

实验 3 SPJ 算法

小组成员:吴天贞 陈师哲 周梦溪 蓝玮毓 张文慧

实验目的:

实现SPJ算法

选择操作算法

Table Scan, Index Scan

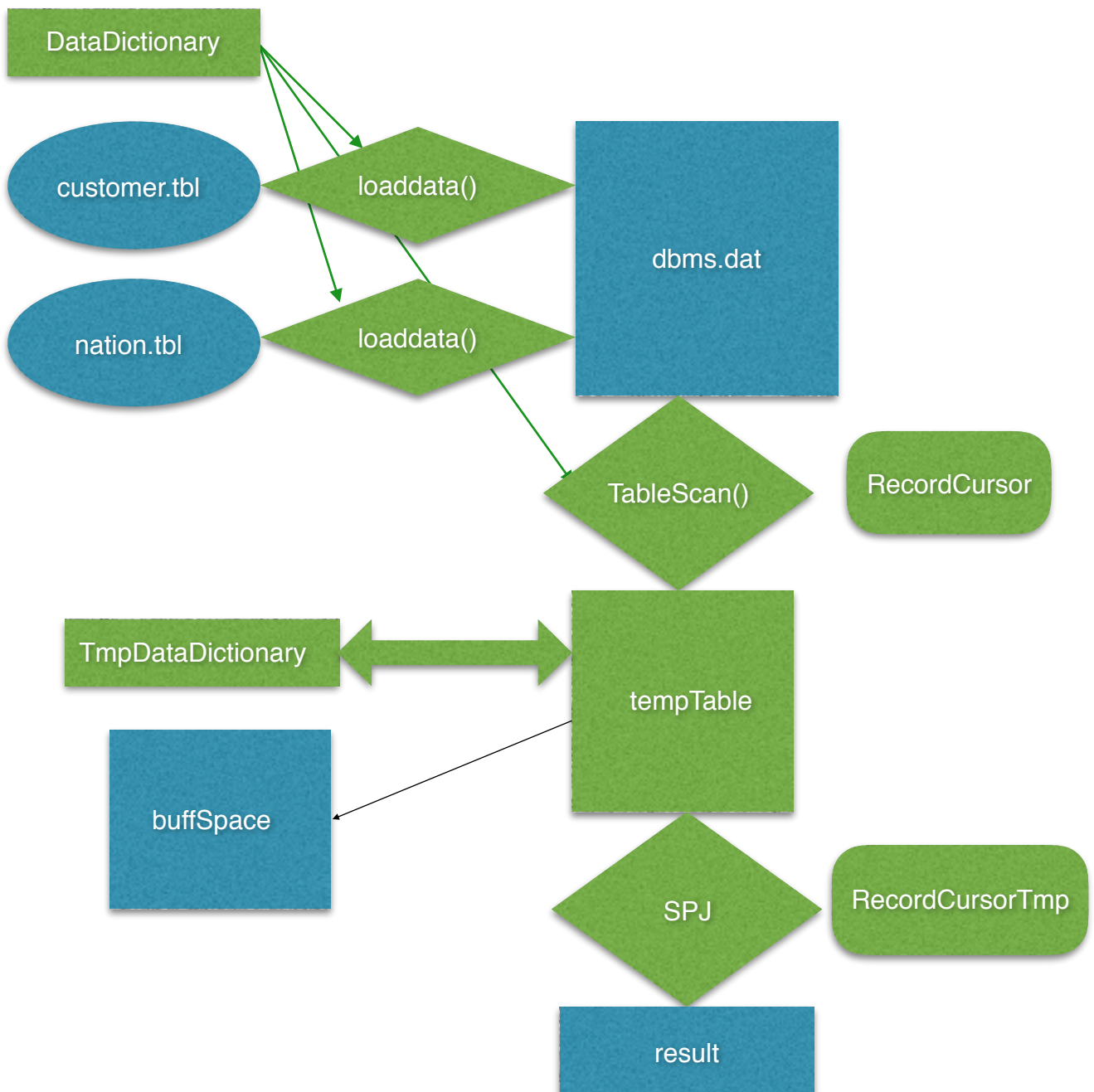
投影操作算法

连接操作算法

嵌套循环连接, 基于排序的等值连接, 基于散列的等值连接, 基于索引的连接

实验设计:

1. 整体框架



2. 数据结构

文件

数据库系统头部

```
struct dbSysHead
{
    struct SysDesc desc;
    struct buffSpace buff[3]; // 3 buffers for SPJ operations
    unsigned long *bitMap;
    relation redef[MAX_FILE_NUM]; //关系数据字典
    FILE *fpdesc;
};
```

