项目报告——简易数据库引擎（MiniDB）

姓名：么冠雄    学号:10234700475

一、项目简介：

简易数据库引擎（MiniDB）是一个用 C++ 实现的简单数据库管理系统，支持以下的SQL基础操作：创建表格，删除表格，单次插入与删除数据，"单个等于条件"查询与更新数据，导出CSV文件。

源代码：<https://github.com/xiong67460/MiniDB.git>

##### 项目结构：

MiniDB/  
├── main.cpp # 主程序入口  
├── common/  
│ └── command.h # 命令类定义  
├── parser/  
│ ├── parser.h # SQL解析器头文件  
│ └── parser.cpp # SQL解析器实现  
├── catalog/  
│ ├── catalog\_manager.h # 目录管理器头文件  
│ └── catalog\_manager.cpp # 目录管理器实现  
├── record/  
│ ├── record\_manager.h # 记录管理器头文件  
│ └── record\_manager.cpp # 记录管理器实现  
├── data/ # 数据文件目录  
├── metadata/ # 元数据文件目录  
└── README.md # 项目说明文档

##### 项目编译与运行：

编译需要C++17或以上的版本，且需要编译器支持 std::filesystem 。

使用 g++ 编译:

g++ -std=c++17 -o MiniDB main.cpp parser/parser.cpp catalog/catalog\_manager.cpp record/record\_manager.cpp

也可以使用 clang++ 编译:

clang++ -std=c++17 -o MiniDB main.cpp parser/parser.cpp catalog/catalog\_manager.cpp record/record\_manager.cpp

在编译完成后可以双击MiniDB.exe文件或者在终端输入以下指令来运行项目

./MiniDB

二、项目功能

##### 1.创建表格：

create table stu(id int,name string,score int);  
--create table 表名（属性1 数据类型，属性2 数据类型，...）;

##### 2.删除表格：

drop table stu;  
--drop table 表名;

##### 3.插入数据：

insert into stu values(1,ygx,99);  
--insert into stu values(属性1的值，属性2的值...);

##### 4.查询数据

select\* from stu;  
--select\*from 表名； 查询表中全部数据  
select\* from stu where id=1;  
--select\*from 表名 where 属性=值; 条件查询

##### 5.更新数据

update stu set score=96 where name=ygx;  
--update 表名 set 属性=值 where 属性=值;

##### 6.删除数据

delete from stu where id=1;  
--delete from 表名 where 属性=值;

##### 7.导出CSV文件

export table stu to 'stu.csv';  
--export table 表名 to '文件名.csv';

三、主要模块说明

* **common/**：定义各种SQL操作类型和命令类。
* //枚举定义了SQL操作类型  
  enum class CommandType  
  {  
   CREATE, // 创建表  
   INSERT, // 插入数据  
   SELECT, // 查询数据  
   DELETE, // 删除数据  
   UPDATE, // 更新数据  
   DROP, // 删除表  
   EXPORT, // 导出表为CSV  
   UNKNOWN // 未知命令  
  };  
    
  // 命令基类  
  class Command  
  {  
  public:  
   CommandType type = CommandType::UNKNOWN; // 命令类型,默认为UNKOWN  
   virtual ~Command() = default;   
  };  
    
  //以CREATE命名子类为例  
  class CreateCommand : public Command  
  {  
  public:  
   string tableName;   
   vector<pair<string, string>> columns;   
  };
* **parser/**：SQL解析器，将SQL字符串转为命令对象。
* class Parser  
  {  
  public:   
   static unique\_ptr<Command> parse(const string &sql);  
  };  
    
