1. 数据说明

我们采用如下三个表:

customer表, nation表, region表

数据类型

Integer (-2,147,483,646 , 2,147,483,647)

Fixed text, size N

variable text, size <= MAX\_LENGTH

Date YYYY-MM-DD

|  |  |  |
| --- | --- | --- |
| Column Name | Datatype | Requirements |
| custkey | Integer | Primary Key |
| cusname | variable text, size 25 |  |
| address | variable text, size 40 |  |
| nationkey | Integer | Foreign Key to NATION\_nationkey |
| phone | fixed text, size 15 |  |
| acctbal | fixed text, size 15 |  |
| mktsegment | fixed text, size 10 |  |
| cuscomment | variable text, size 117 |  |

CUSTOMER Table Layout

|  |  |  |
| --- | --- | --- |
| Column Name | Datatype | Requirements |
| nationkey | Integer | 25, Primary Key |
| naname | fixed text, size 25 |  |
| regionkey | Integer | Foreign Key to REGION\_regionkey |
| nacomment | variable text, size 152 |  |

NATION Table Layout

|  |  |  |
| --- | --- | --- |
| Column Name | Datatype | Requirements |
| regionkey | Integer | 5, Primary Key |
| regionname | fixed text, size 25 |  |
| regioncomment | variable text, size 152 |  |

REGION Table Layout

主要数据结构和函数的说明

class Buffer｛

public:

Buffer(){}; //no use

~Buffer(){ free(data\_); }; //for free data\_

Buffer(struct dbSysHead \* head, int fid); //malloc data\_

bool AppendBuffer(const void \* new\_data, int new\_size); //append one record per time

bool writeBuffer(struct dbSysHead \* head, void \* write\_data, int write\_size); //write one page to file(write from buffer to the disk)

bool writeBufferPage(struct dbSysHead \* head, int buffer\_id\_, void \* write\_data, int write\_size); //write one page to file(write from buffer to the other buffer)

public:

struct dbSysHead \* filehead;

long fileID; //logical fid for writeBuffer

long pageID; //start at 0

char \* data\_; //record content written in the form of char

char \* pointer\_; //wirte pointer at current position

int current\_size\_; //current data length

int max\_size\_; //set maxsize as SIZE\_PER\_PAGE

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

功能：

Buffer类，用于批量写数据，可用于将数据写入外存表文件中，也可用于将数据写入新生成的临时表。成员变量data的大小在类的构造时设为缓冲区一页的页面大小，通过一次次调用AppendBuffer函数，将每条记录逐个写入data中，当data达到页面大小时，将data中的内容一次性拷贝到要写入的缓冲区页面。writeBuffer函数是通过调用存储管理的wtFile函数将data内容写入外存中对应的表文件，writeBufferPage函数则是将data中的内容写入另一块buffer，用于临时表的写入。

主要函数的参数，返回值说明

Buffer(struct dbSysHead \* head, int fid);

类的构造函数。当用于写入外存时，fid有意义，表示要写入的表的logical fileid，当用于写入临时表时，fid无意义，我们可以设置其为负数。

bool writeBuffer(struct dbSysHead \* head, void \* write\_data, int write\_size);

writeBuffer函数是通过调用存储管理的wtFile函数将data内容写入外存中对应的表文件。write\_data表示要写入的内容，write\_size表示内容的长度，函数返回值为true or false表示写成功或失败。

bool writeBufferPage(struct dbSysHead \* head, int buffer\_id\_, void \* write\_data, int write\_size);

writeBufferPage函数是将data中的内容写入另一块buffer，用于临时表的写入。buffer\_id表示临时表所在的buffer的id号，write\_data表示要写入的内容，write\_size表示内容的长度。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct eachAttribute{

char attribute\_name\_[A\_NAME\_LENGTH];

int attribute\_type\_;

int attribute\_length\_;

};

int createTable(struct dbSysHead \* head, const char \* relationName, const char \* constructorName, int attributeNum, struct eachAttribute \* attributeList);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

功能：创建一个新表，分配表空间，创建该表的数据字典。

分配表空间调用了存储管理的creatFileSpace函数，创建数据字典时没有将数据字段写入文件。

函数参数、返回值说明：

relationName新表的表名，constructorName创建者的名字，attributeNum属性个数，attributeList一个属性列表，是以数组形式存放每个属性的信息，包括属性名，属性类型，属性长度。函数成功返回值为0，负值代表相应的错误类型。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class RecordCursor

{

public:

RecordCursor(struct dbSysHead \*head, long fid, long recordLength, int bufferID) ;

bool getNextRecord(void \*des);

long getcLogicLocation();

long getcLogicPage();

long getcBufferOffset();

int getBufferID();

private:

struct dbSysHead \*head; //

long fid; // table ID, logical file id

long cLogicPage; // current logical page ID in the table

long filePageNum; // the total number of pages in the table

int bufferID; // the current buffer block

long cBufferPage; // the current page in the buffer

long bufferOffset; // the current offset in the page

long recordLength; // record length

bool start; // is the first record

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

功能：

RecordCursor类用于逐条读取每行记录。

主要函数说明：

RecordCursor(struct dbSysHead \*head, long fid, long recordLength, int bufferID) ;

