数据库大作业 总结报告

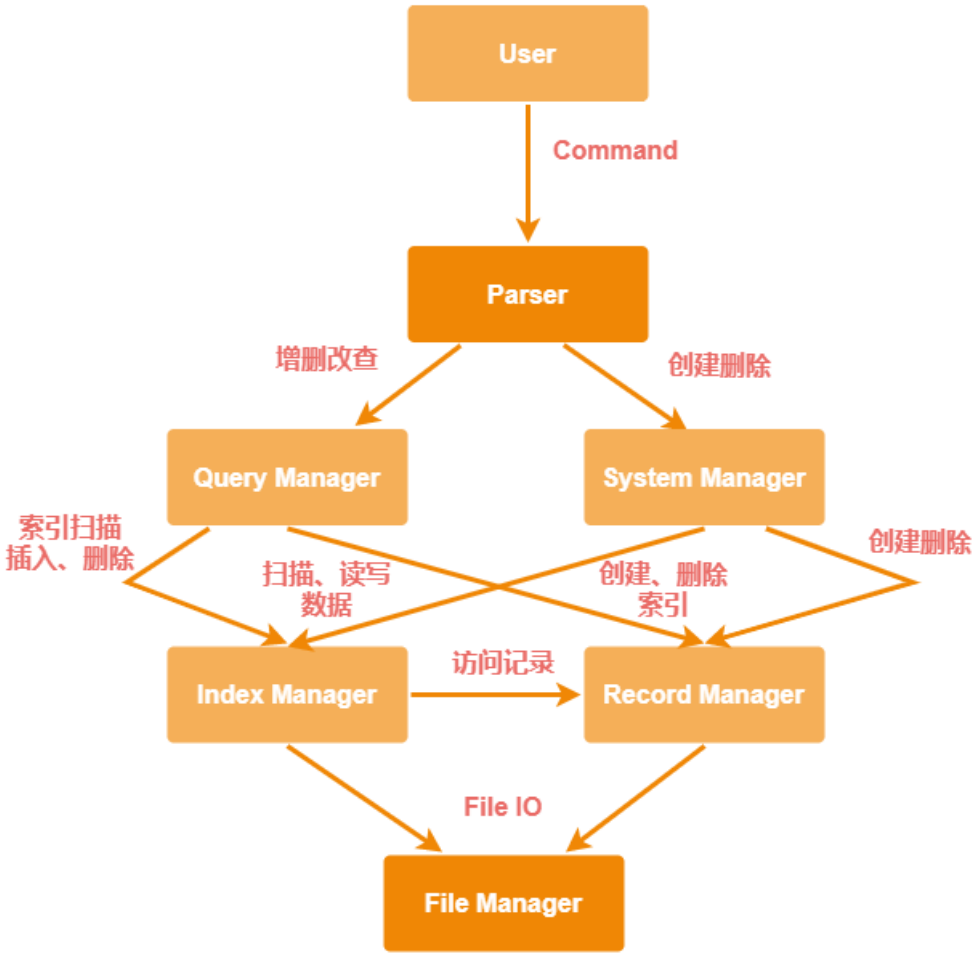
计82　洪昊昀　2017010591

1. 系统架构设计

我的数据库的系统架构设计与《数据库大作业详细说明》和 CS346 的说明是一致的：

1. 最底层是页式管理系统，负责以页的粒度来管理磁盘中的文件和操作文件，记录管理模块和索引管理模块建立在页式管理系统的基础上；
2. 记录管理模块负责表中记录的存储、创建、插入、修改；
3. 索引管理模块负责表的属性索引的创建、插入、删除，使用 B Tree 的数据结构加速查询；
4. 系统管理模块负责数据库和表的管理；
5. 查询解析模块负责解析用户的 SQL 指令，将它们对应为索引管理模块和系统管理模块的增删改查操作。