  /\*在parse函数中，确定输入SQL语句的命令类型，并对SQL语句进行解析，将字符串转化为命令对象\*/  
  //下方是SQL语句为CREATE类型的部分  
   if (lower.find("create table") == 0)  
   {  
   auto cmd = make\_unique<CreateCommand>();  
   cmd->type = CommandType::CREATE;  
   size\_t start = lower.find("table") + 6;  
   size\_t paren = sql.find('(', start);  
   string tableName = sql.substr(start, paren - start);  
   cmd->tableName = clean(tableName);  
   size\_t endParen = sql.find(')', paren);  
   string fields = sql.substr(paren + 1, endParen - paren - 1);  
   stringstream ss(fields);  
   string segment;  
   while (getline(ss, segment, ','))  
   {  
   stringstream part(segment);  
   string colName, colType;  
   part >> colName >> colType;  
   cmd->columns.emplace\_back(colName, colType);  
   }  
   return cmd;  
   }
* **catalog/**：目录管理器，负责表结构的创建和删除。
* class CatalogManager  
  {  
  public:  
   //创建新表  
   static bool createTable(const string &tableName, const vector<pair<string, string>> &columns);  
   //删除表  
   static bool dropTable(const string &tableName);  
  };
* **record/**：记录管理器，负责数据的插入、查询、删除、更新。
* class RecordManager  
  {  
  public:  
   static bool insertRecord(const string &tableName, const vector<string> &values);  
   static vector<vector<string>> selectAll(const string &tableName);  
   static vector<vector<string>> selectWhere(const string &tableName, const string &column, const string &value);  
   static int deleteWhere(const string &tableName, const string &column, const string &value);  
   static int updateWhere(const string &tableName, const string &setColumn, const string &setValue, const string &whereColumn, const string &whereValue);  
   static bool exportToCSV(const string &tableName, const string &filePath);  
   static string trim(const string &s);  
  };
* **data/**：以tbl格式存放数据文件。
* //stu.tbl  
  1,ygx,99  
  2,shr,88  
  3,zkx,96  
  4,jjh,99
* **metadata/**：存放元数据文件, 记录表名、各属性及其数据类型。
* //stu.meta  
  Table: stu  
  Columns:  
  id int  
  name string  
  score int
* **main.cpp**：程序入口，命令行交互，分发SQL命令。
* 接收用户输入的SQL语句，使用解析器（Parser）将其转化为命令对象，分发给目录管理器或记录管理器执行具体操作
* int main()  
  {  
   string sql;  
   cout << "hello, welcome to MiniDB by YGX\n";  
   cout << "Type 'exit' to quit\n\n";  
     
   while (true)  
   {  
   cout << "SQL> ";   
   getline(cin, sql);   
   sql = clean(sql);  
   if (sql == "exit")  
   break;  
   if (sql.empty())  
   continue;  
     
   auto cmd = Parser::parse(sql);  
   //.....各命令类型对应的不同操作  
   }  
  }

四、主要实现思路

##### 1.创建/删除表：

* 接收用户创建（删除）表请求。
* 检查表是否已存在。
* 创建（删除）元数据文件。
* 返回创建（删除）表结果

##### 2.插入：

* 用户输入一条新数据。
* 程序检查 data 目录是否存在，不存在则创建。
* 拼接数据文件路径（如 data/表名.tbl）。
* 以追加模式打开数据文件，将新数据按逗号分隔写入文件末尾。

##### 3.查询：

* 用户请求查询表数据。
* 程序打开对应的 .tbl 文件，逐行读取。
* 跳过空行和以 # 开头（已删除）的行。
* 解析每一行数据，按逗号分割为字段，存入结果集。
* 若有条件查询，先读取元数据文件，获取字段名和索引，再逐行比对目标字段值，筛选出符合条件的记录。

##### 4.删除：

* 用户请求删除某些记录。
* 程序读取元数据文件，确定条件字段的索引。
* 逐行读取数据文件，对每一行判断是否满足删除条件。
* 满足条件的行前加 # 标记为“逻辑删除”，实现“软删除”。

##### 5.更新：

* 用户请求更新某些记录。
* 程序读取元数据文件，确定 set 和 where 字段的索引。
* 逐行读取数据文件，对每一行判断是否满足 where 条件。
* 满足条件的行，修改 set 字段的值。

##### 6.导出CSV文件：

* 用户请求导出表为 CSV 文件。
* 程序读取元数据文件，获取字段名，写入 CSV 表头。
* 读取数据文件，跳过空行和已删除行，将每条记录写入 CSV 文件。

五、测试与运行结果

SQL> create table stu(id int,name string,score int);  
Table 'stu' created successfully with 3 columns.  
  
SQL> insert into stu values(1,ygx,99);  
Successfully inserted 3 values into table 'stu'.  
  
SQL> insert into stu values(2,shr,88);  
Successfully inserted 3 values into table 'stu'.  
  
SQL> insert into stu values(3,zkx,92);  
Successfully inserted 3 values into table 'stu'.  
  
