Bowen Lan **Qing Cao**
UIN: 227001476 **UIN: 127004024**

CSCE-608 Database Systems

Fall 2018

Instructor: Dr. Jianer Chen
Office: HRBB 315C
Phone: 845-4259
Email: chen@cse.tamu.edu
Office Hours: MWF 10:00am-11:00am

Grader: Sambartika Guha
Email: sambartika.guha@tamu.edu

COURSE PROJECT #2

Description.

This course project is to design and implement a simple SQL (called Tiny-SQL) interpreter. The interpreter should accept SQL queries that are valid in terms of Tiny-SQL grammar, execute queries, and output results, which is written in Java.

In the following report, we will discuss:

- System architecture of this database management system is introduced, such as interface, parser, join and sort operation.
- Optimization techniques applied to accelerate the program.
- Experiments and results demonstrating all the queries the interpreter is capable of executing and the performance of the interpreter.

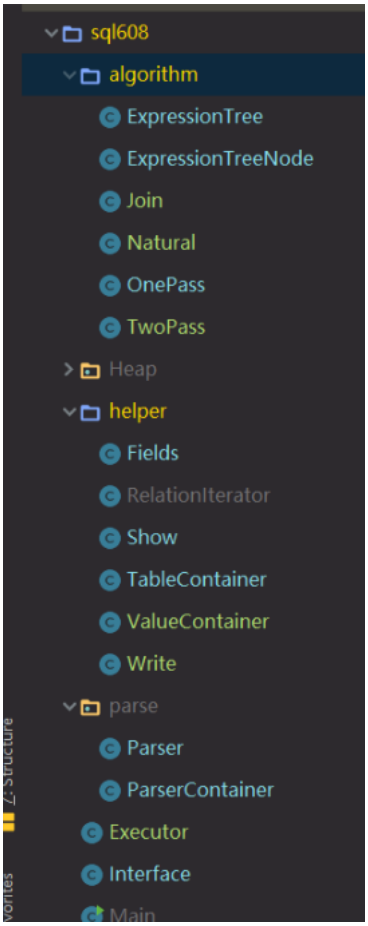
System architecture.

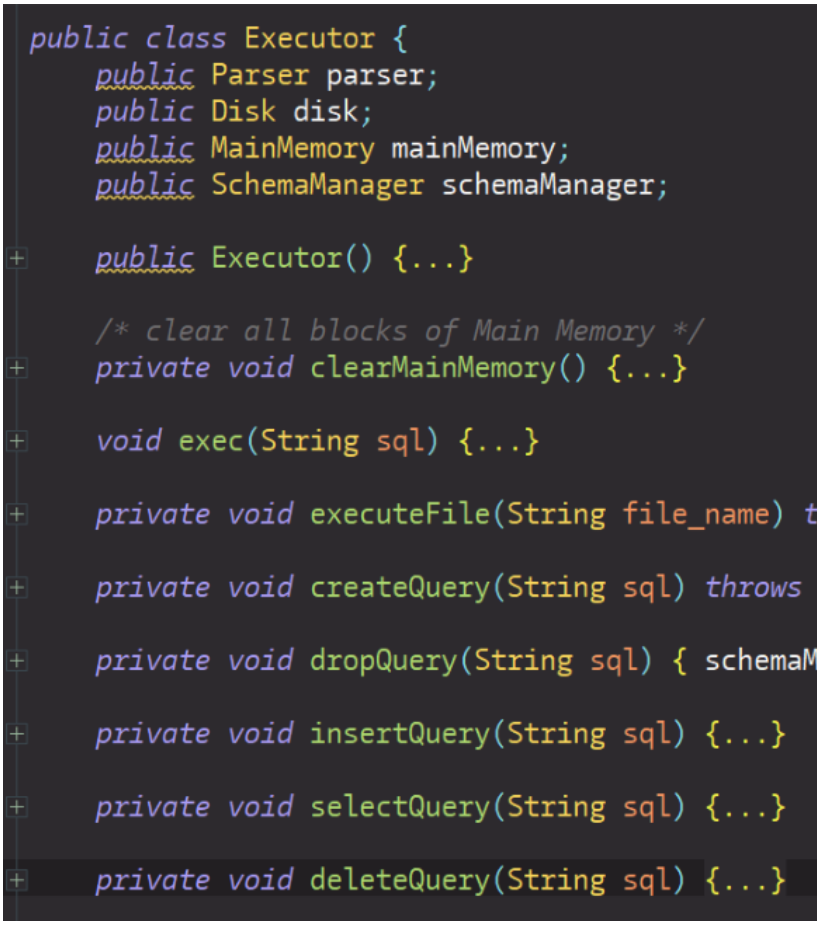
- User Interface & General architecture

  The interface class is able to read a file from its absolute position like the test file in the storageManager folder and able to output the query results to the interface.

