```
1  /*
2  * DBMS Implementation
3  */
4
5  #ifndef _DBMSPROJ_H
6  #define _DBMSPROJ_H
7
8  #include "stdafx.h"
9  #include <fstream>
10 #include <string>
11 #include <stdio.h>
12 #include <iostream>
13 #include <sstream>
14 #include <vector>
15 #include <cstdarg>
16 //#include "bufferOps.h"
17
18 #define TUPLES_PER_ACCOUNT_BLOCK 10
19 #define TUPLES_PER_BRANCH_BLOCK 7
20 #define TUPLES_PER_CUSTOMER_BLOCK 8
21 #define TUPLES_PER_DEPOSITOR_BLOCK 15
22
23 #define NUM_BLOCKS_ACCOUNT 6
24 #define NUM_BLOCKS_BRANCH 1
25 #define NUM_BLOCKS_CUSTOMER 8
26 #define NUM_BLOCKS_DEPOSITOR 4
27
28 #define MAX_RECORDS_PER_BUCKET 20
29
30
31 #define MAX_MEMORY_BLOCKS 5
32
33 //definition of record
34 //structure of records in account table;
35 enum field_code                                                              ↵
     {e_account_number,e_branch_name,e_balance,e_branch_city,e_assets,e_customer_name,e_cust ↵
     omer_street,e_customer_city};
36 inline field_code getfield(std::string const& field)
37 {
38     if (field == "account_number") return e_account_number;
39     if (field == "branch_name") return e_branch_name;
40     if (field == "balance") return e_balance;
41     if (field == "branch_city") return e_branch_city;
42     if (field == "assets") return e_assets;
43     if (field == "customer_name") return e_customer_name;
44     if (field == "customer_street") return e_customer_street;
45     if (field == "customer_city") return e_customer_city;
46 };
47 typedef struct {
48     std::string account_number;
49     std::string branch_name;
50     int balance = 0;
51     bool valid=false; //if set, then the record is valid
52     void setCol(std::string &val1, std::string &val2, std::string &val3)
53     {
54         account_number = val1;
55         branch_name = val2;
56         balance = std::stoi(val3,0);
```

```
57          valid = true;
58      }
59      std::string getCol(std::string const& field)
60      {
61          field_code fields = getfield(field);
62
63          switch (fields)
64          {
65          case e_account_number: return account_number; break;
66          case e_branch_name: return branch_name; break;
67          case e_balance: return std::to_string(balance); break;
68          default: return (" "); break;
69          }
70      }
71
72      void display() {
73          std::cout << account_number << "    " << branch_name << "   " << balance <<
              std::endl;
74      }
75      void project(int num, const char* field, ...)
76      {
77          std::string fields(field);
78          if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
79          va_list arguments;
80          va_start(arguments, field);
81          for (size_t i = 0; i < num - 1; i++)
82          {
83              std::string fields(va_arg(arguments, char*));
84              if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
85          }
86          std::cout << std::endl;
87          va_end(arguments);
88      }
89
90  }account_t;
91
92  //structure of records in branch table;
93  typedef struct {
94      std::string branch_name;
95      std::string branch_city;
96      int assets = 0;
97       bool valid=false; //if set, then the record is valid
98      std::string getCol(std::string const& field)
99      {
100         field_code fields = getfield(field);
101
102         switch (fields)
103         {
104         case e_branch_name: return branch_name; break;
105         case e_branch_city: return branch_city; break;
106         //case 3: return assets; break;
107         default:return (" ");
108             break;
109         }
110     }
111     void setCol(std::string &val1, std::string &val2, std::string &val3)
112     {
113         branch_name = val1;
114         branch_city = val2;
```

```
115         assets = std::stoi(val3, 0);
116         valid = true;
117     }
118     void display() {
119         std::cout << branch_name << "   " << branch_city << "   " << assets << std::endl;
120     }
121     void project(int num, const char* field, ...)