SQL> inssssssert into stu values(4,jjh,99);  
Unrecognized SQL command. Supported commands:  
 - CREATE TABLE <table\_name> (<column\_definitions>)  
 - DROP TABLE <table\_name>  
 - INSERT INTO <table\_name> VALUES (<values>)  
 - SELECT \* FROM <table\_name> [WHERE <condition>]  
 - DELETE FROM <table\_name> WHERE <condition>  
 - UPDATE <table\_name> SET <column> = <value> WHERE <condition>  
 - EXPORT TABLE <table\_name> TO <file\_path>  
  
SQL> insert into stu values(4,jjh,99);  
Successfully inserted 3 values into table 'stu'.  
  
SQL> select\* from stu;  
Found 4 record(s) in table 'stu':  
1 ygx 99  
2 shr 88  
3 zkx 92  
4 jjh 99  
  
SQL> select\* from stu where name=shr;  
Found 1 record(s) in table 'stu' where name = shr:  
2 shr 88  
  
SQL> select\* from stu where score=99;  
Found 2 record(s) in table 'stu' where score = 99:  
1 ygx 99  
4 jjh 99  
  
SQL> update stu set score=97 where name=zkx;  
Successfully updated 1 record(s) in table 'stu' where name = zkx.  
  
SQL> delete from stu where id=2;  
Successfully deleted 1 record(s) from table 'stu' where id = 2.  
  
SQL> select\* from stu;  
Found 3 record(s) in table 'stu':  
1 ygx 99  
3 zkx 97  
4 jjh 99  
  
SQL> export table stu to 'stu.csv';  
Table 'stu' exported to 'stu.csv' successfully.  
  
SQL> drop table stu;  
Table 'stu' dropped successfully.

images文件夹中包含终端的运行结果照片，stu表被删除之前stu.meta、stu.tbl内容的照片，以及在MiniDB目录下生成CSV文件的照片。

六、项目特色

##### 1.极简实现，易于理解

MiniDB 采用 C++ 编写，核心代码简洁明了，便于学习和理解数据库系统的基本原理。

##### 2.文件级存储，结构清晰

数据与元数据分别存储于 data/ 和 metadata/ 目录，便于管理和扩展。

##### 3.支持基础 SQL 操作

实现了表的创建、删除、插入、查询、更新、删除、导出等常用 SQL 功能，覆盖数据库操作主流程。

##### 4.逻辑删除机制

删除数据时采用逻辑删除（在记录前加 #），保证数据可追溯。

##### 5.交互式命令行体验

提供友好的命令行界面，支持智能输入处理和详细的操作反馈，提升用户体验。

##### 6.易于扩展和维护

各模块职责分明，便于后续添加如索引、事务、权限等高级数据库功能。

七、不足之处

##### 1. 数据类型约束不足

尽管在创建表时需要设定各个属性的数据类型，但项目程序会将所有输入作为字符串处理。即使设置 id 为 int 类型，也可以输入 "one"。甚至可以在创建表时随意输入如 abc、strrrring 等作为数据类型，缺乏类型校验和约束。

##### 2. 功能缺失或不足

* 缺失 MySQL 等数据库系统具有的高级功能，如 join、group by、order by 等基础 SQL 语句也无法实现。
* 条件查询、更新和删除命令只能在 where 后添加一个条件，不能使用 and、or 连接多个条件，且仅支持“属性=值”的等值条件。
* 插入数据时只能插入一行数据，无法批量插入。

##### 3. 缺乏事务与并发控制

系统不支持事务（如 begin、commit、rollback），也没有并发控制机制，无法保证多用户或多进程同时操作时的数据一致性和安全性。

##### 4. 缺乏索引与性能优化

所有查询、更新、删除操作均为全表扫描，未实现索引机制，数据量大时性能较低。

##### 5. 安全性与权限控制不足

系统未实现用户管理和权限控制，所有用户均可对所有表进行任意操作，存在安全隐患。

##### 6. 错误处理和提示有限

虽然有部分错误提示，但对于复杂 SQL 语法、文件损坏、磁盘空间不足等异常情况的处理不够完善，用户体验有待提升。

##### 7. 代码健壮性和可维护性有提升空间

部分功能实现较为直接，缺乏模块间解耦和单元测试，后续扩展和维护难度较大。