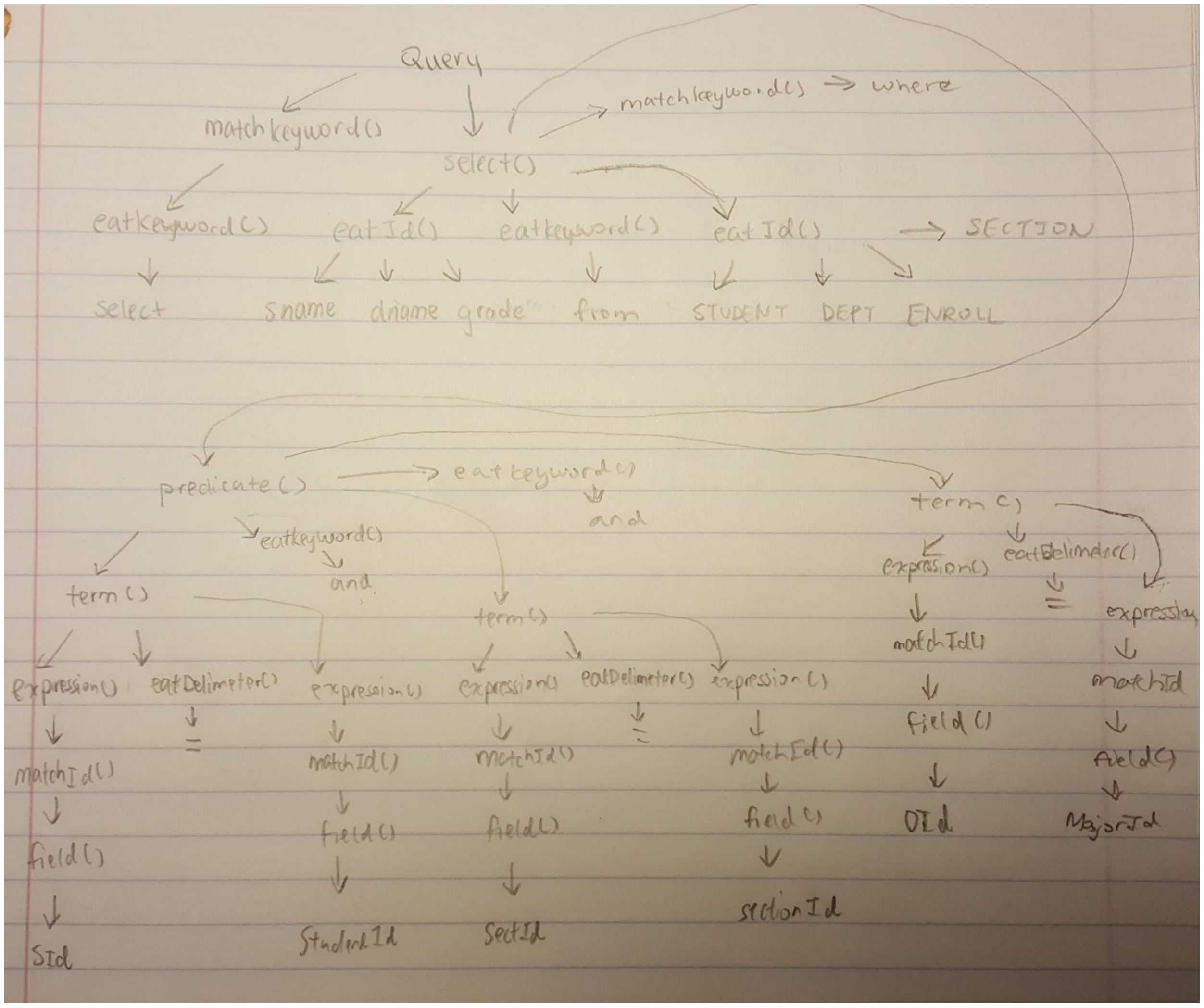


```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
Testing Intersect Scan ...
12 db systems 10
22 compilers 10
32 calculus 20
42 algebra 20
52 acting 30
62 elocution 30

Testing Intersect Plan ...
12 db systems 10
22 compilers 10
32 calculus 20
42 algebra 20
52 acting 30
62 elocution 30
transaction 2 committed
```

