MiniSQL阶段报告5

——Executor

> 第七小组

3.1 实验概述

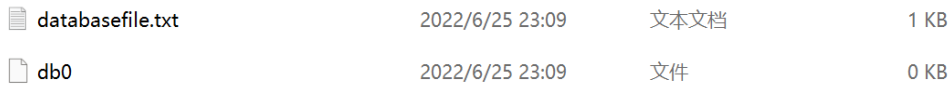
Executor（执行器）的主要功能是根据解释器（Parser）生成的语法树，通过Catalog Manager 提供的信息生成执行计划，并调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行执行，最后通过执行上下文`ExecuteContext`将执行结果返回给上层模块。

3.2 函数实现

- `ExecuteEngine::ExecuteCreateDatabase(*ast, *context)`
  - 思路
    - 先找是否已经有同名数据库
    - 建立数据库，插入`unordered_map dbs_`中，记录当前的数据库
  - 测试
    - 插入前

```
minisql > create database db0;
cost time: 0.002825
minisql >
```

    - 插入后文件中新建了db0，databasefile.txt中已经记录了已有的数据库db0



- `ExecuteEngine::ExecuteDropDatabase(*ast, *context)`
  - 思路
    - 在`dbs_`中找到该数据库，删除
    - 在文件中删除
  - 测试

```
minisql > drop database db0
;
cost time: 0.004111
minisql > show databases;
Database:
No database.
cost time: 2.8e-05
```

- `ExecuteEngine::ExecuteShowDatabases(*ast, *context)`
  - 思路
    - 打印`dbs_`中已有的数据库
  - 测试

```
minisql > show databases;
Database:
db1
db0
cost time: 7.5e-05
```

- `ExecuteEngine::ExecuteUseDatabase(*ast, *context)`
  - 思路
    - 找到database并把它作为`current_db_`
  - 测试

```
minisql > use db0;
cost time: 0.000102
```

- `ExecuteEngine::ExecuteShowTables(*ast, *context)`
  - 思路
    - 在`Current Database`中的`catalog_mgr_`中的函数`GetTables()`得到当前数据库中的表名，输出。
  - 测试
    - 此时打印该database里已经有的tablename

```
minisql > show tables;
t1
cost time: 6.2e-05
```

- `ExecuteEngine::ExecuteCreateTable(*ast, *context)`
  - 思路
    - 遍历语法树将column和type的信息收集，形成一个列的`vector`
    - 注意`primary key`，该列不能有重复的元素
    - 在`Current database`中的`catalog_mgr_`中调用`CreateTable()`来新建一个table
  - 测试
    - ```
      minisql > create table t1(a int, b char(20) unique, c float, primary key(a, c));
      cost time: 0.000494
      ```
- `ExecuteEngine::ExecuteDropTable(*ast, *context)`
  - 思路
    - 得到需要被drop的table name
    - 调用`current database`中的`catalog_mgr_`中的`DropTable()`函数来drop
  - 测试

```
minisql > show tables;
t
cost time: 7.7e-05
minisql > drop table t;
cost time: 0.000147
minisql > show tables;
cost time: 9e-06
```

- `ExecuteEngine::ExecuteShowIndexes(*ast, *context)`
  - 思路
    - 用`GetTableIndexes()`来得到表的索引并且打印
  - 测试

```
minisql > show indexes;
t1 b
t1 c
```

- `ExecuteEngine::ExecuteCreateIndex(*ast, *context)`
  - 思路
    - 检查是否在唯一键上建立索引
```

        - 调用`current database`中的`catalog_mgr_`中的`CreateIndex()`函数
    - 测试

```
minisql > create index id1 on t1(b);
cost time: 0.000113
```

- `ExecuteEngine::ExecuteDropIndex(*ast, *context)`
    - 思路
        - 在现有的tables中寻找同名的index
        - 调用`current database`中的`catalog_mgr_`中的`DropIndex()`函数
    - 测试
        - 此时把之前新建的在b上的index删除

