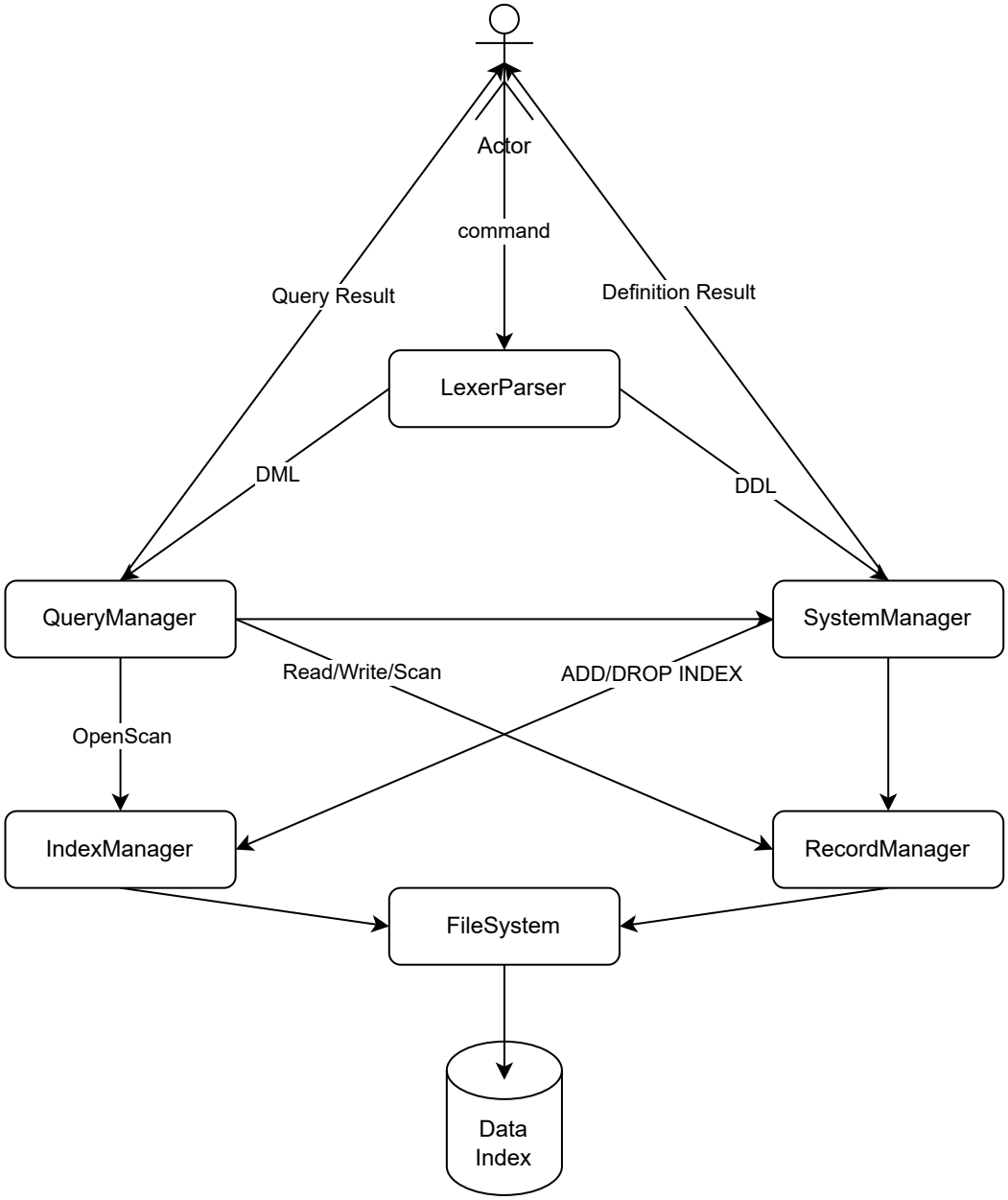
Database System Report

曹伦郗 *2020011020*

系统架构

系统主要由命令解析模块LexerParser（以下简称LP）、查询解析模块QueryManager（以下简称QM）、系统管理模块SystemManager（以下简称SM）、索引管理模块IndexManager（以下简称IM）、记录管理模块RecordManager（以下简称RM）、页式文件系统模块FileSystem（以下简称FS）组成。

具体架构情况如下图所示，与文档中给出的参考架构类似。



当用户传入命令，由LP将其解析，并调用QM或SM中对应操作函数。

- 对DML命令

对于删改查这类含条件子句的命令，QM会先检查操作合法性。而后，QM会检查该条DML命令的条件子句中是否存在可以利用索引加速查询的条件表达式（在我的实现中特指A.a op constant的条件表达式）。

若可以，QM则调用IM筛选出符合部分条件表达式的索引条目，获得其中的记录号（页号和槽号组成），调用RM找到对应记录号的记录，再检查其是否满足全部条件表达式，对满足条件的记录进行相应操作。

