CSCE 608

TinySQL Project

Andrew Bregger

UIN: 724008207

adb3649@tamu.edu

Jicheng Lu

UIN: 525004048

iceljc@tamu.edu

We implement this TinySQL by Java.

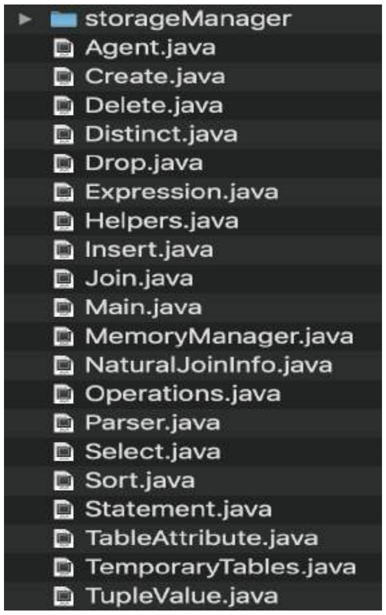
1 Introduction

In this project, we aim to design and implement a simple SQL interpreter, called Tiny-SQL. The queries in this Tiny-SQL consist of CREATE, DROP, SELECT, DELETE, INSERT. We design this database system using Java with a storage manager library.

2 System Architecture

2.1 Overview

In this section, we describe the program flow of our tiny-SQL interpreter. All the classes we developed, as well as the StorageManager, are in the "edu.student" package. The code files included in the package are illustrated in Fig.1.



```
▸ storageManager
  Agent.java
  Create.java
  Delete.java
  Distinct.java
  Drop.java
  Expression.java
  Helpers.java
  Insert.java
  Join.java
  Main.java
  MemoryManager.java
  NaturalJoinInfo.java
  Operations.java
  Parser.java
  Select.java
  Sort.java
  Statement.java
  TableAttribute.java
  TemporaryTables.java
  TupleValue.java
```

Figure 1: Code in the package.

2.2 Program Flow

Basically, there are six components for the Tiny-SQL, including Main, Agent, Parser, Query Manipulation, StoreManager, and Helpers. The program flow is presented in Fig.2.

The program flow can be briefly described as follows:

- A user-interface is created by the 'Main' class that can accept either one query or multiple queries;
- After a query is accepted, it is passed to an 'Agent';
- The 'Agent' first parse the query to different parts. If there is a 'WHERE' clause, it would be evaluated by the 'Expression' class;
- The 'Parser' returns a tree structure, and then the 'Agent' delivers the parsed query to corresponding manipulation section;
- The manipulation section processes the query using 'StorageManager', and returns the results.

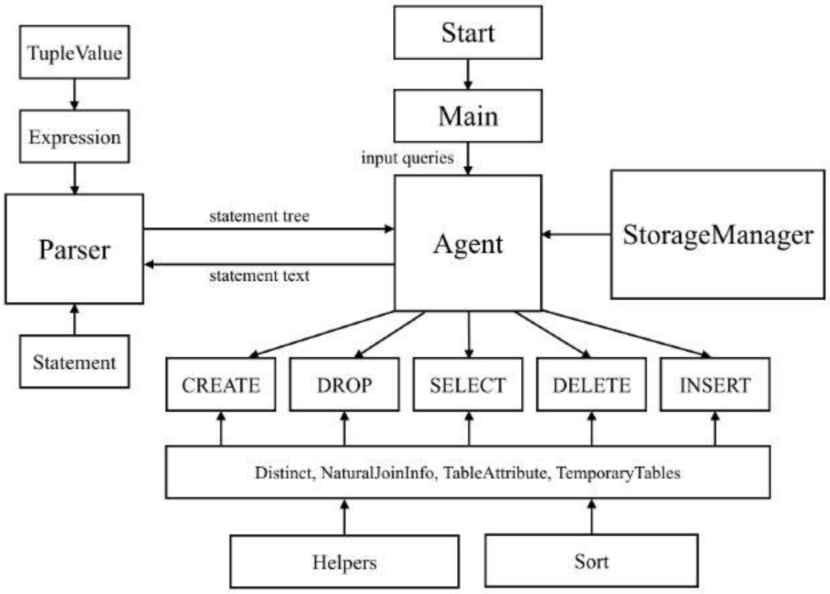


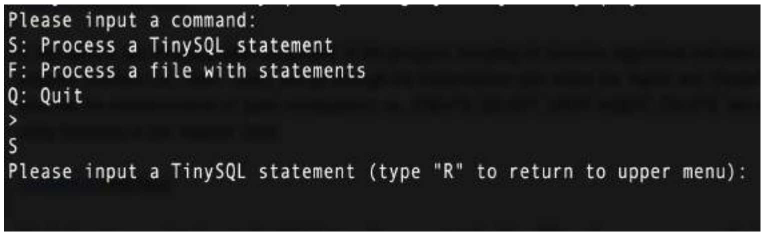
Figure 2: Program flow.

3 Implementation

In this section, we introduce each component of the program, including its function, algorithms and data structures. We start from the program interface (i.e., 'Main' class), and go through the interpretation part where the 'Agent' and 'Parser' are applied. Then we describe the implementation of query manipulation, i.e., CREATE, SELECT, DROP, INSERT, DELETE. We end up with mentioning some utility functions in the 'Helpers' class.

3.1 Interface

We designed a user-interface in the 'Main' class. Once we start the Tiny-SQL interpreter, we can see the interface as in Fig.3. If we only want to process one single query, input "S" and type the query in the command window. If multiple queries need to be handled, we input "F" and then type the file name, such as 'test.txt'. We can return back to the previous menu by inputting "R" and exit the software by inputting "Q".



```
Please input a command:
S: Process a TinySQL statement
F: Process a file with statements
Q: Quit
>
S
Please input a TinySQL statement (type "R" to return to upper menu):
```

Figure 3: The user-interface.

3.2 Interpretation

3.2.1 Agent

The 'Agent' class is the pivot of the entire program. It first accepts the user input from the interface, and then delivers the SQL query to 'Parser.' The 'Parser' returns a statement tree object back to the 'Agent'. Next, according to the type of the query, the 'Agent' decides what manipulation section the query would go to (i.e., CREATE, SELECT, DROP, INSERT, DELETE). Once the manipulation is completed, the 'Agent' would tell the user that the query has been processed successfully. Moreover, the 'Agent' can measure the running time and disk I/O's when processing each query.