具体架构图如下：



2. 各模块详细设计

2.1 页式管理系统

复用了课程所给的页式文件系统的部分代码，再仿照参考文献3（https://github.com/Konano/UselessDatabase）的方式，使用开源代码库 JSON for Modern C++（https://github.com/nlohmann/json），在数据库配置信息（如数据库、索引、表的信息）时，不使用给定的页式文件系统，而使用这个代码库，将这些信息的存储可读化和简化，后文会对如何使用 JSON 进行文件管理作更详细的介绍。

2.2 记录管理模块

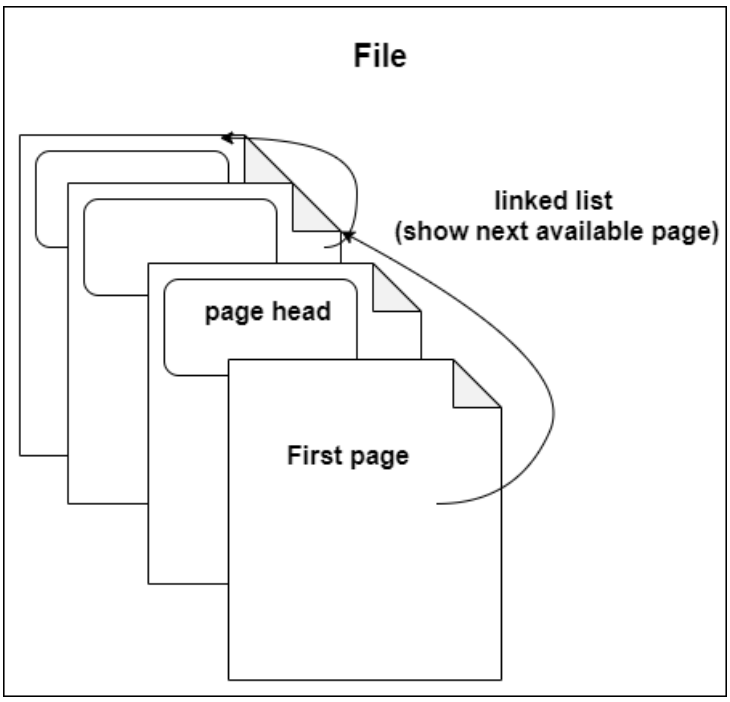
记录管理模块负责表中记录的存储、创建、插入、修改，直接操作页式管理系统。数据库的所有表的头信息以 JSON 文件的形式存储在所在的数据库的文件夹下，以`database.udb`命名，表头信息示例（nation 表）如下：

```
{
            "col_num": 4, // 该表一共有几列
            "col_ty": [  // 每列的数据类型
                {
                    "char_len": 0, // 字符串长度（只有字符串类型的数据该值才大于0）
                    "key": 0, // 索引建立在哪些列上
                    "name": "n_nationkey",  // 列名
                    "null": false, // 是否可为空
                    "ty": 0  // 0表示数据类型为INT，1为CHAR，2为VARCHAR，3为DATE，4为DECIMAL
                },
                {
                    "char_len": 25,
                    "key": 2,
                    "name": "n_name",
                    "null": true,
                    "ty": 1
                },
                {
                    "char_len": 0,
                    "key": 1,
                    "name": "n_regionkey",
                    "null": false,
                    "ty": 0
                },
```

```
            {
                "char_len": 152,
                "key": 2,
                "name": "n_comment",
                "null": true,
                "ty": 2
            }
        ],
        "f_key_index": [ // 外键index
            1
        ],
        "foreign": [  // 外键信息
            {
                "data": {
                    "cols": [
                        2
                    ],
                    "name": "n_regionkey"
                },
                "sid": 1
            }
        ],
        "index": [  // 索引信息（请参见索引模块部分）
            {
                "key": [
                    0
                ],
                "key_num": 1,
                "name": "pk_n_nationkey",
                "next_del_page": 4294967295,
                "offset": [
                    0
                ],
                "page_num": 3,
                "record_size": 12,
                "root_page": 1,
                "ty": [
                    0
                ]
            },
            {
                "key": [
                    2
```

```
                    ],
                    "key_num": 1,
                    "name": "fk_n_regionkey1",
                    "next_del_page": 4294967295,
                    "offset": [
                        0
                    ],
                    "page_num": 3,
                    "record_size": 12,
                    "root_page": 1,
                    "ty": [
                        0
                    ]
                }
            ],
            "index_num": 2,
            "name": "nation",
            "p_key_index": 0,
            "primary": { // 主键信息
                "cols": [
                    0
                ],
                "name": "n_nationkey"
            },
            "record_num": 25, // 记录的条数
            "record_onepg": 199, // 每页可放的记录条数
            "record_size": 41, // 每条记录所占的字节数
            "table_id": 2 // 当前表在整个数据库中的index
        }
```