否则，QM直接调用RM扫描全表筛选出符合条件的记录，进行相应操作。

对于插入记录命令，QM检查完毕操作合法性后，分别调用IM和RM对该条记录进行插入。

最后，QM反馈操作结果。

- 对DDL命令

SM根据命令，修改系统META信息、数据库META信息、表META信息，并调用IM、RM等模块进行索引文件、记录文件的增删等操作，并反馈操作结果。

模块设计

页式文件系统模块FS

该模块我复用了课程组提供的页式文件系统。

记录管理模块RM

空闲页组织

我将空闲页在逻辑上以链表组织，文件头中保存了链表头所在页页号，每页中又保存下一空闲页页号，插入时优先使用链表头该页，维护方式如下。

- 每当一页槽位插满，则将链表头置为该页中保存的下一空闲页页号。
- 每当一页槽位从插满变为空闲出一个槽位，则将其作为新的链表头。
- 当链表头该页页号为0（即文件头所在页），代表不存在空闲页，则新获得一页作为链表头。

文件头

我以每个记录文件的首页作为文件头，包含当前使用的页面数、当前首个空闲页页号、记录槽位中NULL位图长度、记录槽位长度、每页最大槽位数、记录列数。在关闭记录文件时，若文件头发生改变，写回磁盘。

```
struct RM_FileHeader
{
    int _pageNum = 1;          // 0: meta page, 1~n: data page, contains meta
page
    int _firstFreePageID = 0; // the first of linked list of free pages
    int _nullLen;
    int _slotLen;
    int _maxSlotNum;
    int _attrNum;
};
```

页结构和槽结构

我设计的存储页的结构如下，以8字节保存了当前页已使用的槽位数、下一空闲页页号、若干槽位、槽位占用情况的位图。其中为方便寻址，位图从页面末尾开始，从高地址到低地址依次对应槽号从小到大。

```
Page Structrue: [ usedSlotNum | nextFreePageID | slots | existBitMap ]
```

每条槽位的结构如下。

```
slot: [ nullBitMap(nullLen) | column 1 | column 2 | ... | column n ]
```

功能

主要通过FS读或写磁盘上的记录文件，并在增删改记录后以及关闭当前打开的记录文件前将缓存页中的内容写回磁盘。

索引管理模块IM

参考项目

由于对B+树的实现掌握不够，在此处我参考了https://github.com/ZYFZYF/RoboDBMS中索引管理模块的B+树实现，并基于我所使用的页式文件系统完成了保存索引的基于磁盘的B+树。树中节点下保存的条目由键、页号、槽号三项成员组成，比较条目大小时，先比较键，再比较页号，最后比较槽号。

文件头

我以每个记录文件的首页作为文件头，包含索引键类型、索引键长、根节点页号、树中总节点数、单页最大条目数、下一空闲页页号，以及每个节点中键、页号、槽号这三项条目成员，以及子节点页号的页偏移。在关闭索引文件时，若文件头发生改变，也写回磁盘。

```
struct IDX_FileHeader
{
    ColumnType _keyType;
    int _keyLen;
    int _rootPageID; // the root of B+ tree
    int _nodeNum;    // the number of nodes in B+ tree
    int _maxKeyNum;  // the order of B+ tree
    int _nextFreePageID;
    // the offset from the start of the page of 4 fields
    int _keyOffset;
    int _pageIDOffset;
    int _slotIDOffset;
    int _childIDOffset;
};
```

节点结构

我设计的每个节点的页结构如下。

```
struct BTreeNode
{
    PageHeader _pHeader;
    Byte *pageBuf;
    // entry = (key, pageID, slotID, childID)
    Byte *_keys;
    int *_pageIDs;
    int *_slotIDs;
    int *_childIDs;
    // Page Structrue: [ isLeaf | prevPageID | mextPageID | dadPageID | keyNum |
keys | pageIDs | slotIDs | childIDs ]
    int index; // not stored in page
};
```

```
Page Structrue: [ PageHeader | keys | pageIDs | slotIDs | childIDs ]
```

其中，页头结构如下，主要包含节点的各种属性。

```
// IM
struct PageHeader
{
    bool _isLeaf;
    int _prevPageID;
    int _nextPageID;
    int _dadPageID;
    int _keyNum;
};
```

一些改进

其中，由于有时节点中条目会短暂超出限定最大条目数，为了避免覆写到后一内存单元中的内容，我在计算偏移量时，留出一个额外条目的空间，并将最大条目数设为理论值 - 1。

系统管理模块SM

META信息

系统META信息存放在system.meta文件中，包含了各数据库名称，只有当数据库系统关闭时其才会被更新。

数据库META信息存放在生成的对应.dbmeta文件中，包含了各表名称和各表的META信息，每当切换到下一数据库或修改了META信息时，将数据库的META信息写回。

表的META信息如下。

```
struct TableMeta
{
    char _name[MAX_NAME_LEN]{};
    int _columnNum;
    Column _columns[MAX_COL_NUM];
    Index _indexes[MAX_INDEX_NUM];
    PrimaryKey _primaryKey;
    ForeignKey _foreignKeys[MAX_FOREIGN_KEY_NUM];
```

```
    inline int nullByteLen() const
    {
        return (_columnNum + 7) / 8;
    }
};
```