缓冲区

```
struct buffSpace{
    char data[SIZE_BUFF][SIZE_PER_PAGE];
    struct buffMap map[SIZE_BUFF];
    long curTimeStamp;
    bool emptyOrnot;
    //true for empty, false for in use
}
```

数据字典

表

```
class relation
{
public:
    relation();
    relation(relation& RR);
    ~relation();
    int initRelation(struct dbSysHead *head, int fid, char
*relationName, char *constructorName);
    int init(char *relationName, char *constructorName);
    int changeRecordNum(int num);
    int insertAttribute(char *name, int type, int length);
    char *getRelationName();
    char *getConstructor();
    int getAttributeNum();
    int getRecordLength();
    int getRecordNum();
    attribute getAttributeByName(char *name);
    attribute getAttributeByNo(int no);
public:
    long fileID;
private:
    char relationName[NAMELENGTH]; //关系名
    char constructor[NAMELENGTH]; //建立者
    int attributeNum; //属性个数
    int recordLength; //记录长度
    int recordNum; //记录个数
    attribute atb[ATTRIBUTENUM]; //属性表
```

```

属性
class attribute
{
public:
    attribute();
    attribute(attribute& AA);
    ~attribute();
    int initAttribute(char *name, int type, int length, int deviation);
    char *getName();
    int getType();
    int getLength();
    int getRecordDeviation();

private:
    char attributeName[NAMELENGTH]; //属性名
    int type; //属性类型
    int length; //属性长度
    int recordDeviation; //记录内偏移
};

```

3. SPJ算法

3.1 Scan

对外存文件（表1，表2.....）进行顺序扫描读入内存（buffspace中的 char data[SIZE_BUFF][SIZE_PER_PAGE]），生成临时表。

TableScan

参数：文件号（逻辑的表号），临时数据字典数组

思路：读取传入临时表的类是RecordCorsor，用RecordCorsor的getNextRecord()函数遍历临时表，生成新临时表。新临时表存放在一个emptyOrnot为true的buffer中。

调用TableScan的例子：

```
TableScan(&head, FIRST_FID, temp_data_dict);
```

IndexScan：直接在Select里实现

3.2 Select

对上一个操作符生成的临时表进行选择，生成新临时表和新临时数据字典。

参数：临时数据字典数组（传入旧数据字典，生成新数据字典），列名，比较符号和比较值。

思路：读取传入临时表的类是RecordCorsorTmp，用RecordCorsorTmp的getNextRecord()函数遍历临时表，根据旧临时数据字典找到该属性值，与要比较的值进行比较，如果该行满足条件，则将该行用Buffer类一页一页地写入临时表中。新临时表存放在一个emptyOrnot为true的buffer中。再把旧临时表的bufferID置为true，表示旧临时表被释放。

tableScanEqualFilter 等值选择：实现了整型，字符串，日期型的等值选择。

indexScanEqualFilter 基于索引扫描的等值选择

tableScanSemiscopeFilter 条件选择：实现了整型的4种比较条件，和字符串的like

tableScanScopeFilter 范围选择

调用选择的例子：

```
ableScanEqualFilter(&head, FIRST_FID, temp_data_dict, "custkey", "3", &temp_data_dict[5])
```

```
tableScanEqualFilter(&head,FIRST_FID,temp_data_dict,"name","Customer#000000009",&temp_data_dict[5])
tableScanScopeFilter(&head,FIRST_FID,temp_data_dict,"custkey","220",NOT_MORE_THAN,"230",LESS_THAN,&temp_data_dict[5])
```

3.3 Project

对上一个操作符生成的临时表进行投影，生成新临时表和新数据字典。

参数：旧数据字典，新数据字典（包含了要投影的列名）。

思路：读取传入临时表的类是RecordCorsorTmp，用RecordCorsorTmp的getNextRecord()函数遍历临时表，根据新旧临时数据字典找到相同的字段（列名相同），逐行比较，将满足条件的属性生成的新一行，用Buffer类一页一页地写入临时表中。新临时表存放在一个emptyOrnot为true的buffer中。再把旧临时表的bufferID置为true，表示旧临时表被释放。

调用投影的例子

```
relation result;
result.init("customer", "TianzhenWu");
result.insertAttribute("name", 2, 64);
result.insertAttribute("phone", 2, 64);
project(&head, &temp_data_dict[0], &result);
```

3.4 Join