EXECUTOR of MINISQL

实验报告

黄子源，3200105586

计算机科学与技术

0. 实验内容

该部分由黄子源一人完成。

本次实验将设计MiniSQL的执行器部分，该部分处理用户输入的指令，并将运行结果反应到输出终端。主要功能包括：

- 解析解释器根据用户指令生成的语法树；
- 根据语法树调用相应的下层模块，正确完成指令的操作；
- 检查输入参数是否合法，并向用户提供错误信息；
- 给出指令的运行结果。

1. 语法树解析

执行器获取的指令内容，完全来自上层解释器传入的语法树，不同指令的语法树结构，也由解释器预先定义。因此该部分设计流程为：先向解释器输入指令获得语法树，并打印语法树结构，通过参数和节点值的一一对应关系来获知语法树的生成规则，然后再根据规则直接编写解析逻辑。

2. 指令执行

以下为MiniSQL支持的所有指令。

2.1 数据库操作

执行器对数据库的所有操作都是通过下层的`DBStorageEngine`进行的，每一个`DBStorageEngine`对应一个逻辑意义上的数据库，通过数据库名（唯一标识符）区分。当MiniSQL启动时，执行器会从硬盘读取已有的数据库列表，并自动地为每一个数据库创建`DBStorageEngine`实例。相应地，MiniSQL正常退出时，执行器也会更新硬盘中的数据库列表。数据库内部的数据存取，都由具体的`DBStorageEngine`实例完成。

执行器允许用户选择某个数据库指定为 “当前使用中数据库” Current Database，所有对数据记录、数据表和索引的操作都会对Current Database进行，如果没有指定Current Database，执行器将只允许用户进行数据库操作，其它操作无法执行。`create database`和`use`都可以设置Current Database。

2.1.1 Create Database

```
create database DATABASE_NAME;
```

创建一个标识符为`DATABASE_NAME`的数据库，成功后会自动将新数据库指定为Current Database。

2.1.2 Drop Database

```
drop database DATABASE_NAME;
```

删除标识符为`DATABASE_NAME`的数据库。如果删除的数据库已被指定为Current Database，则执行器会将Current Database清空，用户需要手动指定另一个数据库为Current Database才能进一步操作。

2.1.3 Show Database

```
show databases;
```

将当前数据库列表打印到控制台。

2.1.4 Use Database

```
use DATABASE_NAME;
```

将标识符为`DATABASE_NAME`的数据库指定为Current Database。

2.2 表操作

2.2.1 Show Tables

```
show tables;
```

在控制台打印Current Database中所有表的表名。

2.2.2 Create Table

```
create table TABLE_NAME(IDENTIFIER TYPE, ... , primary key(IDENTIFIER...));
```

创建一个标识符为`TABLE_NAME`的数据表。

数据表中的每一个字段都由标识符`IDENTIFIER`和类型`TYPE`定义，如果希望该字段不出现重复数据，可以在`TYPE`后附加关键字`unique`；最后一个参数可以使用关键字`primary key`指定哪些标识符对应的字段为表的主键。

一个正确的示例如下：

```
create table t1(a int, b char(20) unique, c float, primary key(a, c));
```

该指令创建一个名为`t1`的表，表中的记录具有整数字段`a`，唯一的文本字段`b`和浮点数字段`c`，`a`和`c`是表的主键。

创建数据表时，如果表中有`unique`修饰的字段，或是指定了主键，会为这些字段自动生成索引。

2.2.3 Drop Table

```
drop table TABLE_NAME;
```

删除标识符为`TABLE_NAME`的数据表。

2.3 索引操作

2.3.1 Show Indexes

```
show indexes;
```

在控制台打印所有索引的索引名列表，包括由`create table`自动为`unique`字段和`primary key`字段生成的索引。

2.3.2 Create Index

```
create index INDEX_NAME on TABLE_NAME(IDENTIFIER, ...);
```

在数据表`TABLE_NAME`上创建一个标识符为`INDEX_NAME`的索引，该索引基于用户指定的字段建立。

2.3.3 Drop Index

```
drop index INDEX_NAME;
```

删除标识符为`INDEX_NAME`的索引。

2.4 数据操作

2.4.1 Select

```
select IDENTIFIER, ... from TABLE_NAME where CONDITIONS...;
select * from TABLE_NAME where CONDITIONS...;
```

在数据表`TABLE_NAME`中，筛选出所有符合用户指定条件`CONDITIONS`的记录，然后将记录按照用户指定的字段格式打印。若字段格式为`*`，则会将记录原样打印。

过长的记录（如超过12个字符的文本）只打印10个字符，余下内容省略。

`select`打印出的表格末尾会附带一个记录总数。

2.4.2 Insert

```
insert into TABLE_NAME values(VALUE, ...);
```

向数据表`TABLE_NAME`插入一条指定值的新记录，记录内容必须符合创建数据表时定义的字段格式，不允许向主键插入null值。

整数数值允许赋值给浮点数字段，但浮点数数值不允许赋值给整数字段。

2.4.3 Delete

```
delete from TABLE_NAME;
delete from TABLE_NAME where CONDITIONS...;
```

删除数据表`TABLE_NAME`中所有符合用户指定条件`CONDITIONS`的记录，不指定条件时将清空该数据表。

2.4.4 Update

```
update TABLE_NAME set IDENTIFIER = VALUE;
update TABLE_NAME set IDENTIFIER1 = VALUE1, IDENTIFIER2 = VALUE2, ... where
CONDITIONS...;
```

更新数据表`TABLE_NAME`中所有符合用户指定条件`CONDITIONS`的记录，将这些记录的`IDENTIFIER`字段设为新值。

该指令执行后会打印更新过的记录总数。

2.5 其他操作

2.5.1 Execute File

```
execfile FILENAME;
```

从文件`FILENAME`（字符串参数）中读取文本内容，并将其作为SQL指令执行，效果等同于将文件中的文本内容逐行输入MiniSQL控制台。

每执行一条指令，MiniSQL会先输出执行结果再执行下一条指令，某条指令出错不会影响后续指令的执行。

2.5.2 Quit

```
quit;
```

关闭MiniSQL并收获一个贴心告别。

`quit`指令是安全的退出方式，MiniSQL在收到`quit`指令后会将所有数据库内容转移到硬盘文件中，以便于下次启动时读取。若通过其它方式结束程序，则数据可能丢失。

3. 附加设计

3.1 条件判断

`SELECT`，`UPDATE`，`DELETE`等指令执行过程中，需要根据数据库中每一条记录的内容动态判断，符合用户指定的条件才进行相应操作。执行器中将这部分功能设计为一个单独的函数，以便于代码复用。函数原形为：

```
dberr_t ExecuteEngine::LogicConditions(pSyntaxNode ast, ExecuteContext *context,
const Row &row, Schema *schema)
```

该函数从调用处接收包含判断条件的语法树节点（节点类型为`kNodeConditions`），格式上直接沿用解释器给出的语法树格式，并从上层指令取得记录`row`（用于取出数据）和该表的结构信息`schema`（用于根据条件中的标识符找到数据位置），判断结果以布尔值的形式保存在`context->condition_`中返回。当节点为`nullptr`时该函数将无条件返回`true`，对应用户不输入任何条件时，解释器将不在语法树中生成`kNodeConditions`节点。

由于是对单条记录进行判断，该函数可以兼容不同的记录遍历方式，交由上层指令决定使用数据表迭代器还是索引迭代器。

4. 源代码

由于执行器部分代码量较大，若要获知具体的实现细节，可参阅以下源码文件：

- `src/include/executor/execute_engine.h`
- `src/executor/execute_engine.cpp`