模式信息管理器设计报告

许多（3110000090）

一、概述

模式信息管理器（Catalog Manager，下称 CM）是 MiniSQL 的一个组成部分，它负责管理整个数据库的模式信息，主要包括数据库的表定义信息、表中字段的定义信息和数据库的索引信息。数据库需要进行创建、删除、插入、查询等操作时，CM 会根据保存的模式信息，对 SQL 语句的合法性进行检查。此外，CM 还可以向 MiniSQL 的其他模块提供表定义等信息。

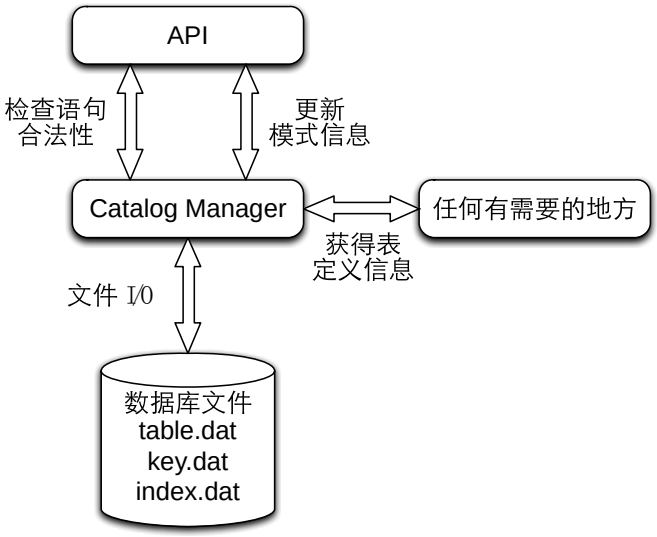
二、原理

在初始化阶段，CM 会读取 catalog 目录下的 table.dat、key.dat 和 index.dat 文件，把表定义信息、字段定义信息和索引定义信息读入内存中的 tableCatalog、keyCatalog 和 indexCatalog 中。为了考虑效率，之后的更新模式信息操作都在内存中完成。到最后的析构阶段，CM 会将内存中的模式信息回写到上述的三个文件。由于模式信息需要维护的文件远小于保存表记录的文件，故未使用缓冲区（buffer）来间接操作文件，而选择直接操作文件。

CM 的语句检查函数和更新模式信息函数均由上层的 API 模块调用。对于语句检查函数如果合法性满足模式信息的要求，这些函数不返回值，否则函数抛出异常信息，异常中包括错误的类型和内容，被捕获后就可以输出这个错误。而对于更新模式信息函数，调用后会对内存中的 tableCatalog、keyCatalog 和 indexCatalog 进行相应修改而不写入文件（在最后的析构阶段回写文件）。

MiniSQL 的其他部分在需要表定义等信息的时候，通过调用 CM 的 getTable()、getAttrName()等函数就可以得到所需的信息。

CM 与 MiniSQL 的其他部分之间的关系结构如图所示。注意：图中没有画出 MiniSQL 的记录管理器（Record Manager）、索引管理器（Index Manager）、缓冲区管理器（Buffer Manager）等部分。



三、文件结构

CM 使用三个文件对维护模式信息：table.dat、key.dat 和 index.dat。这三个文件分别存储了表定义信息、表中字段定义信息和索引定义信息，均位于名为 catalog 的子目录下。在 CM 的构造阶段，这三个文件的内容会被读入到 tableCatalog、keyCatalog、indexCatalog 这三个 vector 变量中；在析构阶段，则是从 vector 变量写回到文件。

文件使用二进制格式存储以节省空间，其中存储名称的元素以字符串的形式存储，其他元素以整数（默认是小端序）的形式存储。下文将对三个文件的存储结构逐一进行介绍。

1. table.dat 文件

该文件由连续的单位构成，每个单位记录一张表的定义信息，格式如下（长度的单位是字节）：

含义	标志	表名	属性（列）数	主键所在属性号	索引标志（列 i 若有索引，则本标志的位 i 为 1）	第一条属性在 key.dat 中的下标	第一条索引在 index.dat 中的下标
长度	1	20	1	1	4	2	2

位于起始位置的标志占 1 个字节，各位的含义如下（左边为高位，1 代表真，0 代表假）：

该单位空间被占用？	该表有主键？	该表有索引？					

2. key.dat 文件

该文件由连续的单位构成，每个单位记录一个字段（属性、键）的定义信息，格式如下：

含义	标志	属性名	属性类型（0、1、2 分别代表整数、字符串、浮点数）	属性长度	该表下一条属性的下标（若无则置-1）
长度	1	20	1	1	2

位于起始位置的标志占 1 个字节，各位的含义如下（左边为高位，1 代表真，0 代表假）：

该单位空间被占用？			该属性是主键？	该属性是唯一值？	该属性必须非空？	该属性有索引？	该属性所属表还存在下一条属性？

3. index.dat 文件

该文件由连续的单位构成，每个单位记录一条索引的定义信息，格式如下：

含义	标志	索引名	索引所属表的下标	索引在所属表的属性号	该表下一条索引的下标（若无则置-1）
长度	1	20	2	1	2

位于起始位置的标志占 1 个字节，各位的含义如下（左边为高位，1 代表真，0 代表假）：

该单位空间被占用？							该索引所属表还存在下一条索引？

四、函数介绍

1. 文件 I/O 函数（private 域，CM 内部使用）

```
void loadTableCatalogFromFile()
void loadKeyCatalogFromFile()
void loadIndexCatalogFromFile()
```

