Assignment#2 Report

2018007938 김민관

Case 1

- 첫번째 relation의 attribute 인 name, age 와 두번째 relation 의 attribute 인 name, salary를 name을 기준으로 natural join 수행한다.
- **Merge Join algorithm** 을 사용하여 natural join 진행 : 첫번째 db 와 두번째 db는 둘다 이름순으로 이미 파일이 정리되어있다. 이 때, Merge Join 알고리즘을 사용하여, current_block 가 항상 1나씩 증가하므로, 파일 open을 매 파일의 한번만 진행할 수 있으므로, Merge Join이 적합하다.

구현 방식 :

```
for(int i = 0; i < 1000; i++)
{
  current_block[1] = 0;
  block[0].open("./name_age/" + to_string(i) + ".csv");
  for(int j = 0; j < 10; j++)
  {
    getline(block[0], buffer[0]);
    temp0.set_name_age(buffer[0]);
    block[1].open("./name_salary/" + to_string(i) + ".csv");
    for(int k = 0; k < 10; k++)
    {
      getline(block[1], buffer[1]);
      temp1.set_name_salary(buffer[1]);
      if(k >= current_block[1])
      {
        if(temp0.name == temp1.name)
        {
          output << make_tuple(temp0.name, temp0.age, temp1.salary);
          if(current_block[1] != i) current_block[1] += 1;
        }
      }
    }
    block[1].close();
  }
  block[0].close();
}
```

첫번째 파일을 block[0] 에 두번째 파일의을 block[1]에 open 한 후 하나씩 tuple을 비교하며, 두 tuple의 이름을 비교하여 같다면 make_tuple 함수를 사용하여 tuple을 만들고 ouput.csv에 쓴다.

Case2

- 첫번째 relation의 attribute 인 name, age 와 두번째 relation 의 attribute 인 name, salary는 각각 name과 age 순으로 정렬 되어있다. 이를 name을 기준으로 natural join 수행한다.
- **Nested-loop Join algorithm** 을 사용하여 natural join 진행 : 각각의 파일이 모두 age와 salary 순으로 정렬되어있어 name으로는 모두 무작위로 정렬되어 있다. 따라서 각각의 파일을 모두 확인해야 하므로 Nested-loop Join이 open을 가장 적절하게 실행한다.

구현 방식:

```
for(int i = 0; i < 1000; i++)
{
  block[0].open("./name_age/" + to_string(i) + ".csv");
  for(int j = 0; j < 10; j++)
  {
    getline(block[0], buffer[0]);
    temp0.set_name_age(buffer[0]);
    for(int l = 0; l < 1000; l+=10)
    {
```

```
      for(int a = 0; a < 10; a ++) block[a+1].open("./name_salary/" + to_string(l+a) + ".csv");
      for(int a = 0; a < 10; a ++)
      {
        for(int k = 0; k < 10; k++)
        {
          getline(block[a+1], buffer[1]);
          temp1.set_name_salary(buffer[1]);
          if(temp0.name == temp1.name)
          {
            output << make_tuple(temp0.name, temp0.age, temp1.salary);

          }
        }
        block[a+1].close();
      }
    }
  }
  block[0].close();
}
```

name_age의 파일을 block[0]에 open 하고, name_salary의 파일 10개를 block[1 ~ 10]까지 open을 한 후, 각 tuple을 비교하여 이름이 같다면 make_tuple 함수를 사용하여 이를 output.csv 에 저장한다.

Case 3

- 첫번째 relation에는 학생 이름과 1학기 성적, 두번째 relation 에는 학생 이름과 2학기 성적, 세번째 relation 에는 학생 이름과 학번이 저장되어 있다. 이 중에서 1학기 성적과 2학기 성적을 비교하여 성적 향상이 2개 이상인 학생을 뽑아, 학생의 이름과 학번을 output.csv에 저장한다.
- **Hash Join algorithm** 을 사용하여 natural join 진행 : 3개의 db가 존재하는데 이중 1학기 성적이 있는 db와 2학기 성적이 있는 db를 두개의 성적을 비교하는 hash 함수를 통해 hash table을 만들고 이를 3번째 db와 비교하여 학생의 이름과 학번을 output.csv에 저장한다.

구현 방식:

```
int hashFunction(int f_korea, int f_math, int f_english, int f_science, int f_social, int f_history,
            int s_korea, int s_math, int s_english, int s_science, int s_social, int s_history)
{
  int gradeChangeNum = 0;
  if(f_korea > s_korea) gradeChangeNum += 1;
  if(f_math > s_math) gradeChangeNum += 1;
  if(f_english > s_english) gradeChangeNum += 1;
  if(f_science > s_science) gradeChangeNum += 1;
  if(f_social > s_social) gradeChangeNum += 1;
  if(f_history > s_history) gradeChangeNum += 1;
  return gradeChangeNum;
}

// make hash table in buckets
block[11].open("../buckets/hashtable.csv");
if(block[11].fail())
{
  cout << "output file opening fail.\n";
}

for(int i = 0; i < 1000; i++)
{
  block[0].open("./name_grade1/" + to_string(i) + ".csv");
  for(int j = 0; j < 10; j++)
  {
    getline(block[0], buffer[0]);
    temp0.set_grade(buffer[0]);
    for(int l = 0; l < 1000; l+=10)
    {
      for(int a = 0; a < 10; a ++) block[a+1].open("./name_grade2/" + to_string(l+a) + ".csv");
      for(int a = 0; a < 10; a ++)
      {
        for(int k = 0; k < 10; k++)
        {
          getline(block[a+1], buffer[1]);
          temp1.set_grade(buffer[1]);
          if(temp0.student_name == temp1.student_name)
          {
            if(hashFunction(temp0.korean, temp0.math, temp0.english, temp0.science, temp0.social, temp0.history,
```

```
                    temp1.korean, temp1.math, temp1.english, temp1.science, temp1.social, temp1.history) >= 2) block[11] << temp1.stud
          }
        }
        block[a+1].close();
      }
    }
  }
  block[0].close();
}
block[11].close();

//result output

for(int i = 0; i < 1000; i++)
{
  block[0].open("./name_number/" + to_string(i) + ".csv");
  for(int j = 0; j < 10; j++)
  {
    getline(block[0], buffer[0]);
    temp2.set_number(buffer[0]);
    block[1].open("../buckets/hashtable.csv");
    if(block[1].fail()) cout << "open hashtable.csv fail \n";
    while(!block[1].eof())
    {
      getline(block[1],buffer[1]);
      if(buffer[1] == temp2.student_name)
      {
        output << make_tuple(temp2.student_name, temp2.student_number);
        break;
      }
    }
    block[1].close();
  }
  block[0].close();
}
```

bucket 파일에 있는 비어있는 hashtable.csv 를 block[11]에 open을 하고, 1학기 성적과 2학기 성적 파일을 block[0] 과 block[1~10] 에 open 하고, 위에 선언한 hashFunction 을 통해, 1학기 성적에서 2학기 성적으로 향상이 2개 이상인 tuple의 이름을 hashtable.csv에 저장한다. 그 후 다시 이름과 학번이 있는 csv 를 block[0]에 open 하고, hashtable과 비교하여 이름이 같다면, make_tuple 함수를 사용하여 만든 tuple을 output.csv에 저장한다.