功能

SM本身支持的执行基础部分的DDL命令功能主要涉及调用RM、IM进行相关记录文件、索引文件的增删，以及对数据库中表的META信息的修改。

除此之外，SM还为QM提供了在插入记录时根据插入命令生成新记录，在增删改时检查是否满足约束条件，检查记录是否满足某个条件表达式的功能。

- 在插入记录时，SM为QM检查新记录是否存在重复主键值、外键引用值不存在两个问题。
- 在删除记录时，SM为QM检查所删记录的主键值是否被其他表的外键引用。
- 在更新记录时，SM为QM检查所更新记录的所更新列中，是否存在主键值被其他表的外键引用却被修改、新的主键值是否重复、外键引用了不存在的主键值等问题。

查询管理模块QM

条件子句

对于删改查这类含条件子句的命令，QM会先检查操作合法性（其中包括检查表名、列名正确性，若是删改命令，还需调用SM检查是否满足约束）。

而后，QM会检查该条DML命令的条件子句中是否存在可以利用索引加速查询的条件表达式（在我的实现中特指A.a op constant的条件表达式）。

若可以，QM则调用IM筛选出符合部分条件表达式的索引条目，获得其中的记录号（页号和槽号组成），调用RM找到对应记录号的记录，再检查其是否满足全部条件表达式，对满足条件的记录进行相应操作。

否则，QM直接调用RM扫描全表筛选出符合条件的记录，进行相应操作。

双表连接

而双表JOIN查询命令和单表查询命令的逻辑类似。我实现的是嵌套循环查询。

对于SELECT * FROM A, B WHERE命令，我会先检查条件子句中是否存在A.a op constant的条件表达式（且表A在列a上建立了索引），按上述逻辑获得记录，再检查其是否满足所有的只涉及A的条件表达式。以满足条件的表A记录作为外层循环。

同理，我也会对表B作一遍同样的操作，以满足条件的表B记录作为内存循环。再通过双层循环筛选出满足所有条件表达式的记录。

插入命令

对于插入记录命令，QM检查完毕操作合法性（其中包括检查表名、列名正确性，调用SM检查是否满足约束）后，分别调用IM和RM对该条记录进行插入。

命令解析模块LP

工具

我使用了Flex&Bison工具生成了词法分析器lexer和语法分析器parser。主要的工作在于设置lexer中保留字和标识符、VARCHAR类型字面量、其余符号优先级的安排，以及在parser中设计文法，和在DML和DDL命令的产生式的语义动作中正确调用QM和SM的对应功能函数。

表达式

为了方便同记录比较，我将条件子句的条件表达式、设置子句的子表达式、选择子句中的列名表达式统一抽象为表达式Expr类。叶表达式为常数值和列名，根表达式由一个列名表达式和常数值表达式组成（实际使用中最多只存在两层的树）。

```
struct Expr
{
    Expr *_l;
    Expr *_r;
    CmpOp _op; // when Expr is used for setting, op 'EQ' means assigning
    bool _isVal;
    bool _isColumn;
    // isVal
    AttrVal _val;
    // isColumn
    std::string _tbName;
    std::string _colName;
}
```

其中，常数值类定义如下。

```
struct AttrVal
{
    ColumnType _type = INT;
    bool _isNull = false;
    int _intVal = 0;
    float _floatVal = 0;
    char *_varcharVal = nullptr;
    int _varcharLen = 0;

    void _setVarcharVal(const char *varchar, int len);

    const Byte *_val() const;

    int _len() const;
};
```

接口说明

RM

Handler

```
class RecordHandler
{
private:
    // 查询位图
```

```
    inline bool _getBit(Byte *bitMap, int idx) const;
    // 设置指定位为1
    inline void _setBitOne(Byte *bitMap, int idx) const;
    // 设置指定位为0
    inline void _setBitZero(Byte *bitMap, int idx) const;
    // 获取首个为0的位
    bool _getFirstZeroBit(Byte *bitMap, int &idx) const;
    // 获取下一个为1的位
    bool _getNextOneBit(Byte *bitMap, int startIdx, int &idx) const;

public:
    int _fileID;
    BufPageManager *_bpm;
    FileManager *_fm;
    RM_FileHeader _fHeader;
    bool _fHeaderChange;

    // for Scanner
    // 获取下一条记录
    bool _getNextRec(int startPageID, int startSlotID, RID &rid, Byte *dataDst)
const;
    // 获取某页的记录数
    bool _getRecNum(int pageID, int &recNum) const;
    // 初始化
    void init(int fileID, BufPageManager *bpm, FileManager *fm);
    // 根据RID获取记录
    bool accessRec(const RID &rid, Byte *dataDst) const;
    // 插入记录
    bool insertRec(const Byte *dataSrc, RID &rid);
    // 删除记录
    bool deleteRec(const RID &rid);
    // 更新记录
    bool updateRec(const RID &rid, const Byte *dataSrc) const;
};
```