  The code arrangement is showed below. The Executor class is to direct different queries to different classes and methods based on first element of parsing results. Disk, MainMemory, SchemaManager will be initialized here. It is simple for CREATE and DROP based on the schema manager to create or delete relation. For INSERT, the value is used to build a tuple and append to the relation. Also there may be values from SELECT. DELETE is to read the disk block and invalidate the tuple, all the fields becoming null and then write the block back to the disk. For SELECT, it will use sort and join methods under algorithm package. One pass algorithm is first considered for the single table case in OnePass class.

When it comes to multiple tables, two pass merge sort and join methods are added





in TwoPass class. Under algorithm package, there are other classes like Natural for natural join and Join for cross join, and they will use the sort and join method under OnePass and TwoPass classes. There is Heap package used for sorting tuples. Helper package contains useful data structure and Show class for showing tuples to the interface, writing class for append tuple and output to disk.

- Parser

We start by doing the Parser without making a tree but just store the different useful part trimmed from sql command into a container class. Generally, the Parser includes two parts. First, a container we called ParserContainer which stores useful flags about whether there is a keyword like DISTINCT, ORDER BY, or multiple tables in the sql command, and also stores the parameter from the sql command like table.attributes, conditions after WHERE. Second, there is a Parser class which has different parsing methods corresponding to different types of commands like ‘SELECT’, ‘DROP’, ‘CREATE’, ‘INSERT’, ‘DELETE’.

We use two other containers located in helper package. One called ValueContainer which are used for INSERT with SELECT results as multiple values. Second is TableContainer which helps the parser methods to return multiple useful information about the table for future execution. Normally it will contain the table name, its fields name and the FIELD. This is mainly used in CREATE and INSERT.

- Condition Check by Expression Tree

  If there is a WHERE key word, we use the 'expression-tree' to store the precedence information. The priority of operators can be seen in the form below. We found there are [ ], ( ) in the command, although they are not required in the tinysql grammar, we implement them in our code with precedence to be lowest. When constructing the tree, it will be done recursively by two stacks. If meeting a new operator, the last two operands will be popped out and construct a binary node. Left parentheses will be pushed in waiting for the right parentheses. If meeting the right parentheses, it will continue construct the tree until meeting the left parentheses. Then left parentheses will be popped out individually and check if there is left operand and continue construct.
  This tree will be used to check if tuple followed the condition followed WHERE keyword also recursively following the tree.

priority	operators
3	/ , *
2	+ , - , > , <
1	=
0	&
-1	|
-2	Parentheses ( )

- Sort and Join implementation

When we have 'ORDER BY' or 'DISTINCT' clause in the query, if the relation size is larger than main memory, we use two pass merge sort algorithm to sort the table in the disk. If the table size is less than the main memory, we can take all tuples from disk to main memory and one pass sort them according to the attributes. To be mentioned, after sorting the relation, it is easy for the Show class to show the tuples to the interface non-repeatedly by simply comparing current with previous one.

In multiple tables case, we have to join tables, two techniques offered are cross join and natural join.

Cross join	Natural join
Do one-pass if one of the tables can be put into memory directly Do two-pass if both tables are too large	Join two tables in the specific field, sort tuples in that filed, compare and join tuples from two tables.

- Optimizations when joining multiple tables

Three basic optimizations are implemented in such order. First check if there is any selection operation in the condition separated by AND. If connected by OR, it is not feasible logically. If there is such selection condition, selection on single table will be done first. Then this selection condition will be eliminated from ParserContainer. For example, if we have condition like "course.exam = 100", this will be used to generate a new tempcourse relation and substitute its original course relation in the ParserContainer which means further operation will be directly operated on this tempcourse relation.

Second, after doing the selection, condition will be examined to see if it can do natural join, condition like course.sid = course2.sid is preferred. Same as selection, doing natural join will generate a temporary relation like coursenaturalJoincourse2.

After doing all these, if there are still more than two tables left, then considered doing the cross join. If more than two tables, join order is considered as doing the join on two smaller tables first because the final size of joined table is determined

and smaller joined table size means smaller cost for next join. Based on this sort, total cost of certain order can be calculated from product and sum, then check this cost for all possible order of several table to find out the best order. Cross join for two tables could be one pass or two pass depending on the the small relation block size compared to the main memory size, if smaller, we can do one-pass which store the smaller in the main memory and loop over blocks of the other relation. If bigger, we bring 8 blocks of one of the tables into memory, use one block of memory to loop over blocks of the other table. Two tuples from them will be joined in the remaining one memory block and write back to disk.