3.2.2 Parser

The 'Parser' receives the query string from 'Agent'. First, it separates each part of the query by using regular expression. Then it creates a root node whose attribute is the prime type of the query. Thus, the attribute of the root node can only be CREATE, SELECT, DROP, INSERT or DELETE. Next, it uses a recursive method to extract each part of the query, and generates a statement tree object based on the Tiny-SQL grammar. Moreover, if there is a 'WHERE' clause in the original query, we keep the entire condition and build a subtree using a stack structure. In order to do this, we assign each operation with a priority value, and recursively put in and extract elements in the stack. As a result, we can obtain a single subtree for the 'WHERE' condition. Note that the 'Statement' class in the package defines a tree structure, where we can do operations during the parsing, such as getting the attribute or children of a node. A parse tree example of a typical SELECT query, "select homework from course where exam = 100", is given below:

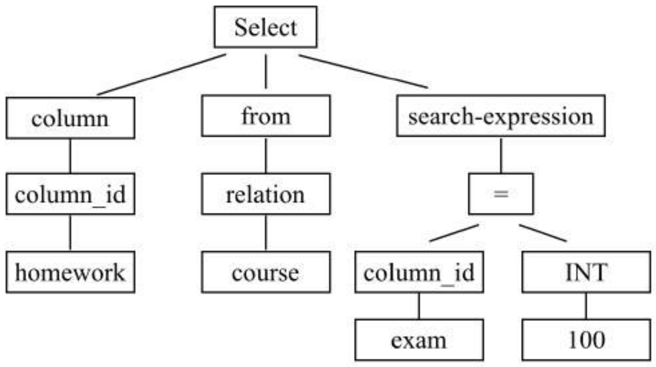


Figure 4: A parse tree example.

3.2.3 Expression

We design an 'Expression' class to evaluate the WHERE clause. That is, to verify whether the clause is valid. According to the Tiny-SQL grammar, we need to deal with the following operators that may appear in search condition: $=$, $+$, $-$, $*$, $<$, $>$, AND, OR. Note that we need to check tuples' field types when evaluating those expressions. We also create a 'TupleValue' class as a helper to verify whether the tuples match in their field types at both sides when dealing with "=" operation.

3.3 Relation Manipulation

3.3.1 CREATE

The "Create" class contains the functions need to analyze the input "Statement" and extract the necessary information about the table that is be created. These function will build the schema and create relation with the "SchemaManager". This operation doesn't have the ability to print anything to console or an output file.

3.3.2 INSERT

The "Insert" class is designed to insert tuples at the end of the relation. It is not expected to perform any sorting of the the current relation in order to insert the tuple into the table such that the table maintains sorting. This class provides the functions necessary extract field names from the relation and build the corresponding Fields such that the schema of the new tuple matches that of the table being inserted.

This class has another operation, it is able to insert the results of a selection to a table as long as the tuples schema match. Because of the implementation of the schema they are not structurally compared; therefor, if the results of the select are a different schema reference, the tuples will need to be rebuild using the correct schema.

3.3.3 SELECT

The "Select" class is designed to handle all cases that the select statement should need to handle. It analyzes the input and determines which algorithm should be performed by looking at how many relations are in the from clause and whether there is a where clause. If there is only one relation, the it looks at how many blocks this relation occupies and will perform the operations accordingly. We decides what method to be run is first how many tables are to be considered. If it is one, then it only depends on how many blocks the relation uses. If it can fit into main memory then all the relation is loaded into main memory a processed accordingly. If it doesn't, then it uses the one pass selection and projection algorithms.

In addition to doing single joins, the interpreter supports the joining of multiple tables using natural join when there is a common attribute for all tables or cross join otherwise. When cross join is applied, an optimization is applied that reorders the tables such that the smallest tables are first followed by the larger tables. This optimization is discussed more in the optimization section of this report. When natural join is applied, the first step is to identify which attributes can be joined on and in which order the tables must be joined it to guarantee the tables can be joined. If there are multiple possible attributes it usually chooses the first found unless there is a column equality specified by the where condition. If one is found in the where condition then that the referenced attribute is used at the join column and the where clause is removed because it is redundant.

The algorithms that are implemented are the single pass natural and cross join. For larger tables, when a cross join is needed then nested loop algorithm is used, for theta join (natural join with a condition) a nested loop algorithm is also used. For natural join, the two pass algorithm was attempted; however, it was not completed as of the time of this writing. It will be attempted again before submission so it could be in the code just the results will not demonstrate it.

If the join algorithm supports a condition, then the condition is applied then. If not, then it is applied following the join during the selection phase of the pipeline. There is currently a problem with the condition being applied to tables that do not have the attributes needed by the condition due to the joining order and which column was joined on.

The secondary operation of this class is to perform selection and projection operators onto the desired tables. The projections are mentioned above when conditions are being applied either during the joining or as a post process. The selection is performed by the "Projection" class. This class handles the the writing of the resulting table either to the necessary table or a desired output. If the relation is small enough to fit into memory then it is manipulated all at once. If it is too big, then one block is used as an output block and the rest of the memory is used to hold tuples for processing. If the sorting or distinct operations are needed to be performed then they are applied after selection.

3.3.4 DELETE

The "Delete" class will delete tuples that satisfy a given condition. Due to the implementation of the ArrayList in Java, when am element is removed, the element is filled by the next element so that holes do not occur when the tuples are being written back to the block.

3.3.5 DROP

The "Drop" class will simply delete the content of an entire relation. Since we are not implementing foreign keys a table can be deleted with out worrying about their relation to other tables.

3.3.6 Helpers

The "Helpers" class implements common functions that could be used by many different classes. Most of the functions relate to building and managing tuples and schemas. It also contains the a function for performing the first pass of the two pass algorithms.

3.3.7 Sort

The "Sort" class implements all of the necessary functions for comparing tuples and sorting relations. There are two functions for sorting and two functions for comparing tuples. The two sorting algorithms are differ because one is the single pass sorting algorithm that can be performed completely in memory and the other is the two pass sorting algorithm which sorts sublists of the relation and them merges the sublists into a single block by removing the minimum relation. The two comparison methods can compare a tuple completely or by a single attribute of the tuple. If two tuples are being compared by an attribute, then tuples are only compared by that attribute and the others are ignored. This allows for both of the sorting algorithms to be sorted by an attribute if one is given.

4 Optimization

The optimization we implemented is the ordering of the cross join order such that the smaller tables are joined first and the larger tables are the last. This will save some cost in the performance because the system will only have to handle small table at the beginning. Although it is cross join that might only last for the first join, increasing the efficiency of one join is still a benefit.

5 Test

5.1 Launch the Tiny-SQL Interpreter

We package the interpreter into a *.jar* file. When we need to launch the Tiny-SQL interpreter, just type "java -jar tinysql.jar" in the command window. We then set up four tests to evaluate the performance of the Tiny-SQL interpreter.

5.2 Single-Table Test

We first test the queries on a single table. The test queries are presented in Box 1. We test the all the five basic operations, CREATE, DROP, SELECT, DELETE, INSERT, without search condition.