表中的 record 信息使用页式管理系统，存储在对应数据路文件夹下的`(table name).usid`文件中，如图所示，每个表都由page构成，每个page都由slot构成，每个文件的第一页记录了文件的信息，比如记录长度、页数、记录条数等。每一个page的页头用Bitmap保存了空slot的信息（因为每页能插入的slot的数目是有限的，所以Bitmap比较合适）。此处还有一个思路，就是用链表存储整个文件中有空slot的page，第一页存储该链表的头，这样就能灵活地找到可以插入的页，在删除操作中也能以很小的代价更新链表，而且相比Bitmap，这种方式不会被受页数的限制。



记录的增删改查使用的是 `recordId`，增是利用它计算得到在页式文件系统中对应的位置来存储，删是将对应的`recordId`置1，改是利用`recordId`修改对应的值，查是利用`recordId`计算所存储的位置然后返回结果。

2.3 索引管理模块

索引管理模块负责表的属性索引的创建、插入、删除，使用 B Tree 的数据结构加速查询。索引信息也存在每个数据库的所有表的头信息文件`database.udb`中：

```
"index": [
                {
                    "key": [ // 索引所建立的列的index
                        0
                    ],
                    "key_num": 1, // 列数
                    "name": "pk_n_nationkey", // 索引名
                    "next_del_page": 4294967295, // 下一个空页的
index,初始值 NULL 为-1
                    "offset": [  // 提取该列信息所需偏移
                        0
                    ],
                    "page_num": 3, // 已经使用的页数
                    "record_size": 12,// 存储索引所需字节数
                    "root_page": 1, // B树根节点对应页index
                    "ty": [  // 索引列的类型
                        0
                    ]
```

```
            },
            {
                "key": [
                    2
                ],
                "key_num": 1,
                "name": "fk_n_regionkey1",
                "next_del_page": 4294967295,
                "offset": [
                    0
                ],
                "page_num": 3,
                "record_size": 12,
                "root_page": 1,
                "ty": [
                    0
                ]
            }
        ],
```


索引就是B树的子节点，按页分配，存在`((table name)_(index name)).usid`中，每个节点包含父节点index，索引值个数等，然后用类似链表的逻辑存储孩子节点。索引管理模块的增删改查底层逻辑就是B树的逻辑。

2.4 系统管理模块

系统管理模块负责数据库和表的管理，同时也实现了主键和外键的创建删除，还有列的增删改操作。我将不同的数据库存在不同的文件夹下，每个数据库文件夹的结构如下所示：

```
testdb
|-- database.udb
|-- nation.usid
|-- nation_fk_n_regionkey1.usid
|-- nation_pk_n_nationkey.usid
```

其中数据库的所有表的头信息以 JSON 文件的形式存储在所在的数据库的文件夹下，以`database.udb`命名，表的内容以页式文件存储在对应数据库文件夹下的`(table name).usid`文件中，表的索引内容以页式文件存储在对应数据库文件夹下的`((table name)_(index name)).usid`文件中。

通过这种存储结果来存储数据库内容，对数据库的增删改查就只需要操作对应的本地文件夹即可，对于表的修改只需要先修改`database.udb`中的对应项，然后修改对应的`.usid`文件即可。对于表中列的操作，首先在表头文件中操作，若是增删列，因为采用连续存储的方式，需要重新生成`(table name).usid`文件，若有索引，还调用索引管理模块的函数来重新生成索引文件。

