TEST CSVFile

```
package database;

import databox.*;
import index.BPlusTree;
import index.BPlusTreeException;
import query.*;
import table.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.Rule;

import org.junit.rules.TemporaryFolder;

import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;

import static org.junit.Assert.assertEquals;

public class TestCSVFile {
    public static final String TestDir = "testDatabase";
    private Database db;
    private String filename;
    private File file;
    private String btree_filename = "TestBPlusTree";

    @Rule
    public TemporaryFolder tempFolder = new TemporaryFolder();

    @Before
    public void beforeEach() throws Exception {
        File testDir = tempFolder.newFolder(TestDir);
        this.filename = testDir.getAbsolutePath();
        this.db = new Database(filename);
        this.db.deleteAllTables();
        this.file = tempFolder.newFile(btree_filename);
    }

    @After
    public void afterEach() {
        this.db.deleteAllTables();
        this.db.close();
    }

    private BPlusTree getBPlusTree(Type keySchema, int order) throws
    BPlusTreeException {
        return new BPlusTree(file.getAbsolutePath(), keySchema, order);
    }

    @Test
    public void testCSVFileDB() throws DatabaseException, IOException {
        List<String> names = Arrays.asList("sid", "name", "major", "gpa");
        List<Type> types = Arrays.asList(Type.intType(), Type.stringType(20),
                Type.stringType(20), Type.floatType());
        Schema s = new Schema(names, types);

        // create table student
        String tableName = "student";
        db.createTable(s, tableName);

        List<String> studentLines = Files.readAllLines(Paths.get("students.csv"),
        Charset.defaultCharset());
```

```java

        Database.Transaction t1 = db.beginTransaction();

        // add recode for student
        for (String line : studentLines) {
            String[] splits = line.split(",");
            List<DataBox> values = new ArrayList<>();

            values.add(new IntDataBox(Integer.parseInt(splits[0])));
            values.add(new StringDataBox(splits[1].trim(), 20));
            values.add(new StringDataBox(splits[2].trim(), 20));
            values.add(new FloatDataBox(Float.parseFloat(splits[3])));

            RecordId rid = t1.addRecord(tableName, values);
            Record rec = t1.getRecord(tableName, rid);
            assertEquals(new Record(values), rec);
        }
        t1.end();
    }


    @Test
    public void testCSVFileBtree() throws DatabaseException, BPlusTreeException,
    IOException{
        List<String> names = Arrays.asList("sid", "name", "major", "gpa");
        List<Type> types = Arrays.asList(Type.intType(), Type.stringType(20),
                Type.stringType(20), Type.floatType());
        Schema s = new Schema(names, types);

        BPlusTree tree = getBPlusTree(Type.intType(), 2);

        // create table student
        String tableName = "student";
        db.createTable(s, tableName);

        List<String> studentLines = Files.readAllLines(Paths.get("students.csv"),
        Charset.defaultCharset());

        Database.Transaction t1 = db.beginTransaction();

        // add recode for student
        for (String line : studentLines) {
            String[] splits = line.split(",");
            ArrayList<DataBox> values = new ArrayList<>();

            values.add(new IntDataBox(Integer.parseInt(splits[0])));
            values.add(new StringDataBox(splits[1].trim(), 20));
            values.add(new StringDataBox(splits[2].trim(), 20));
            values.add(new FloatDataBox(Float.parseFloat(splits[3])));

            RecordId rid = t1.addRecord(tableName, values);
            tree.put(values.get(0), rid);
            Record rec = t1.getRecord(tableName, rid);
            assertEquals(new Record(values), rec);
        }

        Optional<RecordId> opt_rid = tree.get(new IntDataBox(10));
        if (opt_rid.isPresent()){
            RecordId rid = opt_rid.get();
            System.out.println(rid);
            Record rec = t1.getRecord(tableName, rid);
            System.out.println(rec);
        }

        t1.end();
    }

```

```java
@Test
public void testINLJStudentEnrollment() throws DatabaseException,
BPlusTreeException, IOException, QueryPlanException {


    // create second table
    String table1Name = "student";
    String table2Name = "enrollment";

    Database.Transaction t1 = db.beginTransaction();

    BPlusTree rightBtree = loadStudent(t1);
    loadEnrollment(t1);

    SequentialScanOperator leftSCO = new SequentialScanOperator(t1, table2Name);
    BtreeIndexScanOperator rightBTO = new BtreeIndexScanOperator(t1,
    table1Name, rightBtree);
    INLJOperator inljOperator = new INLJOperator(leftSCO, rightBTO, "sid",
    "sid", t1);

    Iterator<Record> recordIterator =  inljOperator.iterator();

    while (recordIterator.hasNext()){
        Record record = recordIterator.next();
        System.out.println(record);
    }


}

@Test