Box 1. Test queries for a single table.

```
CREATE TABLE course (sid INT, homework INT, project INT, exam INT, grade STR20)
INSERT INTO course (sid, homework, project, exam, grade) VALUES (1, 99, 100, 100, "A")
SELECT * FROM course
INSERT INTO course (sid, homework, project, exam, grade) VALUES (2, 100, 100, 100, "E")
SELECT * FROM course
INSERT INTO course (sid, homework, project, exam, grade) VALUES (3, 99, 100, 100, "A")
SELECT * FROM course
INSERT INTO course (sid, homework, project, exam, grade) VALUES (4, 99, 100, 100, "B")
SELECT * FROM course
DELETE FROM course
SELECT * FROM course
DROP TABLE course
CREATE TABLE course (sid INT, homework INT, project INT, exam INT, grade STR20)
INSERT INTO course (sid, homework, project, exam, grade) VALUES (1, 99, 100, 100, "A")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (3, 100, 100, 98, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (3, 100, 69, 64, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (15, 100, 50, 90, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (15, 100, 99, 100, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (17, 100, 100, 100, "A")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (2, 100, 100, 99, "B")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (4, 100, 100, 97, "D")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (5, 100, 100, 66, "A")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (6, 100, 100, 65, "B")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (7, 100, 50, 73, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (8, 50, 50, 62, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (9, 50, 50, 61, "D")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (10, 50, 70, 70, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (11, 50, 50, 59, "D")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (12, 0, 70, 58, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (13, 0, 50, 77, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (14, 50, 50, 56, "D")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (17, 100, 100, 100, "A")
SELECT * FROM course
SELECT sid, grade FROM course
SELECT sid, course.grade FROM course
DROP TABLE course
```

Part of the test results are presented Appendix.A. Note that we print out disk I/O's and query processing time after each operation so that we can evaluate the performance of this Tiny-SQL interpreter. It can be seen that the interpreter processes these test queries correctly.

5.3 'Where' Condition Test

Here we test the queries on a single table with WHERE clause. The test queries are presented in Box 2. We apply multiple search conditions after each selection. This intends to test the validity when processing the search condition.

Box 2. Test queries for a single table with search condition.

```
CREATE TABLE course (sid INT, homework INT, project INT, exam INT, grade STR20)
INSERT INTO course (sid, homework, project, exam, grade) VALUES (1, 99, 100, 100, "A")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (3, 100, 100, 98, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (3, 100, 69, 64, "C")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (15, 100, 50, 90, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (15, 100, 99, 100, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (17, 100, 100, 100, "A")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (2, 100, 100, 99, "B")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (4, 100, 100, 97, "D")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (16, 0, 0, 0, "E")
INSERT INTO course (sid, homework, project, exam, grade) VALUES (17, 100, 100, 100, "A")
SELECT * FROM course WHERE exam = 100
SELECT * FROM course WHERE grade = "A"
SELECT * FROM course WHERE (exam + homework) = 200
SELECT * FROM course WHERE exam = 100 AND project = 100
SELECT * FROM course WHERE exam = 100 OR exam = 99
SELECT * FROM course WHERE exam > 70
SELECT * FROM course WHERE exam = 100 OR homework = 100 AND project = 100
DROP TABLE course
```

Part of the test results are presented in Appendix.B. Again, we print out disk I/O's and query processing time after each operation so that we can evaluate the performance of this Tiny-SQL interpreter. It can be seen that the interpreter handles these search conditions correctly.

5.4 Two-Table Test

Here we test the queries on two tables. The test queries are presented in Box 3. We first create two different tables with common attribute and then select all columns from these two relations. This is set to test the join between two tables.

Box 3. Test queries for two-table join.

```
CREATE TABLE course (sid INT, homework INT, project INT, grade STR20)
INSERT INTO course (sid, homework, project, grade) VALUES (1, 99, 100, ”A”)
INSERT INTO course (sid, homework, project, grade) VALUES (1, 98, 100, ”A”)
INSERT INTO course (sid, homework, project, grade) VALUES (1, 97, 100, ”A”)
INSERT INTO course (sid, homework, project, grade) VALUES (2, 100, 90, ”B”)
INSERT INTO course (sid, homework, project, grade) VALUES (3, 100, 80, ”C”)
SELECT * FROM course
CREATE TABLE course2 (sid INT, exam INT, grade STR20)
INSERT INTO course2 (sid, exam, grade) VALUES (1, 95, ”A”)
INSERT INTO course2 (sid, exam, grade) VALUES (1, 94, ”A”)
INSERT INTO course2 (sid, exam, grade) VALUES (4, 85, ”B”)
INSERT INTO course2 (sid, exam, grade) VALUES (5, 75, ”C”)
SELECT * FROM course2
SELECT * FROM course,course2
DROP TABLE course
DROP TABLE course2
```

Part of the test results are presented in Appendix.C. The disk I/O’s and query processing time are listed after each operation. We can see that the interpreter implements the two-table join correctly.

5.5 Multiple-Table Test

Here we test the queries on multiple tables. The test queries are presented in Box 4. We first create three different tables and then select all columns from these two relations. Note we also apply a search condition based on multi-table join. This test aims to verify the multi-table join.

Box 4. Test queries for multi-table join.

```
create table a (sid int, name str20)
create table b (id int, class str20)
create table c (d int, person str20)
insert into a(sid, name) values(0, ”bob”)
insert into a(sid, name) values(1, ”bob”)
insert into a(sid, name) values(2, ”bob”)
insert into b(id, class) values(0, ”db”)
insert into b(id, class) values(1, ”ai”)
insert into b(id, class) values(2, ”ml”)
insert into c(d, person) values(0, ”a”)
insert into c(d, person) values(1, ”b”)
insert into c(d, person) values(2, ”c”)
select * from a, b, c
select * from a, b, c where a.sid = b.id and c.d ¡ b.id
DROP TABLE a
DROP TABLE b
DROP TABLE c
```

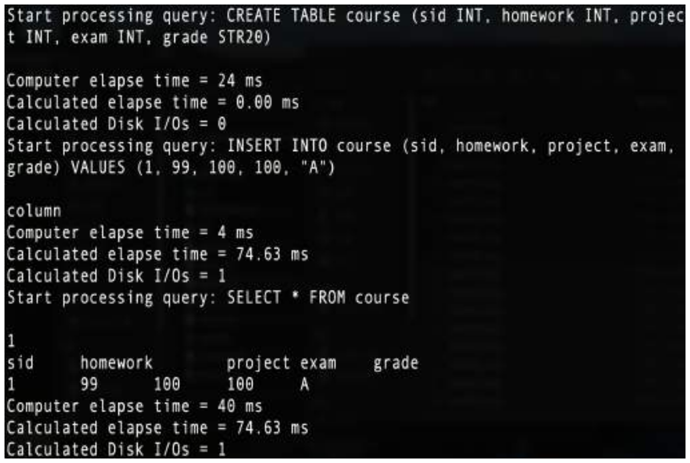
Part of the test results are presented in Appendix.D. The disk I/O's and query processing time are listed after each operation. We can see that the interpreter handles the multi-table join correctly.