Scanner

```
class RecordScanner
{
private:
    RecordHandler *_recHandler;
    // start finding at (_startPageID, _startSlotID)
    int _startPageID;
    int _startSlotID;
    Byte *_recData;

public:
    // 打开扫描器
    void openScan(RecordHandler *recHandler);
    // 获取下一条记录
    bool getNextRecord(RID &rid, Byte *data);
    // 关闭扫描器
    bool closeScan() const;
};
```

Manager

```
class RecordManager
{
private:
    BufPageManager *_bpm;
    FileManager *_fm;

public:
    RecordManager(BufPageManager *bpm, FileManager *fm) : _bpm(bpm), _fm(fm) {}
    // 创建记录文件
    bool createRecFile(const char *fileName, int recLen, int attrNum) const;
    // 删除记录文件
    bool deleteRecFile(const char *fileName) const;
    // 打开记录文件
    bool openRecFile(const char *fileName, RecordHandler &recHandler) const;
    // 关闭记录文件
    bool closeRecFile(RecordHandler &recHandler) const;
};
```

IM

Handler

```
class IndexHandler
{
private:
    // 比较函数
    bool _compare(CmpOp op, const Byte *key1, int pageID1, int slotID1,
                  const Byte *key2, int pageID2, int slotID2) const;
    // 将某节点页从磁盘加载到内存中
    BTreeNode *_loadNode(int nodePageID) const;
    // 设置某节点的某位置条目
    void _setEntry(BTreeNode *node, int idx, const Byte *key, int pageID, int
slotID, int childID = NULL_NODE) const;
    // 将某节点页写回磁盘
    void _writeNodeBack(BTreeNode *node, int nodePageID) const;
    // 将源节点的若干条目复制到目的节点
    void _moveEntries(BTreeNode *dstNode, int dstIdx, BTreeNode *srcNode, int
srcIdx, int n) const;
    // 是否上溢
    inline bool _overflow(int keyNum) const;
    // 是否下溢
    inline bool _underflow(int keyNum) const;
    // 平均左右两节点的条目数
    void _average(BTreeNode *leftNode, int leftNodePageID, BTreeNode *rightNode,
int rightNodePageID) const;
    // 处理上溢
    void _handleOverflow(BTreeNode *node, int nodePageID);
    // 处理下溢
    void _handleUnderflow(BTreeNode *node, int nodePageID);
    // 在某节点中插入条目
    void _insertEntryInNode(BTreeNode *node, int nodePageID, int idx, int
pageID, int slotID, const Byte *key, BTreeNode *childNode, int childID);
    // 在某节点中删除条目
```

```
    void _deleteEntryInNode(BTreeNode *node, int nodePageID, int idx);

public:
    int _fileID;
    BufPageManager *_bpm;
    FileManager *_fm;
    IDX_FileHeader _fHeader;
    bool _fHeaderChange;

    // for Scanner
    // 获取第一条条目中的键
    bool _getFirstEntry(int &targetPageID, int &idx, Byte *firstKey) const;
    // 获取下一条条目中的键
    bool _getNextEntry(int &targetPageID, int &idx, Byte *nextKey) const;
    // 获取某位置条目中的RID
    void _getRID(int nodePageID, int idx, RID &rid) const;
    // 初始化
    void init(int fileID, BufPageManager *bpm, FileManager *fm);

    // when insert == false, find the max entry <= (key, rid), may be not
existing (idx = -1)
    // when insert == true, find which idx (key, rid) should be
    // 搜索条目
    void findEntry(const RID &rid, const Byte *key, bool insert, int &idx, int
&targetPageID, bool &last, Byte *getKey = nullptr) const;
    // 插入条目
    bool insertEntry(const RID &rid, const Byte *key);
    // 删除条目
    bool deleteEntry(const RID &rid, const Byte *key);
};
```

Scanner

```
class IndexScanner
{
private:
    IndexHandler *_idxHandler;
    bool _firstTime;
    bool _noMore; // no more matched entry
    Byte *_cmpKey;
    // leafNode's _idx key is _findKey (when _idx is legal)
    int _leafPageID;
    int _idx;
    Byte *_findKey;
    CmpOp _op;

public:
    // to find key op cmpKey is true
    // 打开搜索某条件的扫描器
    void openScan(IndexHandler *idxHandler, const Byte *cmpKey, CmpOp op);
    // 获取下一条条目中的RID
    bool getNextEntry(RID &rid);
    // 关闭扫描器
    bool closeScan() const;
};
```