You can see the code follows this sequence to do the SELECT on multiple tables. selectFirst method is for doing the selection first if possible. Then check if there is expression tree node satisfy the natural join condition. Use set to record the table which has been natural joined and use the joined table to substitute the original table. Finally do the cross join on tables, the join order will be checked in the Join.crossJoinTables method.

```
private void selectFromMultipleTables(ParserContainer parserContainer) {
    ExpressionTree expressionTree = new ExpressionTree(parserContainer.getConditions(),

    /*
    consider push down the row selection in the conditions
    delete those satisfied conditions in the parser container
    satisfied condition e.g. course.homework = 100
    and generate a new tempcourse table
    */
    if (parserContainer.isWhere()) selectFirst(expressionTree, parserContainer);

    ArrayList<String> attributes = parserContainer.getAttributes();
    ArrayList<String> tableLists = parserContainer.getTables();  // reference

    /*
    SELECT DISTINCT table.attr FROM tables
    distinct means we need to do the sorting on the attr
    from table.attrs get related tables
    store tables and its corresponding field names
    */
    Map<String, ArrayList<String>> tableToAttr = new HashMap<>();
    if (parserContainer.isDistinct()) {...}

    String joinedTableName;
    if (parserContainer.isWhere()) {
        /* First check if can do the Natural Join */
```

Bowen Lan **Qing Cao**
UIN: 227001476 **UIN: 127004024**

```
    ArrayList<ExpressionTreeNode> normalNodes = new ArrayList<>();
    ArrayList<ExpressionTreeNode> naturalJoinNodes = new ArrayList<>();
    ExpressionTree expressionTree2 = new ExpressionTree(parserContainer.getConditions(), relationNa
    ArrayList<ExpressionTreeNode> subNodes = ExpressionTree.getSubTreeNodes(expressionTree2).nodeN
    //...
    for (ExpressionTreeNode node : subNodes) {...}
    //...
    HashSet<String> joinedTableSet = new HashSet<>();
    // get undergoing tables (tmp) after select push down
    ArrayList<String> newTableList = parserContainer.getTables();
    for (ExpressionTreeNode naturalJoinNode : naturalJoinNodes) {...}

    // set tables like r naturalJoin t naturalJoin s
    parserContainer.setTables(newTableList);
    if(normalNodes.size() == 0) {
        parserContainer.setWhere(false);
        parserContainer.setConditions(null);
    } else {
        ExpressionTreeNode remainConditions = mergeNodes(normalNodes);
        parserContainer.setConditions(remainConditions.getString(remainConditions));
    }

    /* Cross Join */
    joinedTableName = Join.crossJoinTables(tableLists, parserContainer.isDistinct(), tableToAttr,
            mainMemory, schemaManager);
}
else {
    /* no WHERE do cross join directly */
    joinedTableName = Join.crossJoinTables(tableLists, parserContainer.isDistinct(), tableToAttr,
            mainMemory, schemaManager);
```

- Some complicated given query results for course, course2:

```
-----------------------------------------------------
SELECT DISTINCT course.grade, course2.grade FROM course, course2 WHERE course.
sid = course2.sid AND [ course.exam > course2.exam OR course.grade = "A" AND
course2.grade = "A" ] ORDER BY course.exam
course.grade course2.grade
course course2

A   A
--------------------------
1 rows of results
Execution time: 51.763seconds
Disk IO taken: 690
```

Bowen Lan **Qing Cao**
UIN: 227001476 **UIN: 127004024**

```
INSERT INTO course (sid, homework, project, exam, grade) SELECT * FROM course
*
-----------------------------------------------------
SELECT * FROM course
*
sid homework    project exam    grade
1   99  100 100 A
2   NULL    100 100 E
3   100 100 100 E
1   99  100 100 A
2   NULL    100 100 E
3   100 100 100 E
---------------------------
6 rows of results
Execution time: 0.449seconds
Disk IO taken: 6
```

```
-----------------------------------------------------
SELECT * FROM course, course2 WHERE course.sid = course2.sid AND [ course.exam
 = 100 OR course2.exam = 100 ]
*
course course2

1   99  100 100 A   1   100 A
17  100 100 100 A   17  0   A
17  100 100 100 A   17  0   A
---------------------------
3 rows of results
Execution time: 37.103seconds
Disk IO taken: 496
```

```
-----------------------------------------------------
SELECT * FROM course WHERE ( exam * 30 + homework * 20 + project * 50 ) / 100
 = 100
*
sid homework    project exam    grade
17  100 100 100 A
17  100 100 100 A
---------------------------
2 rows of results
Execution time: 4.42seconds
Disk IO taken: 58
```

Bowen Lan **Qing Cao**
UIN: 227001476 **UIN: 127004024**

```
-----------------------------------------------------
SELECT * FROM course WHERE exam + homework = 200
*
sid homework    project exam    grade
15  100 99  100 E
17  100 100 100 A
17  100 100 100 A
---------------------------
3 rows of results
Execution time: 4.339seconds
Disk IO taken: 58
```