6 Conclusion

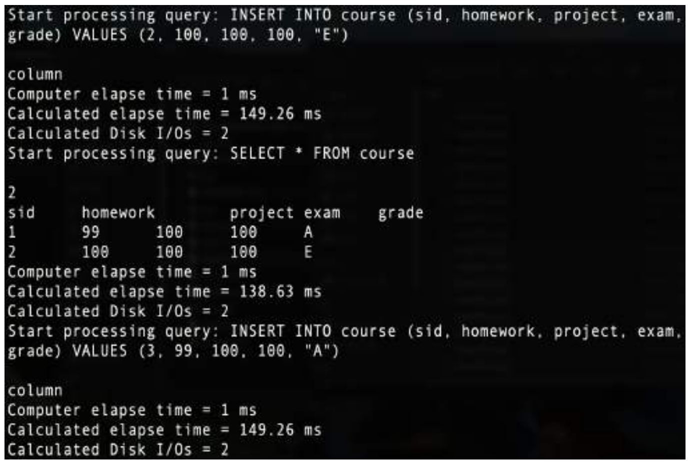
In this project, we develop a Tiny-SQL interpreter that is able to do five basic SQL statements, i.e., CREATE, DROP, SELECT, DELETE, INSERT. We have studied how a database system is implemented and what algorithms we can choose to improve its efficiency. The optimization we implemented is the ordering of the cross join order such that the smaller tables are joined first and the larger tables are joined last. This can save some cost during cross join. Due to the limited time, we are not able to implement some functions, such as DISTINCT and ORDER BY. In sum, we have achieved the basic function of this database system and implemented an optimization technique for multi-table cross join.

7 Appendix.A

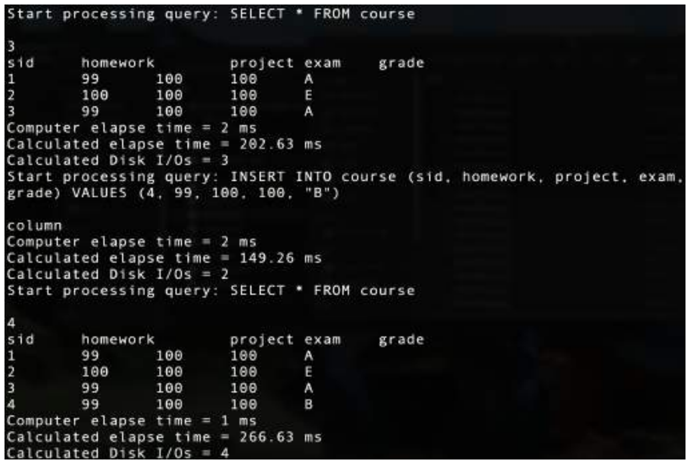
Test results for one table: (the complete results are in 'test.pdf'.)



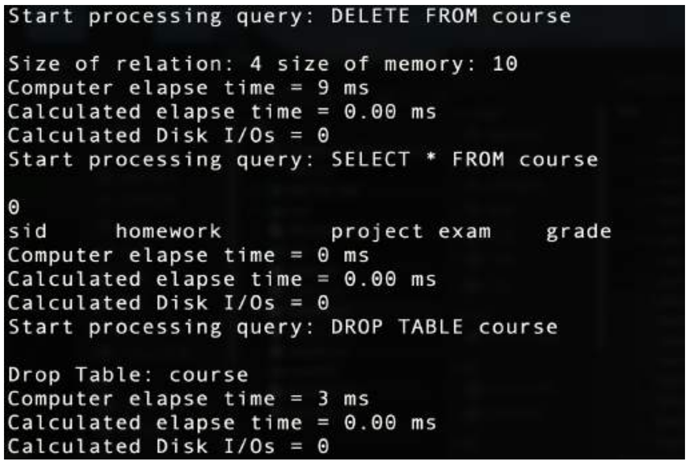
```
Start processing query: CREATE TABLE course (sid INT, homework INT, projec
t INT, exam INT, grade STR20)

Computer elapse time = 24 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (1, 99, 100, 100, "A")

column
Computer elapse time = 4 ms
Calculated elapse time = 74.63 ms
Calculated Disk I/Os = 1
Start processing query: SELECT * FROM course

1
sid     homework        project exam    grade
1       99      100     100     A
Computer elapse time = 40 ms
Calculated elapse time = 74.63 ms
Calculated Disk I/Os = 1
```



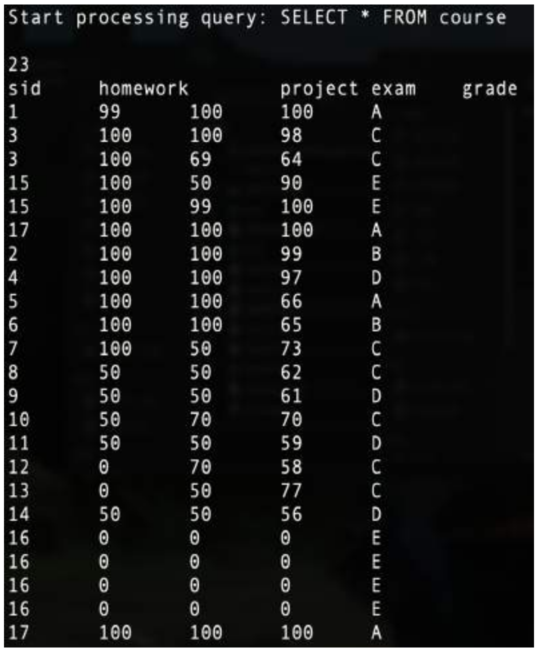
```
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (2, 100, 100, 100, "E")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: SELECT * FROM course

2
sid     homework        project exam    grade
1       99      100     100     A
2       100     100     100     E
Computer elapse time = 1 ms
Calculated elapse time = 138.63 ms
Calculated Disk I/Os = 2
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (3, 99, 100, 100, "A")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
```