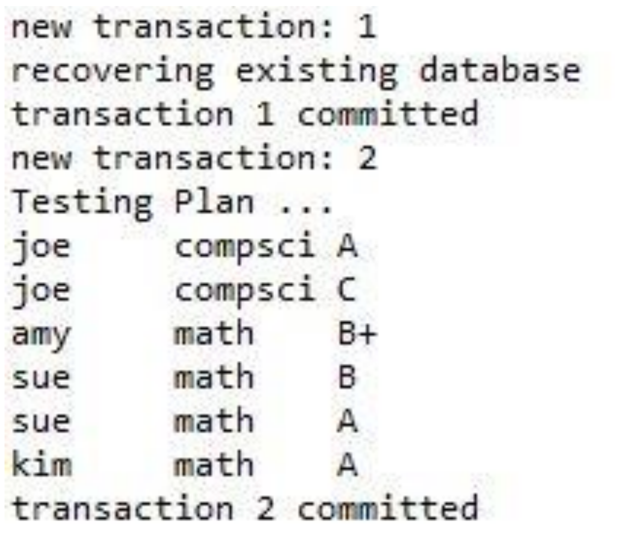


```
package simpledb.query;

import simpledb.tx.Transaction;

public class TestIntersect {
    public static void main(String[] args) {
        SimpleDB.init("studentdb");
        Transaction tx = new Transaction();
        MetadataMgr mdMgr = SimpleDB.mdMgr();

        System.out.println("Testing Intersect Scan ... ");

        TableInfo ti = mdMgr.getTableInfo("course", tx);
        Scan s1 = new TableScan(ti, tx);
        Scan s2 = new TableScan(ti, tx);
        Scan ss = new IntersectScan(s1, s2);

        ss.beforeFirst();
        while(ss.next()){
            int CId = ss.getInt("cid");
            String Title = ss.getString("title");
            int DeptId = ss.getInt("deptid");
            System.out.println(CId + " " + Title + " " + DeptId);
        }

        System.out.println("");
        System.out.println("Testing Intersect Plan ... ");

        Plan p1 = new TablePlan("course", tx);
        Plan p2 = new TablePlan("course", tx);
        Plan p3 = new IntersectPlan(p1, p2);
        Scan pp = p3.open();

        pp.beforeFirst();
        while(pp.next()){
            int CId = pp.getInt("cid");
            String Title = pp.getString("title");
            int DeptId = pp.getInt("deptid");
            System.out.println(CId + " " + Title + " " + DeptId);
        }

        ss.close();
        pp.close();
        tx.commit();
    }
}
```

Problem 3:

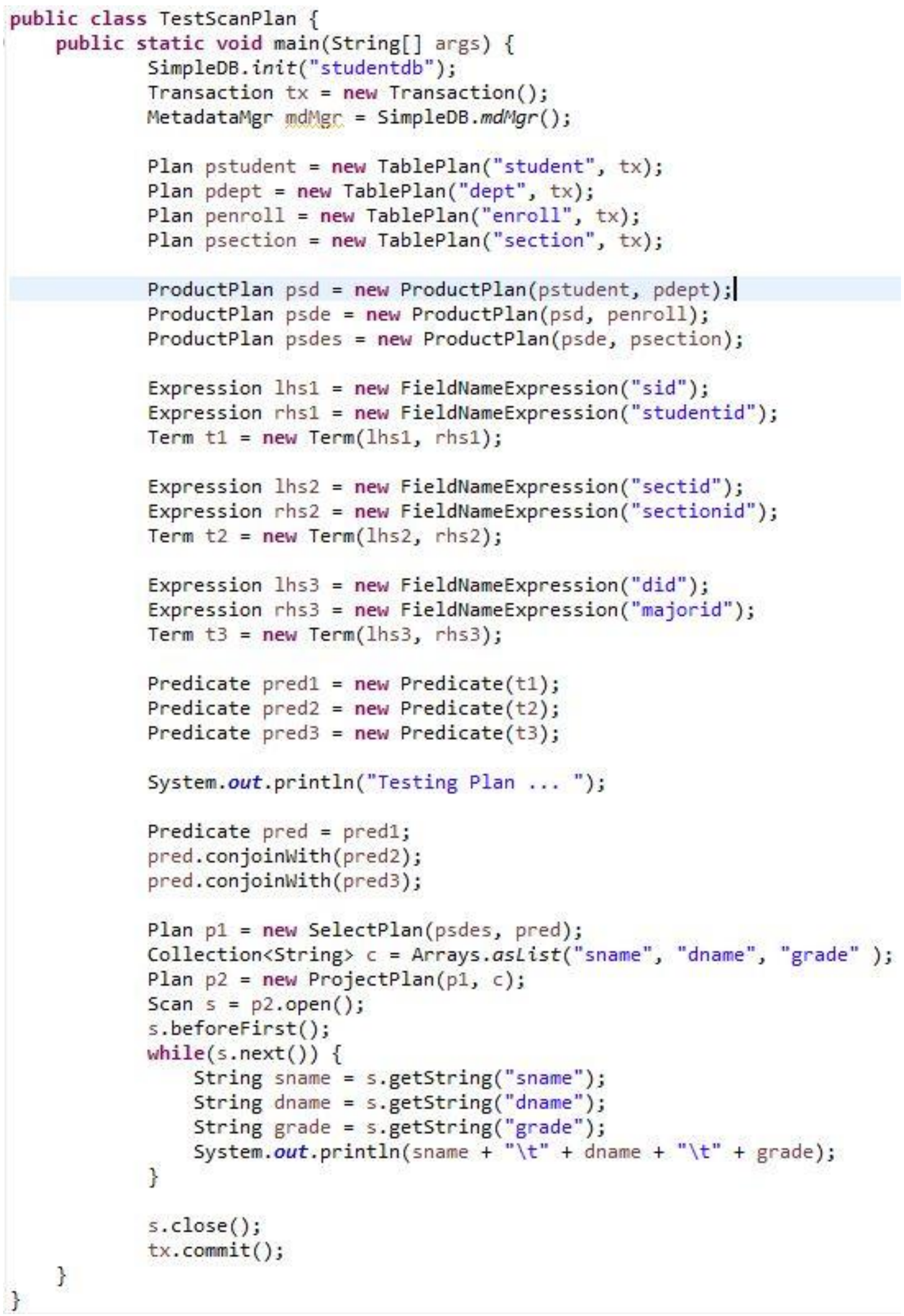


Problem 4



```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
Testing Plan ...
joe     compsci A
joe     compsci C
amy     math    B+
sue     math    B
sue     math    A
kim     math    A
transaction 2 committed
```

:

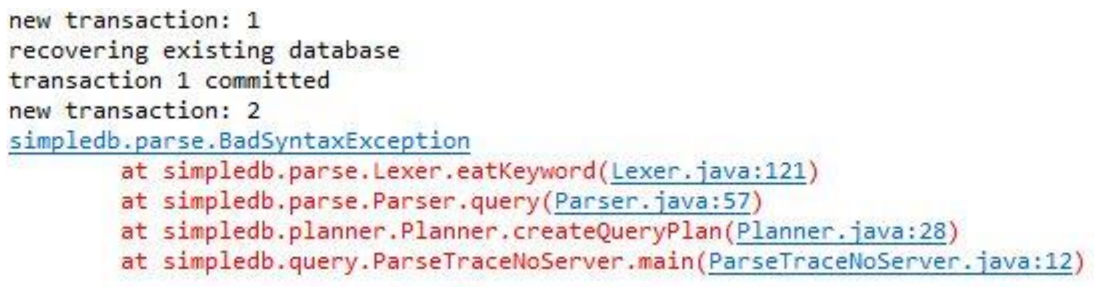


```java
public class TestScanPlan {
    public static void main(String[] args) {
            SimpleDB.init("studentdb");
            Transaction tx = new Transaction();
            MetadataMgr mdMgr = SimpleDB.mdMgr();

            Plan pstudent = new TablePlan("student", tx);
            Plan pdept = new TablePlan("dept", tx);
            Plan penroll = new TablePlan("enroll", tx);
            Plan psection = new TablePlan("section", tx);

            ProductPlan psd = new ProductPlan(pstudent, pdept);
            ProductPlan psde = new ProductPlan(psd, penroll);
            ProductPlan psdes = new ProductPlan(psde, psection);

            Expression lhs1 = new FieldNameExpression("sid");
            Expression rhs1 = new FieldNameExpression("studentid");
            Term t1 = new Term(lhs1, rhs1);

            Expression lhs2 = new FieldNameExpression("sectid");
            Expression rhs2 = new FieldNameExpression("sectionid");
            Term t2 = new Term(lhs2, rhs2);

            Expression lhs3 = new FieldNameExpression("did");
            Expression rhs3 = new FieldNameExpression("majorid");
            Term t3 = new Term(lhs3, rhs3);

            Predicate pred1 = new Predicate(t1);
            Predicate pred2 = new Predicate(t2);
            Predicate pred3 = new Predicate(t3);

            System.out.println("Testing Plan ... ");

            Predicate pred = pred1;
            pred.conjoinWith(pred2);
            pred.conjoinWith(pred3);

            Plan p1 = new SelectPlan(psdes, pred);
            Collection<String> c = Arrays.asList("sname", "dname", "grade" );
            Plan p2 = new ProjectPlan(p1, c);
            Scan s = p2.open();
            s.beforeFirst();
            while(s.next()) {
                String sname = s.getString("sname");
                String dname = s.getString("dname");
                String grade = s.getString("grade");
                System.out.println(sname + "\t" + dname + "\t" + grade);
            }

            s.close();
            tx.commit();
    }
}
```

Problem 5:

INSERT x INTO

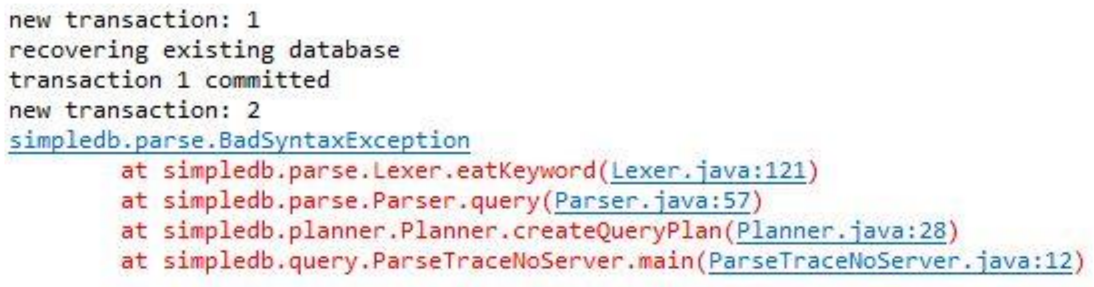
- A BadSyntaxException occurred because our parser was expecting the INTO keyword after INSERT. The value x is not listed as a keyword in the iniKeywords(). This occurs on line 135 in parser.java where the insert() method expects to eat two keywords and is thrown on line 119 in Lexer.java



```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
simpledb.parse.BadSyntaxException
	at simpledb.parse.Lexer.eatKeyword(Lexer.java:121)
	at simpledb.parse.Parser.query(Parser.java:57)
	at simpledb.planner.Planner.createQueryPlan(Planner.java:28)
	at simpledb.query.ParseTraceNoServer.main(ParseTraceNoServer.java:12)
```

INSERT INTO x (x x)