构造函数，创建类，fid是逻辑表号，recordLength是每条记录的长度，bufferID是表所在的buffer的块号。

bool getNextRecord(void \*des);

得到下一条记录的内容，存放在des中。如果拿到的记录内容为$符，说明该记录已被标记为删除，则递归调用getNextRecord函数，直到拿到的记录不是删除的记录。返回值为true表示成功，false表示读头已越界，已读到表的结尾。

RecordCursorTmp类与RecordCursor类原理相似，区别在于RecordCursor是为读外存中的表文件所用，RecordCursorTmp是为读临时表所用。其实应该用类的继承写RecordCursorTmp类，目前的写法不够好。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int deleteAllTuples(struct dbSysHead \*head, const char \* tableName)

功能：实现delete ＊ from Table这类sql语句。

tableName表名，通过表名查找到表的id，回收表空间，将表的写页数和写位置置为0。返回值为0说明删除成功，返回值为负数说明表名不存在。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int dropTable(struct dbSysHead \*head, const char \* tableName)

功能：实现 drop Table 这类sql语句。

tableName表名，通过表名查找到表的id，回收表空间，删除该表的数据字典（没有写外存）。返回值为0说明删除成功，返回值为负数说明表名不存在。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void getOneRecord(char \*oneRec, relation \* dic)

功能：通过数据字典，解析一行数据，并打印输出。

oneRec 一行内容，dic数据字典。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

long insertOneRecord(struct dbSysHead \*head , int fileID, char \* oneRow)

功能：在表的末尾插入一条记录。

通过调用存储管理的wtFile函数插入记录。并修改表的结束位置，写页数，记录条数等有关信息，

fileID逻辑表号，oneRow一行内容，返回值，pos该行记录的位置（用于添加索引）。其实返回值并没有用到。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int insertOneTuple(struct dbSysHead \*head, const char \* tableName, const char\* content)

功能：实现insert into Table的sql语句。

根据表名找到外存中的表，调用parserOneLineFromFile函数将插入的该行记录的每个属性值封装成一个字符串，再调用insertOneRecord函数将记录内容写入外存表文件。

tableName表名，content插入的内容。返回值0表示插入成功，负数表示失败。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int hashjoin(struct dbSysHead \*head, relation \*temp\_datadic, int old\_relation\_1, int old\_relation\_2, const char \*name);

功能：实现在两表上的哈希连接，要求连接属性的属性名相同，这样的连接才有实际意义。

temp\_datadic临时表数组，old\_relation\_1左表所在的buffer的块号，old\_relation\_2右表所在的buffer的块号，name连接的属性名。返回值new\_relation，新表所在的buffer块号，若返回值为负值，说明失败。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printTempTable(struct dbSysHead \*head, relation \* temp\_data\_dict,int new\_relation)

功能：打印临时表，展示最终结果。

通过RecordCursorTmp的getNextRecord拿到每行记录，调用getOneRecord函数打印输出临时表的信息。

temp\_datadic临时表数组，new\_relation要读取的临时表的buffer块号。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void loaddata(struct dbSysHead \* head, int fid)

功能：批量导入表数据。

fid逻辑表号。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void parserOneLineFromFile(char \* row, char \*rec, relation \* dict\_id)

功能：将每行的用｜分割的一行字符串通过数据字典转换为表中的一行记录。

row原始内容，rec新记录的内容，dicid数据字典指针。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int TableScan(struct dbSysHead \* head, relation \* temp\_datadic, const char \* tableName)

功能：通过表名进行全表扫描，将外存的表文件读入内存的一块buffer。

通过表名查找表的logical fileid，查找哪块buffer未被使用作为新建的临时表的块号，利用RecordCursor逐行读取外存中的表的记录，用buffer类装载，并写入新创建的临时表，新表的临时数据字典用旧表的数据字典拷贝得到。

temp\_datadic临时表数组，tableName表名。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int wtFile( struct dbSysHead \*head, int bufferID, long fid, long pos, long length, void \*des)；

功能：将数据写入指定表中的指定位置。

bufferID是要写入的页面所在的buffer的块号，fid是logical fileid，即表的逻辑编号，pos是表中的偏移量，length是写入内容的长度，des是写入的内容。返回值0表示写入成功。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int rdFile( struct dbSysHead \*head, int bufferID, long fid, long pos, long length, void \*des)

功能：从指定表中的指定位置读取数据。

bufferID是要读入的页面所在的buffer的块号，fid是logical fileid，即表的逻辑编号，pos是表中的偏移量，length是读取内容的长度，des是读取的内容。返回值0表示读成功。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int isAvail(void \*p,char \*funcName,int type);

功能：检查是否正常运行及打印出错信息。

funcName函数名称，type错误类型，返回值为1说明政策运行未出错。exit说明出现异常程序退出。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/