Project Report: Database System Implementation (COP6726)

Project 5: Putting it all together

Name: Vikas Chaubey**, UFID:** 3511 5826**, Email:** vikas.chaubey@ufl.edu

I have done this project alone without any project partner. Hence, I am submitting it on my behalf.

1) **Compile and Run Instructions:** This program has been compiled, run and tested on Ubuntu 18.04 operating system. All the table files are present in "tpch" folder, and all the bin heap files are present in "bin" folder present in the main folder directory.

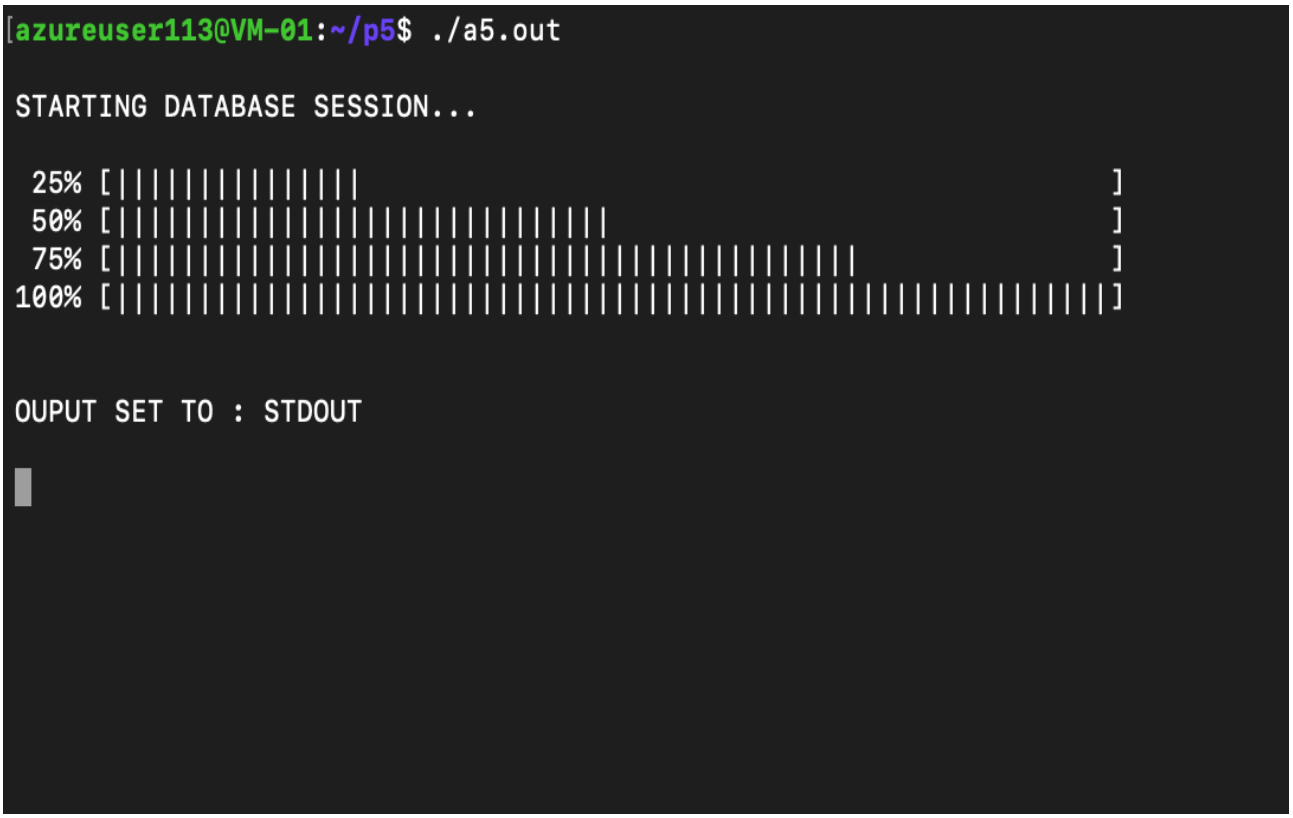
   Step.1 - To run a5.out: In order to make a5.out please run below commands.