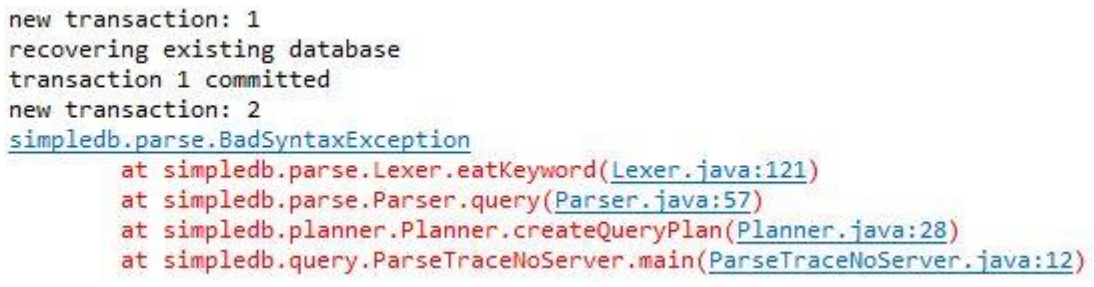
- A BadSyntaxException occurred because our parser was expecting a list. The given input contains two arguments that are not separated by commas. Because there is no comma, the parser expects the next token to be the delimiter ‘)’. This occurs on line 140 in parser.java and is thrown on line 81 in Lexer.java.



```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
simpledb.parse.BadSyntaxException
	at simpledb.parse.Lexer.eatKeyword(Lexer.java:121)
	at simpledb.parse.Parser.query(Parser.java:57)
	at simpledb.planner.Planner.createQueryPlan(Planner.java:28)
	at simpledb.query.ParseTraceNoServer.main(ParseTraceNoServer.java:12)
```

INSERT INTO x (x) VALUES x

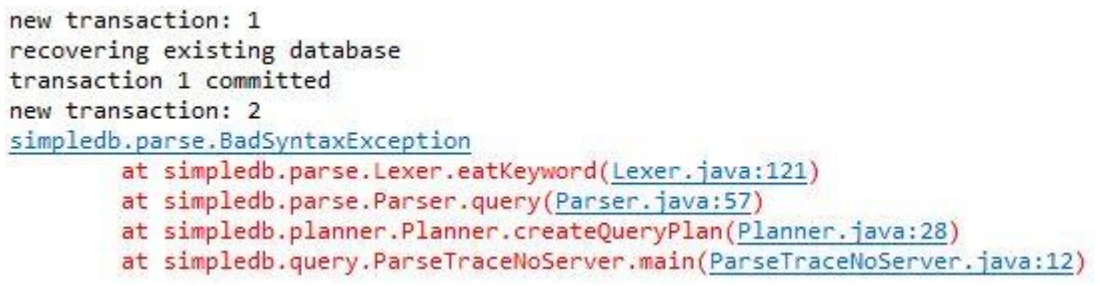
- A BadSyntaxException occurred because our parser requires a corresponding list for values. The error is thrown when the parser expects to eat the ‘(‘ delimiter, however gets the value x. This occurs on line 141 in parser.java and is thrown by line 81 in Lexer.java



```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
simpledb.parse.BadSyntaxException
	at simpledb.parse.Lexer.eatKeyword(Lexer.java:121)
	at simpledb.parse.Parser.query(Parser.java:57)
	at simpledb.planner.Planner.createQueryPlan(Planner.java:28)
	at simpledb.query.ParseTraceNoServer.main(ParseTraceNoServer.java:12)
```

UPDATE x SET x=y;

- A BadSyntaxException occurred because our parser was expecting an expression rather than a fieldname. This occurs on line 178 in parser.java in the modify method when we initialize a new predicate where the exception is thrown on line 91 in lexer.java. Because y is not a constant, the parser tries to parse the expression y as an integer.



```
new transaction: 1
recovering existing database
transaction 1 committed
new transaction: 2
simpledb.parse.BadSyntaxException
	at simpledb.parse.Lexer.eatKeyword(Lexer.java:121)
	at simpledb.parse.Parser.query(Parser.java:57)
	at simpledb.planner.Planner.createQueryPlan(Planner.java:28)
	at simpledb.query.ParseTraceNoServer.main(ParseTraceNoServer.java:12)
```

An Aggie does not lie, cheat, steal, or tolerate those who do.