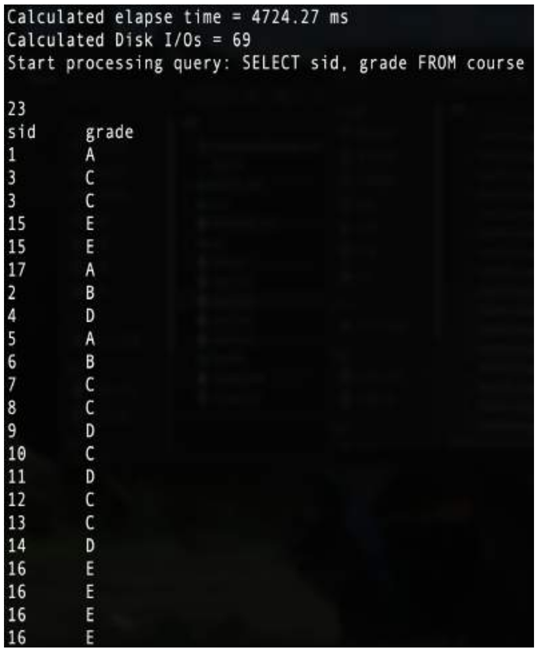
```
Start processing query: SELECT * FROM course

3
sid     homework        project exam    grade
1       99      100     100     A
2       100     100     100     E
3       99      100     100     A
Computer elapse time = 2 ms
Calculated elapse time = 202.63 ms
Calculated Disk I/Os = 3
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (4, 99, 100, 100, "B")

column
Computer elapse time = 2 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: SELECT * FROM course

4
sid     homework        project exam    grade
1       99      100     100     A
2       100     100     100     E
3       99      100     100     A
4       99      100     100     B
Computer elapse time = 1 ms
Calculated elapse time = 266.63 ms
Calculated Disk I/Os = 4
```



```
Start processing query: DELETE FROM course

Size of relation: 4 size of memory: 10
Computer elapse time = 9 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: SELECT * FROM course

0
sid     homework        project exam    grade
Computer elapse time = 0 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: DROP TABLE course

Drop Table: course
Computer elapse time = 3 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
```



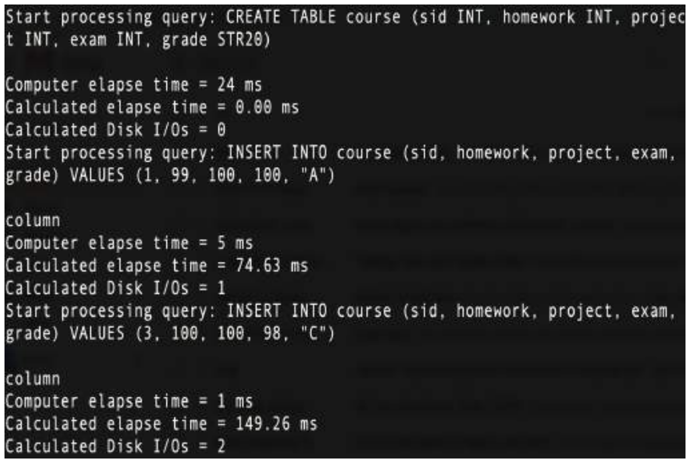
```
Start processing query: SELECT * FROM course

23
sid     homework        project exam    grade
1       99      100     100     A
3       100     100     98      C
3       100     69      64      C
15      100     50      90      E
15      100     99      100     E
17      100     100     100     A
2       100     100     99      B
4       100     100     97      D
5       100     100     66      A
6       100     100     65      B
7       100     50      73      C
8       50      50      62      C
9       50      50      61      D
10      50      70      70      C
11      50      50      59      D
12      0       70      58      C
13      0       50      77      C
14      50      50      56      D
16      0       0       0       E
16      0       0       0       E
16      0       0       0       E
16      0       0       0       E
17      100     100     100     A
```



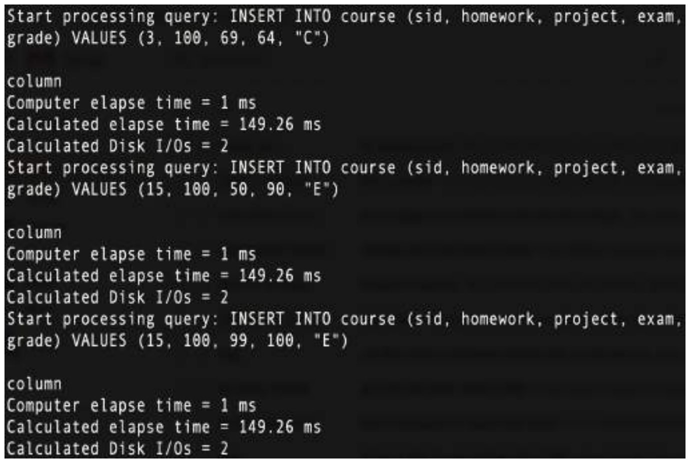
```
Calculated elapse time = 4724.27 ms
Calculated Disk I/Os = 69
Start processing query: SELECT sid, grade FROM course

23
sid     grade
1       A
3       C
3       C
15      E
15      E
17      A
2       B
4       D
5       A
6       B
7       C
8       C
9       D
10      C
11      D
12      C
13      C
14      D
16      E
16      E
16      E
16      E
```

8 Appendix.B

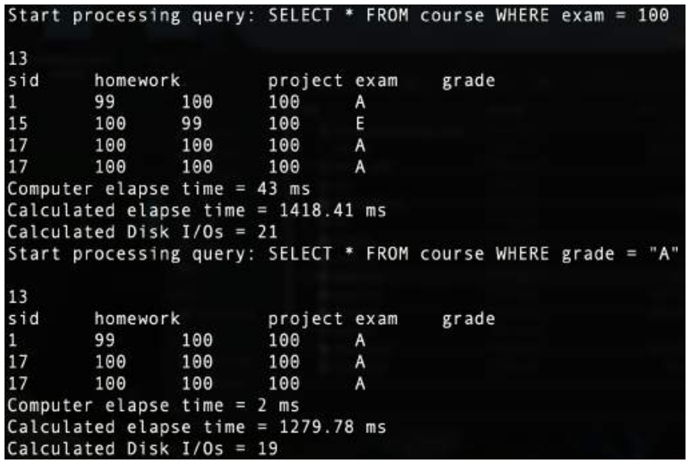
Test results for WHERE condition: (the complete results are in 'test.pdf'.)