   > make clean
   > make a5.out
   (Once a5.out is made successfully then run **a5.out** with below command)
   > ./a5.out

   Steps to run the test queries:

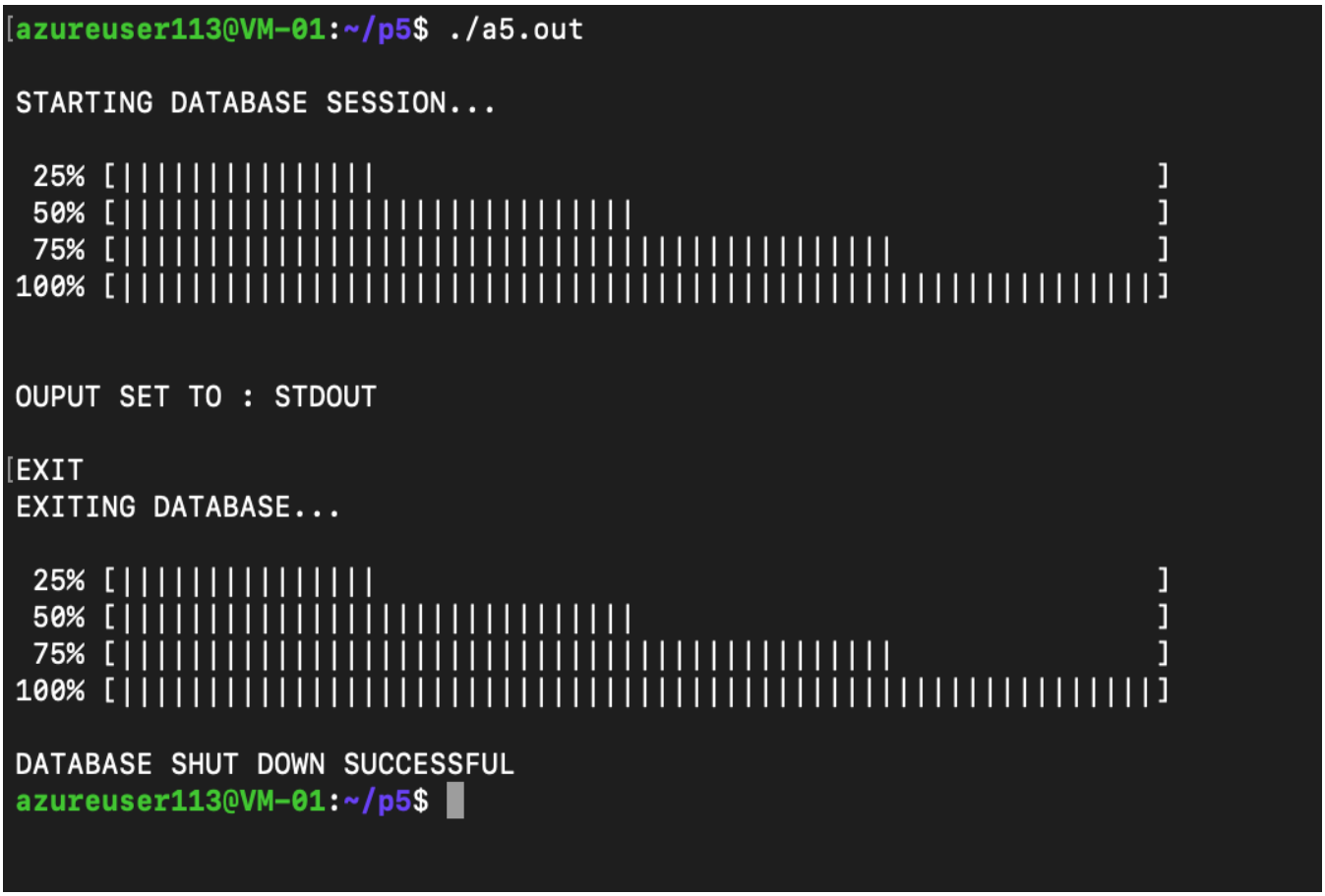
   1) When the a5.out file is run then SQL console appears using which we can run SQL test queries.

      SQL console window looks like as shown below on running a5.out.