Manager

```
class IndexManager
{
private:
    BufPageManager *_bpm;
    FileManager *_fm;

public:
    IndexManager(BufPageManager *bpm, FileManager *fm) : _bpm(bpm), _fm(fm) {}
    // 创建索引文件
    bool createIndexFile(const char *fileName, int idxOfIndex, ColumnType
attrType, int attrLen) const;
    // 删除索引文件
    bool destroyIndexFile(const char *fileName, int idxOfIndex) const;
    // 打开索引文件
    bool openIndexFile(const char *fileName, int idxOfIndex, IndexHandler
&idxHandler) const;
    // 关闭索引文件
    bool closeIndexFile(IndexHandler &idxHandler) const;
};
```

SM

```
class SystemManager
{
public:
    BufPageManager *_bpm;
    FileManager *_fm;
    RecordManager *_rm;
    IndexManager *_im;
    SystemMeta _sysMeta;
    int _sysFileID;
    bool _usingDB;
    DBMeta _DBMeta;      // meta info of DB in use
    std::string _DBName; // name of DB in use

    // expr like A.a op val, A.a op A.b
    // 检查只涉及单表的记录是否满足条件表达式
    bool _satisfy(const Expr &condExpr, int tbID, const int *offsets, const Byte
*recData) const;
    // 检查涉及双表的记录是否满足条件表达式
    // expr like A.a op B.b
    bool _satisfy(const Expr &condExpr, int lTbID, int rTbID, const int
*lOffsets, const int *rOffsets,
                  const Byte *lRecData, const Byte *rRecData) const;
    // 获取某表在数据库中的ID
    int _getTableID(const char *tableName) const;

    // must make sure this table exists
    // 获取某列在表中的ID
    int _getColumnID(int tbID, const char *columnName) const;
    // 将数据库META信息从磁盘加载到内存
    bool _loadDBMeta(const char *DBName);
    // 将系统META信息写回
```

```
    void _writeBackSysMeta() const;
    // 将数据库META信息写回
    void _writeBackDBMeta() const;

    // for QM
    // 检查某值是否在某表的主键中
    bool _foundInPrimaryKey(const Byte *cmpKey, int priTbID) const;
    // 检查某值是否在某表的某外键中
    bool _foundInForeignKey(const Byte *cmpKey, const char *forTbName, int
forTbID, int forColumnID) const;

    // must make sure this table exists
    // 检查插入操作是否满足约束
    bool _checkConstraintWhenInsert(const Byte *recData, int tbID) const;

    // must make sure this table exists
    // 检查删除操作是否满足约束
    bool _checkConstraintWhenDelete(const char *tableName, int tbID, const Byte
*recData, bool callByUpdate = false) const;

    // must make sure this table exists
    // 检查更新操作是否满足约束
    bool _checkConstraintWhenUpdate(const char *tableName, int tbID, const Byte
*oldRecData, const Byte *newRecData, std::vector<int> *columnIDs) const;

    // must make sure this table exists
    // 获取某表的记录长度
    int _getRecLen(int tbID) const;

    // must make sure this table exists
    // 根据给出的一系列值设置一条新记录
    bool _setRecData(Byte *recData, int tbID, const std::vector<Expr> *valList,
bool safe = false) const;

    // keep opening system.meta
    // 构造函数，将系统META信息从磁盘加载到内存
    SystemManager(BufPageManager *bpm, FileManager *fm, RecordManager *rm,
IndexManager *im)
        : _bpm(bpm), _fm(fm), _rm(rm), _im(im), _usingDB(false)
    {
        if (!_fm->openFile(sysMetaFileName, _sysFileID))
        {
            _fm->createFile(sysMetaFileName);
            _fm->openFile(sysMetaFileName, _sysFileID);
        }
        int index;
        BufType pageBuf = _bpm->getPage(_sysFileID, 0, index);
        memcpy(&_sysMeta, pageBuf, sizeof(SystemMeta));
        _bpm->access(index);
    }
    // 析构函数，将系统META信息写回
    ~SystemManager()
    {
        _writeBackSysMeta();
    }
```

```
    // system
    // 展示所有数据库
    void showDBs() const;

    // DB
    // 创建数据库
    bool createDB(const char *DBName);
    // 删除数据库
    bool dropDB(const char *DBName);
    // 使用数据库
    bool useDB(const char *DBName);
    // 显示当前使用的数据库
    void showUse() const;
    // 展示所有表
    bool showTables() const;

    // Table
    // 创建表
    bool createTable(const char *tableName, const std::vector<Column>
*columnList);
    // 删除表
    bool dropTable(const char *tableName, bool callByDropTable = false);
    // 展示表信息
    bool descTable(const char *tableName) const;

    // Index
    // 建立索引
    bool addIndex(const char *tableName, const char *attrName);
    // 删除索引
    bool dropIndex(const char *tableName, const char *attrName);

    // Key
    // 建立主键
    bool addPrimaryKey(const char *tableName,
                       const char *attrName,
                       bool callByCreateTb = false);
    // 删除主键
    bool dropPrimaryKey(const char *tableName);
    // 建立外键
    bool addForeignKey(const char *name,
                       const char *forTbName, const char *forAttr,
                       const char *priTbName, const char *priAttr,
                       bool callByCreateTb = false);
    // 删除外键
    bool dropForeignKey(const char *tableName, const char *forKeyName);
};
```