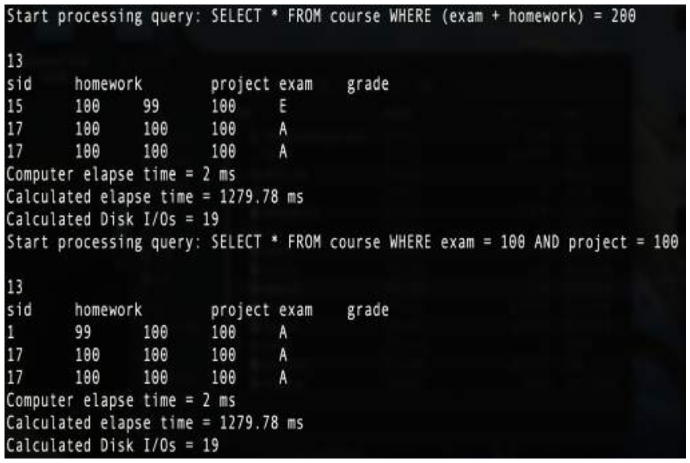
```
Start processing query: CREATE TABLE course (sid INT, homework INT, projec
t INT, exam INT, grade STR20)

Computer elapse time = 24 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (1, 99, 100, 100, "A")

column
Computer elapse time = 5 ms
Calculated elapse time = 74.63 ms
Calculated Disk I/Os = 1
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (3, 100, 100, 98, "C")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
```



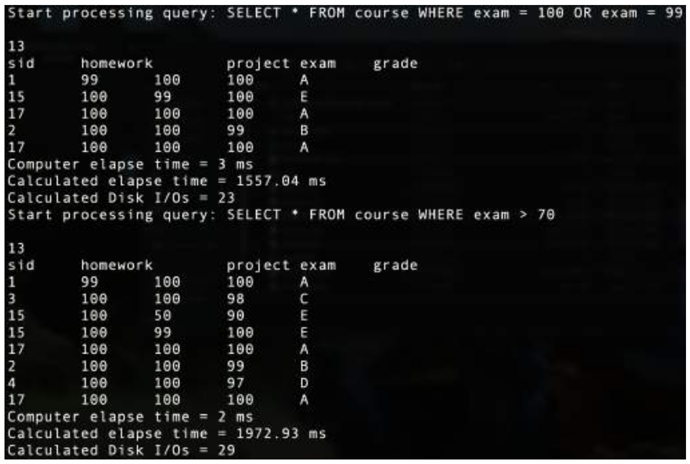
```
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (3, 100, 69, 64, "C")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (15, 100, 50, 90, "E")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: INSERT INTO course (sid, homework, project, exam,
grade) VALUES (15, 100, 99, 100, "E")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
```



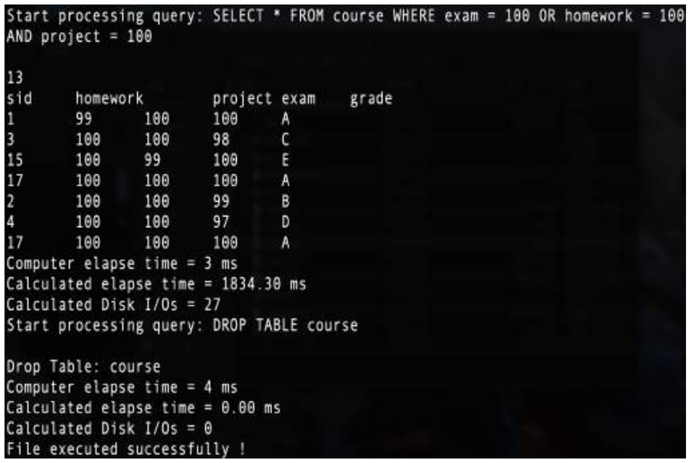
```
Start processing query: SELECT * FROM course WHERE exam = 100

13
sid     homework        project exam    grade
1       99      100     100     A
15      100     99      100     E
17      100     100     100     A
17      100     100     100     A
Computer elapse time = 43 ms
Calculated elapse time = 1418.41 ms
Calculated Disk I/Os = 21
Start processing query: SELECT * FROM course WHERE grade = "A"

13
sid     homework        project exam    grade
1       99      100     100     A
17      100     100     100     A
17      100     100     100     A
Computer elapse time = 2 ms
Calculated elapse time = 1279.78 ms
Calculated Disk I/Os = 19
```



```
Start processing query: SELECT * FROM course WHERE (exam + homework) = 200

13
sid     homework        project exam    grade
15      100     99      100     E
17      100     100     100     A
17      100     100     100     A
Computer elapse time = 2 ms
Calculated elapse time = 1279.78 ms
Calculated Disk I/Os = 19
Start processing query: SELECT * FROM course WHERE exam = 100 AND project = 100

13
sid     homework        project exam    grade
1       99      100     100     A
17      100     100     100     A
17      100     100     100     A
Computer elapse time = 2 ms
Calculated elapse time = 1279.78 ms
Calculated Disk I/Os = 19
```



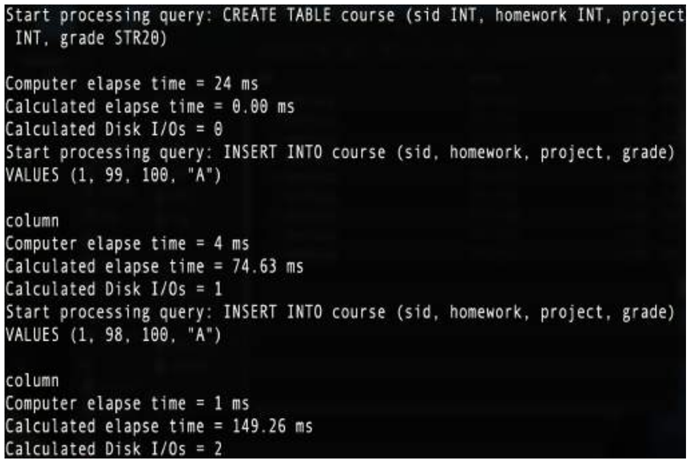
```
Start processing query: SELECT * FROM course WHERE exam = 100 OR exam = 99

13
sid     homework        project exam    grade
1       99      100     100     A
15      100     99      100     E
17      100     100     100     A
2       100     100     99      B
17      100     100     100     A
Computer elapse time = 3 ms
Calculated elapse time = 1557.04 ms
Calculated Disk I/Os = 23
Start processing query: SELECT * FROM course WHERE exam > 70

13
sid     homework        project exam    grade
1       99      100     100     A
3       100     100     98      C
15      100     50      90      E
15      100     99      100     E
17      100     100     100     A
2       100     100     99      B
4       100     100     97      D
17      100     100     100     A
Computer elapse time = 2 ms
Calculated elapse time = 1972.93 ms
Calculated Disk I/Os = 29
```