   2) In step 2 enter the query, and then press "ENTER" and then press "CTRL D".

3) In order to close the database console, type “EXIT”, then press “ENTER” and then press “CTRL D”.



Step.2 - To run gtest (google tests):

In order to run **gtest** please use below commands sequentially to run googletests. In order to run gtests the queries which were given in project 4 part 2 has to be piped with the executable file. Tc1.sql query file is included in the main directory.

> make clean
> make gtest.out
> ./gtest < tc1.sql

2) Project files, structure, classes and functions:

1) **QueryTreeNode.h File:** Execution tree node is represented by this file , different classes present in this file provide the basic structure for the execution tree node. The class defines various nodes in form of classes which are explained below:

   a) **The Base Node (RelOpNode Class):** RelOpNode class represents the base node. This base node is inherited by all other nodes in the file. RelOpNode class has been modified to have two more methods, these methods are "ExecuteNode()" and "Wait()".

```
class RelOpNode {
public:
    int pid;  // Pipe ID
    RelOpType t;
    Schema schema;  // Ouput Schema
    RelationalOp *relOp;
    RelOpNode ();
    RelOpNode (RelOpType type) : t (type) {}
    ~RelOpNode () {}
    virtual void PrintNode () {};
    virtual void ExecuteNode (unordered_map<int, Pipe *> &pipeMap){};
    virtual void Wait () {
        relOp->WaitUntilDone ();
    }
};
```

1) **virtual void ExecuteNode (unordered_map<int, Pipe *> &pipeMap){};**

   Description: This function is newly added to the base class RelOpNode, which is inherited by all the node classes, the ExecuteNode() function defines the new instance of relational operator type for each of the inheriting subclass , all these nodes implement this method. This method creates the new instance of relational operator as well as run it. This function ensures that previous node's execute function executes

2) **virtual void Wait ()**

   Description: wait function makes the requesting thread wait till the previous node finish the execution and then control is transferred to the next node.

a. **Class JoinOpNode (The Join Node):** JoinOpNode class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class JoinOpNode : public RelOpNode {
public:
    RelOpNode *left;
    RelOpNode *right;
    CNF cnf;
    Record literal;
    JoinOpNode () : RelOpNode (J) {}
    ~JoinOpNode () {
        if (left) delete left;
        if (right) delete right;
    }
    void PrintNode () {
        left->PrintNode ();
        right->PrintNode ();
        cout << "-----------------------------------" << endl;
        cout << " JOIN OPERATION" << endl;
        cout << " Left Input Pipe ID : " << left->pid << endl;
        cout << " Right Input Pipe ID : " << right->pid << endl;
        cout << " Output Pipe ID : " << pid << endl;
        cout << " Output Schema : " << endl;
        schema.Print ();
        cout << " Join CNF : " << endl;
        cnf.PrintWithSchema(&left->schema,&right->schema,&literal);
        cout << "-----------------------------------" << endl;
    }

    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
        pipeMap[pid] = new Pipe (BUFFSIZE);
        relOp = new Join ();
        left->ExecuteNode (pipeMap);
        right->ExecuteNode (pipeMap);
        ((Join *)relOp)->Run (*(pipeMap[left->pid]), *(pipeMap[right->pid]), *(pipeMap[pid]), cnf, literal);
        left->Wait ();
        right->Wait ();
    }
};
```

b. **Class ProjectOpNode (The Join Node): ProjectOpNode** class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class ProjectOpNode : public RelOpNode {
public:
    int numIn;
    int numOut;
    int *attsToKeep;
    RelOpNode *from;
    ProjectOpNode () : RelOpNode (P) {}
    ~ProjectOpNode () {
    if (attsToKeep) delete[] attsToKeep;
    }
    void PrintNode () {
     from->PrintNode ();

            cout << "---------------------------------" << endl;
            cout << " PROJECT OPERATION" << endl;
            cout << " Input Pipe ID : " << from->pid << endl;
            cout << " Output Pipe ID " << pid << endl;
            cout << " Number Attrs Input : " << numIn << endl;
            cout << " Number Attrs Output : " << numOut << endl;
            cout << " Attrs To Keep : [";
            for (int i = 0; i < numOut; i++) {
                cout << attsToKeep[i];
                if (i!=numOut-1){
                    cout<<",";
                }
            }
            cout<< "]"<< endl;
            cout << " Output Schema:" << endl;
            schema.Print ();
            cout << "---------------------------------" << endl;

    }
    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
        pipeMap[pid] = new Pipe (BUFFSIZE);
        relOp = new Project ();
        from->ExecuteNode (pipeMap);
        ((Project *)relOp)->Run (*(pipeMap[from->pid]), *(pipeMap[pid]), attsToKeep, numIn, numOut);
        from->Wait ();
    }
};
```