public void testINLJStudentEnrollmentCourses() throws DatabaseException,
BPlusTreeException, IOException, QueryPlanException {


    // create second table
    String table1Name = "student";
    String table2Name = "enrollment";
    String table3Nmae = "course";

    Database.Transaction t1 = db.beginTransaction();

    BPlusTree studentBtree = loadStudent(t1);
    loadEnrollment(t1);
    BPlusTree courseBtree = loadCourse(t1);

    SequentialScanOperator leftSCO = new SequentialScanOperator(t1, table2Name);
    BtreeIndexScanOperator rightBTO = new BtreeIndexScanOperator(t1,
    table1Name, studentBtree);
    INLJOperator inljOperator = new INLJOperator(leftSCO, rightBTO, "sid",
    "sid", t1);

    Iterator<Record> recordIterator =  inljOperator.iterator();

    List<Record> student_enrollment = new ArrayList<>();
    while (recordIterator.hasNext()){
        Record record = recordIterator.next();
        student_enrollment.add(record);
    }


    // schema
    List<String> names = Arrays.asList("cid", "cname", "dept");
    List<Type> types = Arrays.asList(Type.intType(), Type.stringType(20),
    Type.stringType(20));
    Schema s = new Schema(names, types);
```

```
        TestSourceOperator sourceOperator = new
        TestSourceOperator(student_enrollment, s, student_enrollment.size());

        BtreeIndexScanOperator courseBTO = new BtreeIndexScanOperator(t1,
        table3Nmae, courseBtree);

        INLJOperator inljOperator2 = new INLJOperator(sourceOperator, courseBTO,
        "cid", "cid", t1);

        Iterator<Record> recordIterator1 =  inljOperator2.iterator();

        while (recordIterator1.hasNext()){
            Record record = recordIterator1.next();
            System.out.println(record);
        }

    }



    private BPlusTree loadStudent(Database.Transaction t1) throws
    DatabaseException, BPlusTreeException, IOException{
        List<String> names = Arrays.asList("sid", "cid", "major", "gpa");
        List<Type> types = Arrays.asList(Type.intType(), Type.stringType(20),
                Type.stringType(20), Type.floatType());
        Schema s = new Schema(names, types);

        BPlusTree tree = getBPlusTree(Type.intType(), 2);

        // create table student
        String tableName = "student";
        db.createTable(s, tableName);

        List<String> studentLines = Files.readAllLines(Paths.get("students.csv"),
        Charset.defaultCharset());


        // add recode for student
        for (String line : studentLines) {
            String[] splits = line.split(",");
            ArrayList<DataBox> values = new ArrayList<>();

            values.add(new IntDataBox(Integer.parseInt(splits[0])));
            values.add(new StringDataBox(splits[1].trim(), 20));
            values.add(new StringDataBox(splits[2].trim(), 20));
            values.add(new FloatDataBox(Float.parseFloat(splits[3])));

            RecordId rid = t1.addRecord(tableName, values);
            tree.put(values.get(0), rid);
        }
        return tree;
    }

    private void loadEnrollment(Database.Transaction t1) throws  DatabaseException,
    BPlusTreeException, IOException{
        List<String> names = Arrays.asList("sid", "cid");
        List<Type> types = Arrays.asList(Type.intType(), Type.intType());
        Schema s = new Schema(names, types);

        // create table student
        String tableName = "enrollment";
        db.createTable(s, tableName);

        List<String> studentLines =
        Files.readAllLines(Paths.get("enrollments.csv"), Charset.defaultCharset());

```

```java
        // add recode for student
        for (String line : studentLines) {
            String[] splits = line.split(",");
            ArrayList<DataBox> values = new ArrayList<>();

            values.add(new IntDataBox(Integer.parseInt(splits[0])));
            values.add(new IntDataBox(Integer.parseInt(splits[1])));

            t1.addRecord(tableName, values);
        }
    }

    private BPlusTree loadCourse(Database.Transaction t1) throws
    DatabaseException, BPlusTreeException, IOException{
        List<String> names = Arrays.asList("cid", "cname", "dept");
        List<Type> types = Arrays.asList(Type.intType(), Type.stringType(20),
        Type.stringType(20));
        Schema s = new Schema(names, types);

        // create table student
        String tableName = "course";
        db.createTable(s, tableName);

        BPlusTree tree = getBPlusTree(Type.intType(), 2);


        List<String> courseLines = Files.readAllLines(Paths.get("courses.csv"),
        Charset.defaultCharset());

        // add recode for student
        for (String line : courseLines) {
            String[] splits = line.split(",");
            ArrayList<DataBox> values = new ArrayList<>();

            values.add(new IntDataBox(Integer.parseInt(splits[0])));
            values.add(new StringDataBox(splits[1].trim(), 20));
            values.add(new StringDataBox(splits[2].trim(), 20));

            RecordId rid = t1.addRecord(tableName, values);
            tree.put(values.get(0), rid);
        }

        return tree;
    }
}
```