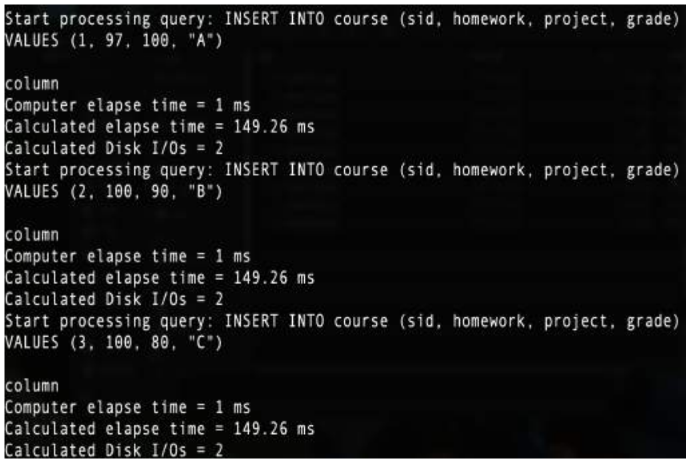
```
Start processing query: SELECT * FROM course WHERE exam = 100 OR homework = 100
AND project = 100

13
sid     homework        project exam    grade
1       99      100     100     A
3       100     100     98      C
15      100     99      100     E
17      100     100     100     A
2       100     100     99      B
4       100     100     97      D
17      100     100     100     A
Computer elapse time = 3 ms
Calculated elapse time = 1834.30 ms
Calculated Disk I/Os = 27
Start processing query: DROP TABLE course

Drop Table: course
Computer elapse time = 4 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
File executed successfully !
```

9 Appendix.C

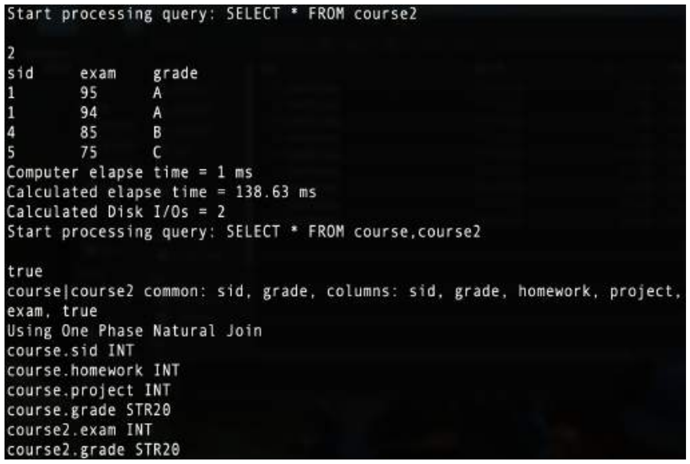
Test results for two-table join: (the complete results are in 'test.pdf'.)



```
Start processing query: CREATE TABLE course (sid INT, homework INT, project
 INT, grade STR20)

Computer elapse time = 24 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: INSERT INTO course (sid, homework, project, grade)
VALUES (1, 99, 100, "A")

column
Computer elapse time = 4 ms
Calculated elapse time = 74.63 ms
Calculated Disk I/Os = 1
Start processing query: INSERT INTO course (sid, homework, project, grade)
VALUES (1, 98, 100, "A")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
```



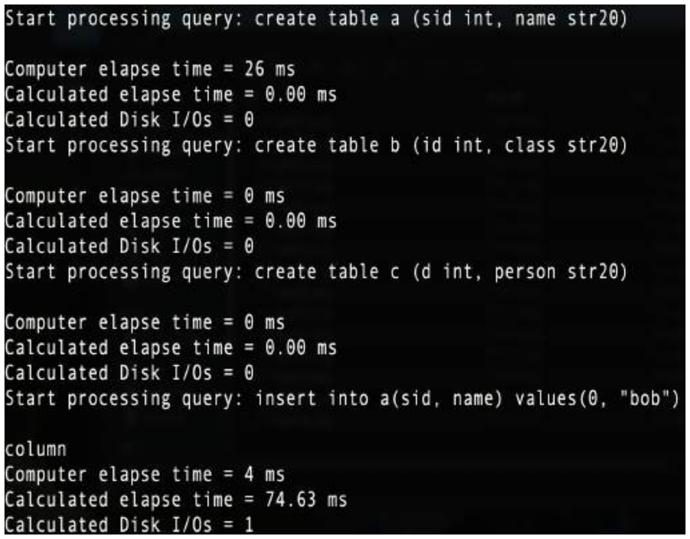
```
Start processing query: INSERT INTO course (sid, homework, project, grade)
VALUES (1, 97, 100, "A")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: INSERT INTO course (sid, homework, project, grade)
VALUES (2, 100, 90, "B")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: INSERT INTO course (sid, homework, project, grade)
VALUES (3, 100, 80, "C")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
```



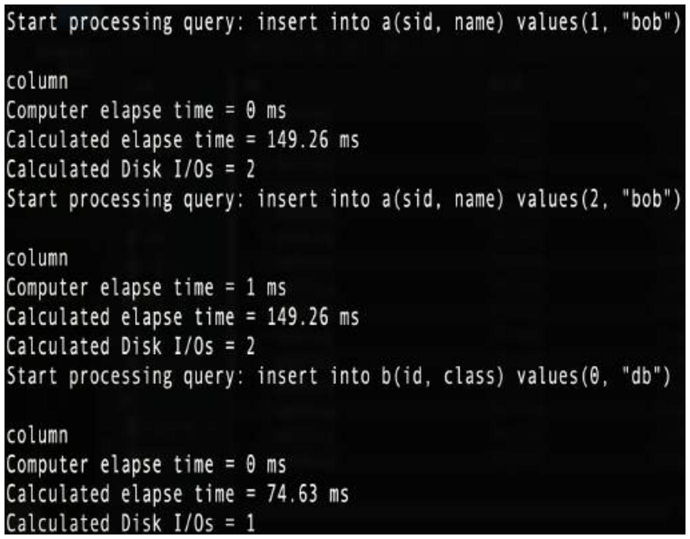
```
Start processing query: SELECT * FROM course2

2
sid     exam    grade
1       95      A
1       94      A
4       85      B
5       75      C
Computer elapse time = 1 ms
Calculated elapse time = 138.63 ms
Calculated Disk I/Os = 2
Start processing query: SELECT * FROM course,course2

true
course|course2 common: sid, grade, columns: sid, grade, homework, project,
exam, true
Using One Phase Natural Join
course.sid INT
course.homework INT
course.project INT
course.grade STR20
course2.exam INT
course2.grade STR20
```

10 Appendix.D

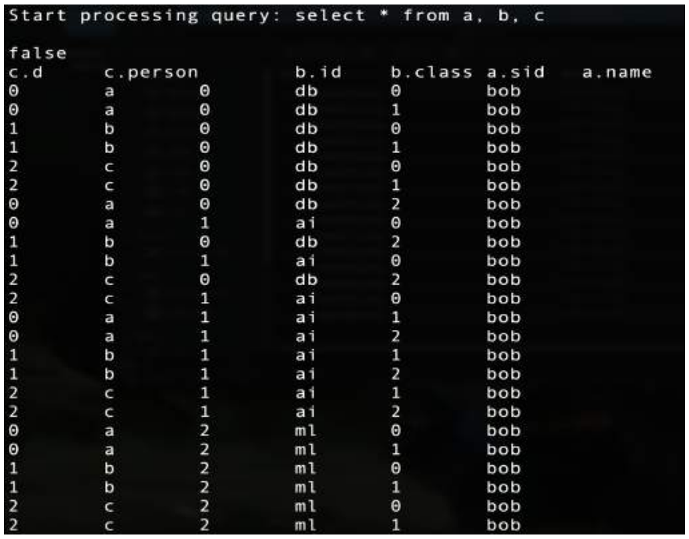
Test results for multiple-table join: (the complete results are in 'test.pdf'.)