122     {
123         std::string fields(field);
124         if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
125         va_list arguments;
126         va_start(arguments, field);
127         for (size_t i = 0; i < num - 1; i++)
128         {
129             std::string fields(va_arg(arguments, char*));
130             if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
131         }
132         std::cout << std::endl;
133         va_end(arguments);
134     }
135 }branch_t;
136
137 //structure of records in customer table;
138 typedef struct {
139     std::string customer_name;
140     std::string customer_street;
141     std::string customer_city;
142     bool valid=false; //if set, then the record is valid
143     void setCol(std::string &val1, std::string &val2, std::string &val3)
144     {
145         customer_name = val1;
146         customer_street = val2;
147         customer_city = val3;
148         valid = true;
149     }
150     std::string getCol(std::string const& field)
151     {
152         field_code fields = getfield(field);
153         switch (fields)
154         {
155         case e_customer_name: return customer_name; break;
156         case e_customer_street: return customer_street; break;
157         case e_customer_city: return customer_city; break;
158         default:return (" ");
159             break;
160         }
161     }
162     void display() {
163         std::cout << customer_name << " " << customer_street << "   " << customer_city <<
              std::endl;
164     }
165     void project(int num, const char* field, ...)
166     {
167         std::string fields(field);
168         if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
169         va_list arguments;
170         va_start(arguments, field);
171         for (size_t i = 0; i < num - 1; i++)
```

```
172         {
173             std::string fields(va_arg(arguments, char*));
174             if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
175         }
176         std::cout << std::endl;
177         va_end(arguments);
178     }
179 }customer_t;
180
181 //structure of records in depositor table;
182 typedef struct {
183     std::string customer_name;
184     std::string account_number;
185     bool valid=false; //if set, then the record is valid
186     void setCol(std::string &val1, std::string &val2, std::string &val3)
187     {
188         customer_name = val1;
189         account_number = val2;
190         valid = true;
191     }
192
193     std::string getCol(std::string const& field)
194     {
195         field_code fields = getfield(field);
196         switch (fields)
197         {
198         case e_customer_name: return customer_name; break;
199         case e_account_number: return account_number; break;
200         default: return (" "); break;
201         }
202     }
203     void display() {
204         std::cout << account_number << "    " << customer_name<< std::endl;
205     }
206     void project(int num, const char* field, ...)
207     {
208         std::string fields(field);
209         if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
210         va_list arguments;
211         va_start(arguments, field);
212         for (size_t i = 0; i < num - 1; i++)
213         {
214             std::string fields(va_arg(arguments, char*));
215             if (getCol(fields) != (" ")) std::cout << getCol(fields) << "   ";
216         }
217         std::cout << std::endl;
218         va_end(arguments);
219     }
220 }depositor_t;
221
222 template<typename T> struct block_t {
223     unsigned int blockid;
224     unsigned int nreserved;
225     unsigned int maxRecords;
226     bool valid;
227     std::vector<T> entries;
228
229     block_t()
230     {
```

```
231         blockid = 0;
232         nreserved = 0;
233         valid = false;
234         if (std::is_same<T, account_t>::value) maxRecords = 10;
235         if (std::is_same<T, branch_t>::value)  maxRecords = 7;
236         if (std::is_same<T, customer_t>::value)  maxRecords = 8;
237         if (std::is_same<T, depositor_t>::value)  maxRecords = 15;
238         //entries.reserve(maxRecords);
239     }
240
241     void printrecord()
242     {
243         for (unsigned int i = 0; i < entries.size(); i++)
244         {
245             entries[i].display();
246         }
247     }
248
249 };
250
251
252
253
254 template<typename T1, typename T2> struct join_t {
255     T1 rec1;
256     T2 rec2;
257     void display(int num, std::string field, ...)