- Test query main contains three types of tables. First is course, second is rst, third is t123456. We will compare the two results of each of them when doing natural join or comment out natural join functions which means using cross join. The queries we choose are the simple ones because there is only the simplest case for 6 table join:

  SELECT * FROM course, course2 WHERE course.sid = course2.sid

  SELECT * FROM r, s, t WHERE r.a=t.a AND r.b=s.b AND s.c=t.c

  SELECT * FROM t1, t2, t3, t4, t5, t6

Using natural join:

	Disk IO	Execution Time
course	496	37s
rst	392	29s

Using cross join

	Disk IO	Execution Time
course, course2	4767	357s
rst	21180	1589s
t123456	965	72s

It is very clear that natural join helps a lot when satisfying the condition. Cross join on three tables is already taken like forever but natural join them cost even lower disk IO and time.

- Experiments on Table with 200 tuples

  We use a 200-tuple 3-field table from project 1 as dataset to test our program. The queries and performance are shown as below.

  CREATE TABLE Customer (id INT, age INT, name STR20)
  Insert into Customer (id, age, name) values (1, 46, 'Cully');
  insert into Customer (id, age, name) values (2, 77, 'Mohandis');
  insert into Customer (id, age, name) values (3, 15, 'Kass');
  insert into Customer (id, age, name) values (4, 94, 'Gabby');
  insert into Customer (id, age, name) values (5, 28, 'Roth');
  insert into Customer (id, age, name) values (6, 65, 'Kalinda');
  insert into Customer (id, age, name) values (7, 6, 'Edgar');
  insert into Customer (id, age, name) values (8, 31, 'Hilton');
  insert into Customer (id, age, name) values (9, 73, 'Leonore');
  insert into Customer (id, age, name) values (10, 55, 'Wynnie');
  insert into Customer (id, age, name) values (11, 93, 'Tanya');
  insert into Customer (id, age, name) values (12, 37, 'Rycca');
  insert into Customer (id, age, name) values (13, 88, 'Petronilla');
  insert into Customer (id, age, name) values (14, 84, 'Jacinta');
  insert into Customer (id, age, name) values (15, 48, 'Ricca');
  insert into Customer (id, age, name) values (16, 3, 'Prisca');
  insert into Customer (id, age, name) values (17, 18, 'Frazier');
  insert into Customer (id, age, name) values (18, 81, 'Mora');
  ……

Bowen Lan **Qing Cao**
UIN: 227001476 **UIN: 127004024**

```
--------------------------
SELECT * FROM Customer
*
id  age name
1   46  'cully'
2   77  'mohandis'
3   15  'kass'
4   94  'gabby'
5   28  'roth'
6   65  'kalinda'
7   6   'edgar'
8   31  'hilton'
9   73  'leonore'
10  55  'wynnie'
11  93  'tanya'
12  37  'rycca'
13  88  'petronilla'
14  84  'jacinta'
15  48  'ricca'
16  3   'prisca'
17  18  'frazier'
18  81  'mora'
19  9   'skipp'
20  76  'darda'
21  6   'anton'
22  75  'farrand'
23  98  'jamie'
24  63  'cinderella'
--------------------------
24 rows of results
Execution time: 0.899seconds
Disk IO taken: 12
```

```
-----------------------------------
SELECT * FROM Customer ORDER BY age
*
id  age name
16  3   'prisca'
7   6   'edgar'
21  6   'anton'
19  9   'skipp'
3   15  'kass'
17  18  'frazier'
5   28  'roth'
8   31  'hilton'
12  37  'rycca'
1   46  'cully'
15  48  'ricca'
10  55  'wynnie'
24  63  'cinderella'
6   65  'kalinda'
9   73  'leonore'
22  75  'farrand'
20  76  'darda'
2   77  'mohandis'
18  81  'mora'
14  84  'jacinta'
13  88  'petronilla'
11  93  'tanya'
4   94  'gabby'
23  98  'jamie'
--------------------------
24 rows of results
Execution time: 5.142seconds
Disk IO taken: 68
```

We compare the performances of these two commands on different size of table.

A: SELECT * FROM Customer

B: SELECT * FROM Customer ORDER BY age

Size	Disk I/O (A/B)	Execution Time (A/B)

200	100/500	7.48/37
150	75/380	5.617/28.5
100	50/250	3.745/18.76
50	25/130	1.873/9.787
20	10/10	0.75/0.8
35	18/91	1.35/6.9
25	12/68	0.89/5.14

From table above, we can know that when the size of table larger than 20, command B changing from the one-pass to the two-pass.

Reference:

- https://github.com/Stephen2526/CSCE-608-Project-2-DBMS
- https://github.com/melonskin/tinySQL

Since we have no past large project experience in coding in Java or C++ (both of team members are not from CS department ☹), we have read and learnt these two codes on GitHub, during the session, we have referred to them as well. There are lots of comments in our code showing the process.