2.5 查询解析模块

查询解析模块负责解析用户的 SQL 指令，将它们对应为索引管理模块和系统管理模块的增删改查操作。这部分我使用了 flex 和 bison 包的 Lex/Yacc 工具。查询解析模块本质上是数据库的入口，读入并解析 SQL 命令，并为用户输出结果（或报错）。

因为对于每个查询命令，我都会建立一个临时的数据表`seldata`，对于聚集查询，我会对该临时数据表进行运算；对于嵌套查询，会先对最内部的查询建立临时数据表，外层的查询再查询它后再建立临时数据表。

3. 主要接口说明

查询解析模块调用记录管理模块、索引管理模块和系统管理模块的接口，主要接口存在于记录管理模块、索引管理模块和系统管理模块。

3.1 记录管理模块接口

由`Table.h`，`Table.cpp`组成

3.1.1 Table类

```
class Table {
    ...
public:
    // 类函数
    Table(Database* db, json j);
    ...
    ~Table();

    // 建表
    int createTable(uint table_id, Database* db, const char* name,
uint col_num, Type* col_ty, bool create = false);

    // 记录的增删改查
```

```
    int insertRecord(Any* data);
    int removeRecord(const int record_id);
    int queryRecord(const int record_id, Any* &data);
    int removeRecords(std::vector<WhereStmt> &where);
    int updateRecords(std::vector<Pia> &set, std::vector<WhereStmt>
&where);

    // 索引
    int findIndex(std::string s);
    int createIndex(std::vector<uint> col_id, std::string name);
    void removeIndex(uint index_id);

    // 列
    int createColumn(Type ty);
    int removeColumn(uint col_id);
    void updateColumns();

    // 主键与外键
    int createForeignKey(ForeignKey* fk, PrimaryKey* pk);
    int removeForeignKey(ForeignKey* fk);
    int createPrimaryKey(PrimaryKey* pk);
    int removePrimaryKey();
    int removePrimaryKey(PrimaryKey* pk);

    // 数据完整性检查
    int constraintCol(uint col_id);
    int constraintKey(Key* key);
    int constraintRow(Any* data, uint record_id, bool ck_unique);

    // 打印
    void print();
    void printCol();
};
```

3.2 索引管理模块接口

由`Node.h`，`Index.h`，`Index.cpp`组成

3.2.1 Index类

```
class Index {
public:
    // 类函数
    Index() {}
    Index(Table* table, const char* name, std::vector<uint> key,
int btree_max_per_node);
    Index(Table* table, json j);

    // B树加速查询
    Node* convert_buf_to_Node(int index);
    void convert_Node_to_buf(Node* node);
    void Node_remove(Node* node);

    // 记录增删查
    void insertRecord(Anys* info, int record_id);
    void removeRecord(Anys* info, int record_id);
    vector<int> queryRecord(Anys* info);
};
```

3.3 系统管理模块接口

由`Database.h`，`Database.cpp`组成

3.3.1 Database 类

```
class Database {
...
public:
    // 类函数
    Database(const char* name, bool create);
    ~Database();

    // 表的增删
    Table* createTable(const char* name, int col_num, Type*
col_ty);
    int deleteTable(const char* name);

    // 表的打印
    void showTables();
```

```
    int findTable(std::string s);

    // 创建查询的临时表
    void buildSel(uint idx, bool print = false);

     // 写回磁盘
    void update();
};
```

4. 实验结果

- 数据库的创建、删除
- 表的重命名 、创建、删除
- 列的重命名 、创建、删除
- 主外键的创建、删除、完整性维护
- 索引的创建与删除
- 嵌套查询
- 聚集查询
- 多表查询
- 相对用户友好的打印输出与报错

5. 小组分工

因为是一人组，所以所有模块都由不同时间段的洪昊昀完成qwq。

6. 参考文献

1. 《数据库大作业详细说明》
2. CS346 RedBase Project https://web.stanford.edu/class/cs346/2015/redbase.html
3. https://github.com/Konano/UselessDatabase
4. https://github.com/nlohmann/json
5. Lex/Yacc工具
6. https://github.com/duzx16/MyDB