258     {
259         std::string fields(field);
260         if (rec1.getCol(fields) != (" ")) std::cout << rec1.getCol(fields) << " ";
261         if (rec2.getCol(fields) != (" ")) std::cout << rec2.getCol(fields) << " ";
262         va_list arguments;
263         va_start (arguments, field);
264         for (size_t i = 0; i < num-1; i++)
265         {
266             std::string fields(va_arg(arguments, char*));
267             if (rec1.getCol(fields) != (" ")) std::cout << rec1.getCol(fields) <<"  ";
268             if (rec2.getCol(fields) != (" ") ) std::cout << rec2.getCol(fields)<<"  ";
269         }
270         std::cout << std::endl;
271         va_end(arguments);
272     }
273 };
274
275
276
277 /*
      ------------------------------------------------------------------------------------------
      --------------------------------
278   Functions
279
    ------------------------------------------------------------------------------------------
    ----------------------------
280  */
281
282 template<typename T> void fillTable(char* filename, std::vector<block_t<T>> &table,
      unsigned int numblocks, int degbugmode = 0) {
283
284     std::ifstream file(filename);
285
```

```
286     int rec_counts = 0;
287     int block_counts = 0;
288
289     for (unsigned int i = 0; i < numblocks; i++)
290     {
291         std::string   line;
292         unsigned int index = 0;
293         block_t<T> block;
294
295         while ((index<block.maxRecords) && (std::getline(file, line)))
296         {
297             std::string   val1;
298             std::string   val2;
299             std::string   val3;
300             T rec;
301             std::stringstream   linestream(line);
302             std::string         data;
303             std::getline(linestream, val1, '\t');
304             std::getline(linestream, val2, '\t');
305             std::getline(linestream, val3, '\t');
306
307             rec.setCol(val1, val2, val3);
308             rec.valid = true;
309             block.entries.push_back(rec);
310             rec_counts++;
311             index++;
312         }
313         block.blockid = i;
314         block.nreserved = index;
315         block.valid = true;
316         table.push_back(block);
317         block_counts++;
318     }
319
320     if (degbugmode !=0)
321     {
322
323     printf("%s Table in NYC site has %d records in %d blocks.\n", filename,rec_counts,
          block_counts);
324     for each (block_t<T> block in table)
325     {
326         block.printrecord();
327         printf("\n");
328     }
329     printf("\n");
330     }
331 }
332 template <typename T> void select(std::vector<block_t<T>> &in, std::vector<block_t<T>>
      &out, std::string const & field, std::string const & val,int debugmod=0)
333 {
334     printf("Start processing selection.... \n");
335     block_t<T> buffer[MAX_MEMORY_BLOCKS];
336     //figure out hom many times need to load the blocks for the input relation
337     unsigned int size = ((in.size() + MAX_MEMORY_BLOCKS - 2)) / (MAX_MEMORY_BLOCKS - 1);
338
339     //set the out block pointer to last block of the buffer
340     block_t<T> *bufferOut = buffer + MAX_MEMORY_BLOCKS - 1;
341
342     //process the blocks loaded into the buffer each time
```

```
343     for (unsigned int i = 0; i < size; i++)
344     {
345         int recordcounts = 0;
346         int nblocks=readBlocks<T>(in, buffer, (MAX_MEMORY_BLOCKS - 1), i*
              (MAX_MEMORY_BLOCKS - 1));
347
348         recordPtr start = newPtr(0, buffer->maxRecords);
349         //calculate the total records of the loaded blocks in the buffer
350         unsigned int offset = (nblocks - 1)*buffer->maxRecords + (buffer + nblocks - 1)-
              >nreserved;
351         recordPtr end = newPtr(offset, buffer->maxRecords);
352
353         // starting from the very first record, all valid records in valid blocks are
              hashed
354         for (; start < end; incr(start, buffer->maxRecords))
355         {
356             if (!buffer[start.block].valid) {
357                 start.record = buffer->maxRecords - 1;
358                 continue;
359             }
360             T record = getRecord<T>(buffer, start);
361             if (record.getCol(field) == val)
362             {
363                 recordcounts++;
364                 bufferOut->entries.push_back(record);
365                 bufferOut->nreserved++;
366                 if (bufferOut->nreserved == bufferOut->maxRecords)
367                 {
368                     out.push_back(*bufferOut);
369                     emptyBlock<T>(bufferOut, bufferOut->maxRecords);
370                 }
371             }
372         }
373         printf("%d records are selected... \n",recordcounts);
374     }
375     //if there has any records left in the output buffer, push into the out vector
376     if (bufferOut->nreserved != 0) out.push_back(*bufferOut);
377     printf("End of selection operation.... \n");
378
379 }
380
381 #endif
382
```