QM

```
class QueryManager
{
private:
    SystemManager *_sm;
```

```
    // expr like A.a = val
    // 检查设置子句是否合法
    bool _checkSetList(const char *tableName, int tbID, const std::vector<Expr>
*setList,
                       std::vector<int> *columnIDs) const;

    // expr like A.a
    // 检查选择子句是否合法
    bool _checkColumnList(const char *tableName, int tbID, const
std::vector<Expr> *columnList,
                          std::vector<int> *columnIDs) const;

    // expr like A.a, B.b
    // 检查双表查询的选择子句是否合法
    bool _checkColumnListWhenJoin(const char *tableNameA, const char
*tableNameB,
                                  int tbAID, int tbBID, const std::vector<Expr>
*columnList) const;

    // expr like A.a op val, A.a op A.b
    // 检查条件子句
    bool _checkCondList(const char *tableName, int tbID, const std::vector<Expr>
*condList) const;

    // expr like A.a op B.b, B.b op A.a, A.a op A.b, B.a op B.b, A.a op val, B.b
op val
    // 检查双表查询的条件子句
    bool _checkCondListWhenJoin(const char *tableNameA, const char *tableNameB,
                                int tbAID, int tbBID, const std::vector<Expr>
*condList) const;

    // deal with cond expr like A.a op val, A.a op A.b
    // if callByJoin, it means condList may contain B.b op val, B.a op B.b, B.a
op A.a
    // 筛选器
    std::vector<RID> _filter(const char *tableName, int tbID,
                             const std::vector<Expr> *condList, bool callByJoin
= false) const;

    // cond expr expr like A.a op B.b, B.b op A.a, A.a op A.b, B.a op B.b, A.a op
val, B.b op val
    // column expr like A.a
    // outer loop A, inner loop B
    // 筛选内层表，进行双循环后再筛选最终结果
    std::vector<Byte *> *_join(const char *tableNameA, const char *tableNameB,
int tbAID, int tbBID,
                               const std::vector<Expr> *columnList, const
std::vector<Expr> *condList,
                               const std::vector<Byte *> *recDataAs, TableMeta
&resultTbMeta) const;
    // 获取某表的各列的偏移量
    int *_getOffsets(const TableMeta &tbMeta) const;
    // 展示一条记录
    void _showRec(const TableMeta &tbMeta, const int *offsets, const Byte
*recData,
```

```
                  const std::vector<int> *columnIDs) const;

public:
    QueryManager(SystemManager *sm) : _sm(sm) {}
    // 插入一条记录
    bool insertRec(const char *tableName, const std::vector<Expr> *valList, bool
safe = false) const;
    // 批量插入合法的记录
    bool safeInsert(const char *tableName, const std::vector<std::vector<Expr>>
*valLists) const;
    // 删除记录
    bool deleteRec(const char *tableName, const std::vector<Expr> *condList)
const;
    // 更新记录
    bool updateRec(const char *tableName,
                   const std::vector<Expr> *setList,
                   const std::vector<Expr> *condList) const;
    // 单表或双表查询记录
    bool selectRec(bool join, const char *tableNameA, const char *tableNameB,
                   const std::vector<Expr> *columnList,
                   std::vector<Expr> *condList) const;
};
```

实验结果

testdb

我设计的SQL文件内容如下，主要检查了约束是否起效，输出正确。

```
create database testdb;

use testdb;

create table tb1(
    pk INT not null,
    fp float,
    nm float,
    sl INT,
    primary key (pk)
);

create table tb2(
    pkk INT not null,
    fp float,
    nm float,
    sll INT,
    primary key (pkk),
    foreign key (sll) references tb1 (pk)
);

insert into
    tb1
values
    (77, 7.7, 0.77, 80),
    (55, 5.5, 0.55, 50),
```

```
    (33, 3.3, 0.33, 30);

insert into
    tb2
values
    (88, 8.8, 0.88, 33),
    (66, 6.6, 0.66, 77),
    (65, 6.6, 0.66, 77),
    (64, 6.6, 0.66, 0),
    (44, 4.4, 0.44, 33);

select
    *
from
    tb1,
    tb2
where
    tb1.pk <= tb2.sll
    and tb1.pk < 77
    and tb2.pkk > 44
    and tb2.sll >= tb2.pkk
    and tb1.pk >= tb1.sl;

delete from
    tb2
where
    pkk < 66
    and pkk > 64;

select
    *
from
    tb1,
    tb2
where
    tb1.pk <= tb2.sll
    and tb1.pk < 77
    and tb2.pkk > 44
    and tb2.sll >= tb2.pkk
    and tb1.pk >= tb1.sl;

update
    tb2
set
    pkk = 65,
    sll = 100
where
    pkk > 65
    and pkk < 88;

update
    tb1
set
    sl = -100
where
```

```
    pk = 33;

update
    tb1
set
    pk = -100
where
    pk = 33;

update
    tb1
set
    pk = 111
where
    pk = 55;

delete from
    tb1
where
    pk < 78
    and pk > 76;

select
    *
from
    tb1,
    tb2
where
    tb1.pk <= tb2.sll
    and tb1.pk < 77
    and tb2.pkk > 44
    and tb2.sll >= tb2.pkk
    and tb1.pk >= tb1.sl;

alter table
    tb2
add
    index (sll);

delete from
    tb1
where
    pk > 32
    and pk <= 55;

insert into
    tb2
values
    (88, 8.8, 0.88, 111);

insert into
    tb2
values
    (89, 8.8, 0.88, 111);
```

```
insert into
    tb2
values
    (89, 8.8, 0.88, 0);

insert into
    tb2
values
    (90, 8.8, 0.88, 33);

quit;
```