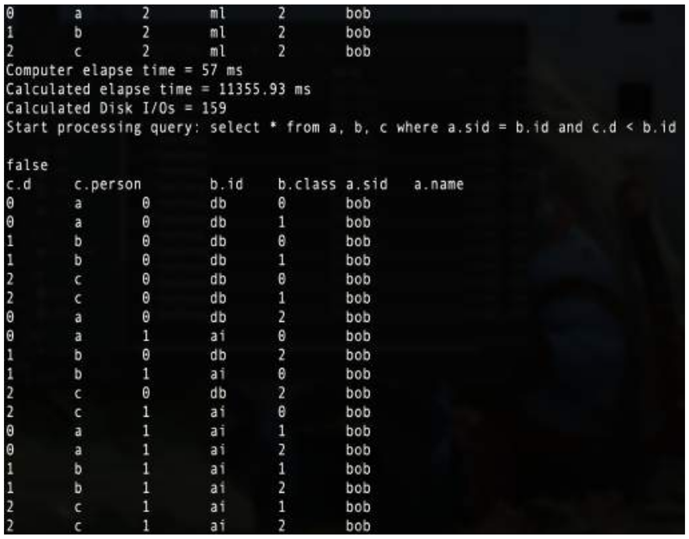
```
Start processing query: create table a (sid int, name str20)

Computer elapse time = 26 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: create table b (id int, class str20)

Computer elapse time = 0 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: create table c (d int, person str20)

Computer elapse time = 0 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: insert into a(sid, name) values(0, "bob")

column
Computer elapse time = 4 ms
Calculated elapse time = 74.63 ms
Calculated Disk I/Os = 1
```



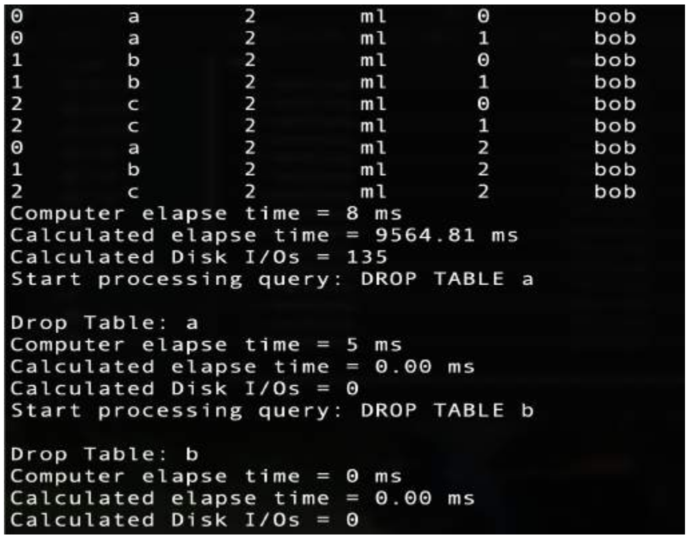
```
Start processing query: insert into a(sid, name) values(1, "bob")

column
Computer elapse time = 0 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: insert into a(sid, name) values(2, "bob")

column
Computer elapse time = 1 ms
Calculated elapse time = 149.26 ms
Calculated Disk I/Os = 2
Start processing query: insert into b(id, class) values(0, "db")

column
Computer elapse time = 0 ms
Calculated elapse time = 74.63 ms
Calculated Disk I/Os = 1
```



```
Start processing query: select * from a, b, c

false
c.d     c.person        b.id    b.class a.sid   a.name
0       a       0       db      0       bob
0       a       0       db      1       bob
1       b       0       db      0       bob
1       b       0       db      1       bob
2       c       0       db      0       bob
2       c       0       db      1       bob
0       a       0       db      2       bob
0       a       1       ai      0       bob
1       b       0       db      2       bob
1       b       1       ai      0       bob
2       c       0       db      2       bob
2       c       1       ai      0       bob
0       a       1       ai      1       bob
0       a       1       ai      2       bob
1       b       1       ai      1       bob
1       b       1       ai      2       bob
2       c       1       ai      1       bob
2       c       1       ai      2       bob
0       a       2       ml      0       bob
0       a       2       ml      1       bob
1       b       2       ml      0       bob
1       b       2       ml      1       bob
2       c       2       ml      0       bob
2       c       2       ml      1       bob
```



```
0       a       2       ml      2       bob
1       b       2       ml      2       bob
2       c       2       ml      2       bob
Computer elapse time = 57 ms
Calculated elapse time = 11355.93 ms
Calculated Disk I/Os = 159
Start processing query: select * from a, b, c where a.sid = b.id and c.d < b.id

false
c.d     c.person        b.id    b.class a.sid   a.name
0       a       0       db      0       bob
0       a       0       db      1       bob
1       b       0       db      0       bob
1       b       0       db      1       bob
2       c       0       db      0       bob
2       c       0       db      1       bob
0       a       0       db      2       bob
0       a       1       ai      0       bob
1       b       0       db      2       bob
1       b       1       ai      0       bob
2       c       0       db      2       bob
2       c       1       ai      0       bob
0       a       1       ai      1       bob
0       a       1       ai      2       bob
1       b       1       ai      1       bob
1       b       1       ai      2       bob
2       c       1       ai      1       bob
2       c       1       ai      2       bob
```



```
0       a       2       ml      0       bob
0       a       2       ml      1       bob
1       b       2       ml      0       bob
1       b       2       ml      1       bob
2       c       2       ml      0       bob
2       c       2       ml      1       bob
0       a       2       ml      2       bob
1       b       2       ml      2       bob
2       c       2       ml      2       bob
Computer elapse time = 8 ms
Calculated elapse time = 9564.81 ms
Calculated Disk I/Os = 135
Start processing query: DROP TABLE a

Drop Table: a
Computer elapse time = 5 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
Start processing query: DROP TABLE b

Drop Table: b
Computer elapse time = 0 ms
Calculated elapse time = 0.00 ms
Calculated Disk I/Os = 0
```