以上三个函数在 CM 的构造阶段被调用，三个函数分别从 table.dat、key.dat 和 index.dat 文件读入内容，并保存到内存中的 tableCatalog、keyCatalog 和 indexCatalog 变量中，在析构函数被调用之前，所有对模式信息的修改都在内存中完成。

```
void saveTableCatalogToFile()
void saveKeyCatalogToFile()
void saveIndexCatalogToFile()
```

以上三个函数在 CM 的析构阶段（程序结束时发生）被调用，三个函数分别把内存中的 tableCatalog、keyCatalog 和 indexCatalog 变量所储存的数据回写到文件 table.dat、key.dat 和 index.dat。在此之后 CM 被析构。

2. 获取模式信息的函数

```
Table &getTable(const std::string &tableName)
```

getTable()用于获得一张表的模式信息（以 MiniSQL 的 Table 型呈现），包括表本身和它的属性信息（MiniSQL 的 Attribute 型）。它接受一个字符串参数，表示表的名称。

```
std::string getAttrName(const std::string &indexName)
std::string getTableName(const std::string &indexName)
```

以上两个函数均以索引名为参数，分别用于获得属性（键）名、表名。

3. 创建表相关函数

```
void createTableCheck(const std::string &tableName, const std::vector<Attribute> &attributes)
```

在 API 模块中，createTableCheck()函数在下文的 createTable()前被调用，用于检查要执行的 SQL create table 语句是否可行。如果没有错误，函数正常运行，没有返回值；否则函数内部会抛出一个异常对象（为简便起见，异常都是字符串的形式），包括错误的类型和内容，以供输出错误信息（这种机制同样适用于下文所有*Check()型函数，后面不再重复说明）。函数接受两个参数，第一个参数代表要创建的表名，第二个参数是 vector<Attribute>型，代表表的属性信息。

```
void createTable(const std::string &tableName, const std::vector<Attribute> &attributes)
```

createTable()在 createTableCheck()函数之后被调用，接受的参数和 createTableCheck()相同，效果是向 tableCatalog 存入该表的表定义信息、向 keyCatalog 存入属性定义信息。

4. 删除表相关函数

```
void dropTableCheck(const std::string &tableName)
```

dropTableCheck()在 dropTable()函数之前被调用，用于检查要删除的表是否可以删除（即 SQL drop table 语句，例如，如果表不存在是不可以删除的）。函数接受一个参数 tableName，表示要删除的表名。

```
void dropTable(const std::string &tableName)
```

dropTable()函数接受的参数同上，作用是具体执行删除该表相关的一切模式信息，包括 tableCatalog 中的表定义信息、keyCatalog 中所有该表的属性信息和 indexCatalog 中所有针对该表属性的索引信息（如果有）。

5. 创建索引相关函数

```
void createIndexCheck(const std::string &indexName, const std::string &tableName, const std::string &keyName)
```

createIndexCheck()用于检查要创建的索引是否合法（即 SQL create index 语句）。函数接受三个参数：indexName 表示索引名，tableName 表示索引针对的表名，keyName 表示索引针对 tableName 表的属性名。

```
void createIndex(const std::string &indexName, const std::string &tableName, const std::string &keyName)
```

createIndex()在 createIndexCheck()后被调用，接受的参数同上。函数在 tableCatalog、keyCatalog 的相应位置中设置索引标志，并在 indexCatalog 中插入这一索引信息。

6. 删除索引相关函数

```
void dropIndexCheck(const std::string &indexName)
```

dropIndexCheck()在 dropIndex()函数之前被调用，用于检查要删除的索引是否可以删除（即 SQL drop index 语句，例如，如果索引不存在是不可以删除的）。函数接受一个参数 indexName，表示要删除的索引名。

```
void dropIndex(const std::string &indexName)
```

dropIndex()函数同样接受一个 indexName 参数，函数在 indexCatalog 中找到该索引之后，记录下来它针对的表和属性的位置（用来重置索引标志），并删除这一索引，最后在 keyCatalog 和 tableCatalog 对应的位置重置索引标志。如果此时该表没有索引了，还会重置“该表是否存在索引”标志。

7. 查询、插入、删除记录相关检查函数

```
void selectCheck(const Query &query)
```

selectCheck()函数接受一个代表查询语句的 Query 型对象（包括查询针对的表、查询条件等信息），检查 SQL select 语句的合法性：检查表是否存在，每个条件的左端是否是该表的属性，等等。

```
void insertTupleCheck(const std::string &tableName, const std::vector<std::string> &tuple)
```

insertTupleCheck()函数检查 SQL insert 语句的合法性，它接受两个参数：第一个参数表示要插入记录的目标表名，第二个参数是一组字符串，表示要插入的一行数据。函数会检查插入数据的个数、长度等是否符合模式信息中的规定。

```
void deleteTupleCheck(const Query &query)
```

deleteTupleCheck()函数检查 SQL delete 语句的合法性，它接受的参数和 selectCheck()相同，检查内容也是相同的：表是否存在、各条件左端是否是该表属性，等等。因此，该函数内部直接调用了 selectCheck()函数，以减少代码重复。