c. **Class SelectFileOpNode (The Join Node): SelectFileOpNode** class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class SelectFileOpNode : public RelOpNode {
public:
    bool opened;
    CNF cnf;
    DBFile file;
    Record literal;
    SelectFileOpNode () : RelOpNode (SF) {}
    ~SelectFileOpNode () {
        if (opened) {
            file.Close ();
        }
    }
    void PrintNode () {
            cout << "---------------------------------" << endl;
            cout << " SELECT FILE OPERATION" << endl;
            cout << " Output Pipe ID " << pid << endl;
            cout << " Output Schema:" << endl;
            schema.Print ();
            cout << " Select CNF:" << endl;
            cnf.PrintWithSchema(&schema,&schema,&literal);
            cout << "---------------------------------" << endl;

    }
    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
        pipeMap[pid] = new Pipe (BUFFSIZE);
        relOp = new SelectFile ();
        ((SelectFile *)relOp)->Run (file, *(pipeMap[pid]), cnf, literal);
    }
};
```

d. **Class SelectPipeOpNode (The Join Node): SelectPipeOpNode** class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class SelectPipeOpNode : public RelOpNode {
public:
    CNF cnf;
    Record literal;
    RelOpNode *from;
    SelectPipeOpNode () : RelOpNode (SP) {}
    ~SelectPipeOpNode () {
        if (from) delete from;
    }
    void PrintNode () {
        from->PrintNode ();
        cout << "-----------------------------------" << endl;
        cout << " SELECT PIPE OPERATION" << endl;
        cout << " Input Pipe ID : " << from->pid << endl;
        cout << " Output Pipe ID : " << pid << endl;
        cout << " Output Schema:" << endl;
        schema.Print ();
        cout << " Select CNF:" << endl;
        cnf.PrintWithSchema(&schema,&schema,&literal);
        cout << "-----------------------------------" << endl;
    }
    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
        pipeMap[pid] = new Pipe (BUFFSIZE);
        relOp = new SelectPipe ();
        from->ExecuteNode (pipeMap);
        ((SelectPipe *)relOp)->Run (*(pipeMap[from->pid]), *(pipeMap[pid]), cnf, literal);
        from->Wait ();
    }
};
```

e. **Class SumOpNode (The Join Node): SumOpNode** class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class SumOpNode : public RelOpNode {
public:
    Function compute;
    RelOpNode *from;
    SumOpNode () : RelOpNode (S) {}
    ~SumOpNode () {
    if (from) delete from;
    }
    void PrintNode () {

        from->PrintNode ();
        cout << "---------------------------------" << endl;
        cout << " SUM OPERATION" << endl;
        cout << " Input Pipe ID : " << from->pid << endl;
        cout << " Output Pipe ID : " << pid << endl;
        cout << " Function :" << endl;
        compute.Print (&from->schema);
        cout << " Output Schema:" << endl;
        schema.Print ();
        cout << "---------------------------------" << endl;