输出

```
  Create DB well.


 Use DB well.









Create table well.
Spend 0.005431s









Create table well.
Spend 0.005413s









       Successfully insert 3 record of 3.
Spend 0.023434s
```

```
       No same attr on referred table's primary key is unacceptable.
Successfully insert 4 record of 5.
Spend 0.047273s









       Can use index for filter.
Can use index for inner filter.
       | tb1.pk           |tb1.fp           |tb1.nm           |tb1.sl
|tb2.pkk          |tb2.fp           |tb2.nm           |tb2.sll          |
1      | 33               |3.30             |0.33             |30
|65               |6.60             |0.66             |77               |
2      | 55               |5.50             |0.55             |50
|65               |6.60             |0.66             |77               |
3      | 33               |3.30             |0.33             |30
|66               |6.60             |0.66             |77               |
4      | 55               |5.50             |0.55             |50
|66               |6.60             |0.66             |77               |
Spend 0.012316s




       Can use index for filter.
Delete in rec well.
Successfully delete 1 record.
Spend 0.009940s









       Can use index for filter.
Can use index for inner filter.
```

```
       | tb1.pk           |tb1.fp           |tb1.nm           |tb1.sl
|tb2.pkk          |tb2.fp           |tb2.nm           |tb2.sll          |
1      | 33               |3.30             |0.33             |30
|66               |6.60             |0.66             |77               |
2      | 55               |5.50             |0.55             |50
|66               |6.60             |0.66             |77               |
Spend 0.012531s









       Can use index for filter.
No same attr on referred table's primary key is unacceptable.
Successfully update 0 record.
Spend 0.012200s









      Can use index for filter.
Update well.
Successfully update 1 record.
Spend 0.007099s









      Can use index for filter.
Updating primary key when it has references is unacceptable.
Successfully update 0 record.
Spend 0.009793s









      Can use index for filter.
Update well.
Successfully update 1 record.
Spend 0.014523s
```

```
       Can use index for filter.
Deleting when primary key has references is unacceptable.
Successfully delete 0 record.
Spend 0.010137s
```

```
       Can use index for filter.
Can use index for inner filter.
       | tb1.pk           |tb1.fp           |tb1.nm           |tb1.sl
|tb2.pkk          |tb2.fp           |tb2.nm           |tb2.sll          |
1      | 33               |3.30             |0.33             |-100
|66               |6.60             |0.66             |77               |
Spend 0.012069s
```

```
     Add index well.
Spend 0.005491s
```

```
       Can use index for filter.
Deleting when primary key has references is unacceptable.
Successfully delete 0 record.
Spend 0.009763s
```

```
       Duplicate attr on primary key is unacceptable.
Successfully insert 0 record of 1.
Spend 0.002299s
```

```
       Successfully insert 1 record of 1.
Spend 0.011895s




       Duplicate attr on primary key is unacceptable.
Successfully insert 0 record of 1.
Spend 0.002329s




       Successfully insert 1 record of 1.
Spend 0.012030s
```

DATASET

在提供的DATASET数据库上测试用SQL文件内容如下，主要用于检查较大数据量下能否正常工作，输出正确。

```sql
USE DATASET;

SELECT
    S_NAME
FROM
    SUPPLIER
WHERE
    S_SUPPKEY <= 2049
    AND S_NAME > 'Supplier#000002045'
    AND S_NAME < 'Supplier#000002052';

SELECT
    S_NAME
FROM
    SUPPLIER
WHERE
    S_SUPPKEY < 2049
    AND S_NAME > 'Supplier#000002045'
    AND S_NAME < 'Supplier#000002052';

SELECT
    S_NAME
FROM
    SUPPLIER
WHERE
    S_SUPPKEY > 2049
    AND S_NAME > 'Supplier#000002045'
```

```
    AND S_NAME < 'Supplier#000002052';

SELECT
    S_NAME
FROM
    SUPPLIER
WHERE
    S_SUPPKEY >= 2049
    AND S_NAME > 'Supplier#000002045'
    AND S_NAME < 'Supplier#000002052';

SELECT
    S_NAME
FROM
    SUPPLIER
WHERE
    S_SUPPKEY = 2049
    AND S_NAME > 'Supplier#000002045'
    AND S_NAME < 'Supplier#000002052';

SELECT
    S_NAME
FROM
    SUPPLIER
WHERE
    S_SUPPKEY <> 2049
    AND S_NAME > 'Supplier#000002045'
    AND S_NAME < 'Supplier#000002052';

SELECT
    P_SIZE,
    P_CONTAINER,
    P_RETAILPRICE
FROM
    PART
WHERE
    P_PARTKEY > 59990
    AND P_PARTKEY <> 59995
    AND P_SIZE <> 13;

SELECT
    PART.P_SIZE,
    SUPPLIER.S_PHONE
FROM
    PART,
    SUPPLIER
WHERE
    SUPPLIER.S_SUPPKEY <> 2049
    AND SUPPLIER.S_NAME > 'Supplier#000002045'
    AND SUPPLIER.S_NAME < 'Supplier#000002052'
    AND PART.P_PARTKEY > 59990
    AND PART.P_PARTKEY <> 59995
    AND PART.P_SIZE <> 13;
```