```
minisql > drop index id1;
cost time: 0.000153
minisql > show indexes;
t1 c
```

- `ExecuteEngine::ExecuteSelect(*ast, *context)`
    - 思路
        - 遍历语法树得到需要被select的columns
        - 通过语法树得到每一个条件，每得到一个条件返回一个`vector<RowId>`最后对结果做布尔运算
            - 用`table iterator`结合需要找的columns找到每个符合要求的`field`然后输出结果
    - 测试

```
minisql > select * from t1;
1 aaaU 1.100000
-------------
2 abcU 3.000000
-------------
```

        - 带条件的查找：

```
minisql > select a from t1 where b = "aaa";
1
cost time: 0.000177
```

        - 嵌套条件

```
minisql > select a from t1 where b = "aaa" or c = 3;
1
2
cost time: 0.00017
```

- `ExecuteEngine::ExecuteInsert(*ast, *context)`
    - 思路
        - 调用`current database`中的`catalog_mgr_`中的`GetTable()`函数找到要被插入的表
        - 通过语法树得到需要被插入的一条记录的type以及内容，与该表的column做检查
        - 检查unique以及primary key
        - 调用`current table`中的`GetTableHeap()`中的`InsertTuple()`函数插入一条记录
        - 更新index
    - 测试

```
minisql > insert into t1 values(1, "aaa", 1.1);
cost time: 0.000234
```

```
minisql > insert into t1 values(2, "abc", 3);
cost time: 0.000197
```

        - Unique约束：

```
minisql > insert into t1 values(3, "abc", 2.2);
对于Unique列，不应该插入重复的元组
cost time: 0.000103
```

- Primary key 约束：

```
minisql > insert into t1 values(2, "nbv", 3);
conflict with primary key!
cost time: 0.000166
```

- 格式错误的情况：

```
minisql > insert into t1 values(3, 2);
Wrong Type!
cost time: 4.9e-05
```

- `ExecuteEngine::ExecuteDelete(*ast, *context)`
  - 思路
    - 调用`current database`中的`catalog_mgr_`中的`GetTable()`函数找到要被删除记录的表
    - 通过类似于`select`中的方法来找到需要被删除的`vector<RowId>`然后通过迭代器删除
    - 删除index
  - 测试

```
minisql > delete from t1 where b = "bbbbb" and c = 3;
cost time: 0.000309
minisql > select * from t1;
1 aaaU 1.100000
-------------
cost time: 0.000204
```

- `ExecuteEngine::ExecuteUpdate(*ast, *context)`
  - 思路
    - 与`select`、`insert`、`delete`中原理一致
    - 更新也需判断unique和primary key等约束条件
    - 更新index
  - 测试

```
minisql > update t1 set b = "bbbbb" where a =2;
cost time: 0.000184
```

此时select，可知表中内容为：

```
minisql > select * from t1;
1 aaaU 1.100000
-------------
2 bbbbb 3.000000
-------------
```

已经成功update。

与primary key冲突的update：

```
minisql > update t1 set a = 1 where a = 2;
cost time: 0.000405
minisql > select * from tq;
cost time: 1.1e-05
minisql > select * from t1;
1 aaaU 1.100000
-------------
1 bbbbb 3.000000
-------------
cost time: 0.000166
minisql > update t1 set c = 1.1 where b = "bbbbb";
conflict with primary key!
cost time: 0.000218
```

与unique冲突的update：

```
minisql > update t1 set b = "aaa" where c = 3;
对于Unique列，不应该插入重复的元组
cost time: 0.000148
```

- `ExecuteEngine::ExecuteExecfile(*ast, *context)`
  - 思路
    - 与`main.cpp`中的内容相同，只是执行文件中的每一行指令
  - 测试
    - 最后验收的时候展示
- `ExecuteEngine::ExecuteQuit(*ast, *context)`
  - 思路
    - `context->flag_quit_ = true`
  - 测试
    - 命令行中输入`quit;`回车后程序结束