    }
    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
    pipeMap[pid] = new Pipe (BUFFSIZE);
    relOp = new Sum ();
    from->ExecuteNode (pipeMap);
    ((Sum *)relOp)->Run (*(pipeMap[from->pid]), *(pipeMap[pid]), compute);
    from->Wait ();
    }
};
```

f. **Class DistinctOpNode (The Join Node): DistinctOpNode** class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class DistinctOpNode : public RelOpNode {
public:
    RelOpNode *from;
    DistinctOpNode () : RelOpNode (D) {}
    ~DistinctOpNode () {
        if (from) delete from;
    }
    void PrintNode () {
        from->PrintNode ();
        cout << "---------------------------------" << endl;
        cout << " DISTINCT OPERATION" << endl;
        cout << " Input Pipe ID : " << from->pid << endl;
        cout << " Output Pipe ID : " << pid << endl;
        cout << " Output Schema:" << endl;
        schema.Print ();
        cout << "---------------------------------" << endl;
    }
    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
        pipeMap[pid] = new Pipe (BUFFSIZE);
        relOp = new DuplicateRemoval ();
        from->ExecuteNode (pipeMap);
        ((DuplicateRemoval *)relOp)->Run (*(pipeMap[from->pid]), *(pipeMap[pid]), schema);
        from->Wait ();
    }
};
```

g. **Class GroupByOpNode (The Join Node): GroupByOpNode** class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class GroupByOpNode : public RelOpNode {
public:
    RelOpNode *from;
    Function compute;
    OrderMaker group;
    bool distinctFunc;
    GroupByOpNode () : RelOpNode (GB) {
        distinctFunc=false;
    }
    ~GroupByOpNode () {
        if (from) delete from;
    }
    void PrintNode () {
        from->PrintNode ();
        cout << "---------------------------------" << endl;
        cout << " GROUP OPERATION" << endl;
        cout << " Input Pipe ID : " << from->pid << endl;
        cout << " Output Pipe ID : " << pid << endl;
        cout << " Output Schema : " << endl;
        schema.Print ();
        cout << " Function : " << endl;
        compute.Print (&from->schema);
        if (distinctFunc){
            cout << " Distinct Function :  True" <<  endl;
        }
        cout << " Grouping On : " << endl;
        group.PrintWithSchema(&from->schema);
        cout << "---------------------------------" << endl;
    }
    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
        pipeMap[pid] = new Pipe (BUFFSIZE);
        relOp = new GroupBy ();
        from->ExecuteNode(pipeMap);
        ((GroupBy *)relOp)->Run (*(pipeMap[from->pid]), *(pipeMap[pid]), group, compute,distinctFunc);
        from->Wait ();
    }
};
```

h. **Class WriteOutOpNode (The Join Node): WriteOutOpNode** class represents the join node. It inherits the basic node structure defined in the RelOpNode Class.

```
class WriteOutOpNode : public RelOpNode {
public:
    RelOpNode *from;
    FILE *output;
    WriteOutOpNode () : RelOpNode (W) {}
    ~WriteOutOpNode () {
        if (output) delete output;
        if (from) delete from;
    }
    void PrintNode () {
        from->PrintNode ();
        cout << "----------------------------------" << endl;
        cout << " WRITE OPERATION" << endl;
        cout << " Input Pipe ID : " << from->pid << endl;
        cout << "----------------------------------" << endl;
    }
    void ExecuteNode (unordered_map<int, Pipe *> &pipeMap) {
        relOp = new WriteOut ();
        from->ExecuteNode (pipeMap);
        ((WriteOut *)relOp)->Run (*(pipeMap[from->pid]), output, schema);
        from->Wait ();
    }
};
```

2) SQL.h File: The SQL class is the acts as the database driver class. SQL class main method creates the instance of the class and start the database engine using StartSQLDatabase() method.