输出

```
 Use DB well.
```

```
       Can use index for filter.
       | S_NAME           |
1      | Supplier#0000020 |
2      | Supplier#0000020 |
3      | Supplier#0000020 |
4      | Supplier#0000020 |
Spend 0.113310s
```

```
       Can use index for filter.
       | S_NAME           |
1      | Supplier#0000020 |
2      | Supplier#0000020 |
3      | Supplier#0000020 |
Spend 0.119180s
```

```
       Can use index for filter.
       | S_NAME           |
1      | Supplier#0000020 |
2      | Supplier#0000020 |
Spend 0.114262s
```

```
       Can use index for filter.
       | S_NAME           |
1      | Supplier#0000020 |
2      | Supplier#0000020 |
3      | Supplier#0000020 |
Spend 0.109722s
```

```
       Can use index for filter.
       | S_NAME           |
1      | Supplier#0000020 |
Spend 0.008261s
```

```
       Can use index for filter.
       | S_NAME           |
1      | Supplier#0000020 |
2      | Supplier#0000020 |
3      | Supplier#0000020 |
4      | Supplier#0000020 |
5      | Supplier#0000020 |
Spend 0.167181s
```

```
       Can use index for filter.
       | P_SIZE           |P_CONTAINER      |P_RETAILPRICE    |
1      | 32               |MED PKG          |1950.99          |
2      | 19               |JUMBO BAG        |1951.99          |
3      | 25               |JUMBO PKG        |1952.99          |
4      | 40               |MED PACK         |1953.99          |
```

```
5       | 3                 |MED CASE          |1955.99           |
6       | 37                |JUMBO PKG         |1956.99           |
7       | 16                |SM CAN            |1957.99           |
8       | 40                |LG BOX            |960.00            |
Spend 1.453344s
```

```
       Can use index for filter.
Can use index for inner filter.
       | PART.P_SIZE       |S.S_PHONE         |
1      | 32                |31-837-215-6004   |
2      | 19                |31-837-215-6004   |
3      | 25                |31-837-215-6004   |
4      | 40                |31-837-215-6004   |
5      | 3                 |31-837-215-6004   |
6      | 37                |31-837-215-6004   |
7      | 16                |31-837-215-6004   |
8      | 40                |31-837-215-6004   |
9      | 32                |10-262-377-2302   |
10     | 19                |10-262-377-2302   |
11     | 25                |10-262-377-2302   |
12     | 40                |10-262-377-2302   |
13     | 3                 |10-262-377-2302   |
14     | 37                |10-262-377-2302   |
15     | 16                |10-262-377-2302   |
16     | 40                |10-262-377-2302   |
17     | 32                |10-939-898-3098   |
18     | 19                |10-939-898-3098   |
19     | 25                |10-939-898-3098   |
20     | 40                |10-939-898-3098   |
21     | 3                 |10-939-898-3098   |
22     | 37                |10-939-898-3098   |
23     | 16                |10-939-898-3098   |
24     | 40                |10-939-898-3098   |
25     | 32                |20-432-928-3484   |
26     | 19                |20-432-928-3484   |
27     | 25                |20-432-928-3484   |
28     | 40                |20-432-928-3484   |
29     | 3                 |20-432-928-3484   |
30     | 37                |20-432-928-3484   |
31     | 16                |20-432-928-3484   |
32     | 40                |20-432-928-3484   |
33     | 32                |23-305-823-9298   |
```

```
34      | 19                |23-305-823-9298  |
35      | 25                |23-305-823-9298  |
36      | 40                |23-305-823-9298  |
37      | 3                 |23-305-823-9298  |
38      | 37                |23-305-823-9298  |
39      | 16                |23-305-823-9298  |
40      | 40                |23-305-823-9298  |
Spend 1.821832s
```

小组分工

我单人完成了该项目。

参考文献

1. 学长项目https://github.com/ZYFZYF/RoboDBMS
2. Stanford CS346https://web.stanford.edu/class/cs346/2015/redbase