```
// SQL class parses and executes all the queries
class SQL{
    public:
        SQL();
        void StartSQLDatabase();
        void ParseAndExecuteSelect();
        void ParseAndExecuteCreate();
        void ParseAndExecuteInsert();
        void ParseAndExecuteDrop();
};
```

Function Description:

1) void StartSQLDatabase ();

Description: This function is responsible to start the SQL database engine.

2) void ParseAndExecuteSelect ();

Description: This method is responsible to make a query tree by parsing ANdlist and loading statistics.txt file. This function makes use of estimate function present in the statistics class in order to estimate the cost of joins. The join with least cost is selected. The tree is executed after the join is selected and tree is built.

3) void ParseAndExecuteCreate ();

Description: By using query inputs this method generates the heap files or sorted DB files.The chema entry for the new tables is saved in the catalog file.

4) void ParseAndExecuteInsert ();

Description: This method loads the schema stored in the catalog file previously and then this function loads the data from the text file given.

5) void ParseAndExecuteDrop ();

Description: This function is used to delete the database heap file. To do that the schema from the catalog file is removed so that whole table is deleted.

3) **Query Syntax:** The syntax for all query "CRUD" operations are as below:

a) **CREATE operation:**

> CREATE TABLE tableName (att1 INTEGER, att2 DOUBLE, att3 STRING) AS HEAP

> CREATE TABLE tableName (att1 INTEGER, att2 DOUBLE, att3 STRING) AS SORTED ON att1, att2

b) **INSERT Operation:**

> INSERT 'fileName' INTO tableName

c) **DROP Operation:**

> DROP TABLE tableName

d) **SET Operation:**

> SET OUTPUT STDOUT

> SET OUTPUT 'fileName'

> SET OUTPUT NONE

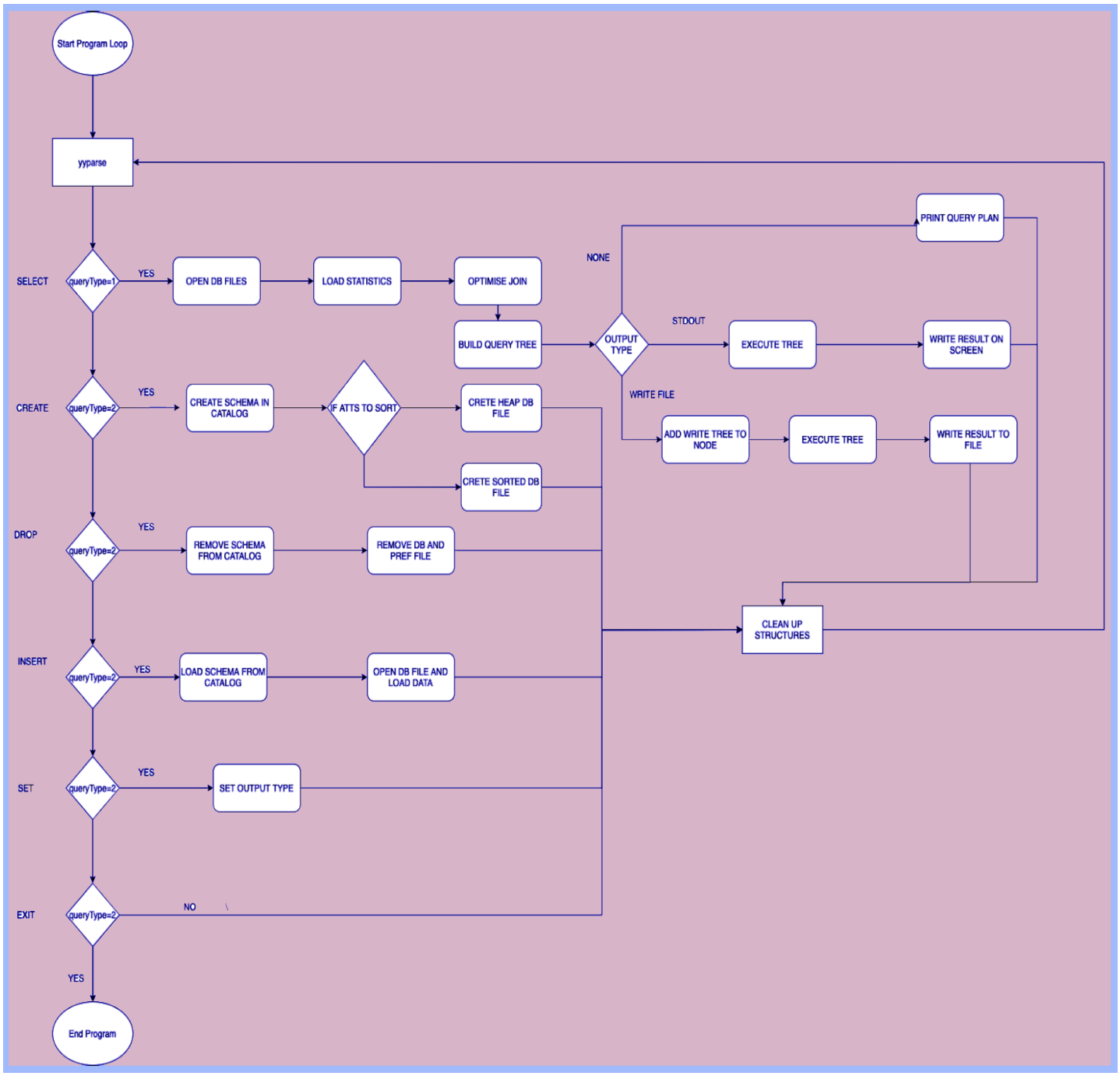
e) **EXIT Operation:**

> EXIT

f) **SELECT Operation:**

> SELECT can contain projection, join, group by, selection, distinct, sum aggregate function, sum aggregate function with distinct.

4) Execution Flow: Query Execution Flow is as below :



3) Program Run Results: All the screenshots are added in screenshots folder of this submission**.**

Query 1

```
azureuser113@VM-01:~/p5$ ./a5.out

STARTING DATABASE SESSION...

 25% [|||||||||||||||                                                      ]
 50% [||||||||||||||||||||||||||||||                                       ]
 75% [|||||||||||||||||||||||||||||||||||||||||||||||                      ]
100% [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]


OUPUT SET TO : STDOUT

[CREATE TABLE nation (n_nationkey INTEGER, n_name STRING, n_regionkey INTEGER, n_comment STRING ) AS HEAP
CREATE executed successfully !

[INSERT 'nation.tbl' INTO nation
INSERT executed successfully !

SELECT n.n_nationkey
FROM nation AS n
[WHERE (n.n_name = 'UNITED STATES')
n.n_nationkey: [24]
1 records found!
```

Query 2

```
[CREATE TABLE region (r_regionkey INTEGER, r_name STRING, r_comment STRING) AS HEAP
CREATE executed successfully !


[INSERT 'region.tbl' INTO region
INSERT executed successfully !

SELECT n.n_name
FROM nation AS n, region AS r
[WHERE (n.n_regionkey = r.r_regionkey) AND (n.n_nationkey > 5)
n.n_name: [MOZAMBIQUE]
n.n_name: [MOROCCO]
n.n_name: [KENYA]
n.n_name: [UNITED STATES]
n.n_name: [PERU]
n.n_name: [JAPAN]
n.n_name: [INDONESIA]
n.n_name: [INDIA]
n.n_name: [CHINA]
n.n_name: [VIETNAM]
n.n_name: [GERMANY]
n.n_name: [ROMANIA]
n.n_name: [RUSSIA]
n.n_name: [UNITED KINGDOM]
n.n_name: [FRANCE]
n.n_name: [IRAQ]
n.n_name: [JORDAN]
n.n_name: [IRAN]
n.n_name: [SAUDI ARABIA]
19 records found!
```

Query 3

```
SELECT SUM (n.n_nationkey)
FROM nation AS n, region AS r
WHERE (n.n_regionkey = r.r_regionkey) AND (n.n_name = 'UNITED STATES')
sum: [24]
1 records found!
```

Query 4

```
SELECT SUM (n.n_regionkey)
FROM nation AS n, region AS r
WHERE (n.n_regionkey = r.r_regionkey) AND (n.n_name = 'UNITED STATES')
GROUP BY n.n_regionkey

sum: [1], n.n_regionkey: [1]
1 records found!
```

Query 5

```
SELECT SUM DISTINCT (n.n_nationkey + r.r_regionkey)
FROM nation AS n, region AS r, customer AS c
WHERE (n.n_regionkey = r.r_regionkey) AND (n.n_nationkey = c.c_nationkey) AND (n.n_nationkey > 10)
GROUP BY r.r_regionkey

sum: [45], r.r_regionkey: [0]
sum: [43], r.r_regionkey: [1]
sum: [57], r.r_regionkey: [2]
sum: [73], r.r_regionkey: [3]
sum: [56], r.r_regionkey: [4]
5 records found!
```

a) Gtest Results: